

IEEE NNC Student Summer Research Support Program Report

by

Serdar Iplikci
EE Department, Bogazici University, 80815 Bebek, Istanbul, Turkey
iplikcis@boun.edu.tr

This research has been conducted at the University of Idaho Boise Centre in August 2000.

ABSTRACT

This report presents the studies carried out on two modifications suggested in the literature for Levenberg-Marquardt algorithm. The modifications are applicable to feed-forward neural networks. One modification [18], made on performance index, reduces computational complexity of the Levenberg-Marquardt algorithm, while the other one [17], made on calculation of the gradient information, improves convergence rate. These modifications have been performed on several benchmark problems.

1. INTRODUCTION

Slow convergence rate has been the major drawback of the error back-propagation algorithm, although it has accepted as one of the most important milestones in the area of training neural networks. In the literature, some well-known heuristics [1-11], and some second order methods [12-16] can viewed as main approaches proposed so far in order to improve convergence properties of the error back-propagation algorithm. Among second order approaches, the Levenberg-Marquardt algorithm is widely accepted as the most efficient one in the sense of realisation accuracy [14]. It gives a good compromise between the speed of the Gauss-Newton algorithm and the stability of the Steepest Descent method, and consequently it provides a good transition between these methods.

On the other hand, the requirement of large amount of memory while inverting the Jacobian matrix at every weight updates constitutes the main disadvantage of the Levenberg-Marquardt algorithm. The size of the Jacobian matrix is proportional to the number of adjustable parameters in the system. As the dimensionality (number of adjustable parameters) of the network increases, it should be clear that the training would entail costly hardware due to the exponential growth in the computational complexity.

In this report, we focused on two modifications: one is made on the performance index in order to reduce the computational complexity [18], the other one is made on the calculation of the gradient in order to improve the convergence [17].

2. STANDARD LEVENBERG-MARQUARDT ALGORITHM AND ITS MODIFICATIONS

Standard Levenberg-Marquardt algorithm, a variation on the error back-propagation algorithm, provides us with a good switching capability between the Gauss-Newton algorithm and the Steepest Descent method. The quadratic performance index $F(\underline{w})$ to be minimized is sum

of the squares of the error between desired output and actual output for all patterns as given by Eq.(1)

$$F(\underline{w}) = \underline{e}^T \underline{e} \quad (1)$$

In Eq.(1), \underline{e} is the error vector defined by

$$\underline{e} = [e_{1,1} \cdots e_{R,1} \ e_{1,2} \cdots e_{R,2} \cdots e_{1,Q} \cdots e_{R,Q}]^T \quad (2)$$

where $e_{r,q}$ is error between $d_{r,q}$ (desired value for the r^{th} output and q^{th} pattern) and $a_{r,q}$ (actual value of the r^{th} output for q^{th} pattern), Q is the number of patterns, and R is the number of outputs. Moreover, in Eq.(1), \underline{w} is the parameter vector given by

$$\underline{w} = [w_1 \ w_2 \ \dots \ w_M]^T \quad (3)$$

where M is the number of adjustable parameters. Eq.(1) can also be written as

$$F(\underline{w}) = \sum_{q=1}^Q \sum_{r=1}^R (d_{r,q} - a_{r,q})^2 \quad (4a)$$

$$= \sum_{q=1}^Q \sum_{r=1}^R (e_{r,q})^2 \quad (4b)$$

In general, the update rule is

$$\underline{w}_{k+1} = \underline{w}_k + \Delta \underline{w}_k \quad (5)$$

In the Newton's method, adjustable parameters are updated by

$$\Delta \underline{w}_k = -[\underline{H}(\underline{w}_k)]^{-1} \underline{g}(\underline{w}_k) \quad (6)$$

where $\underline{H}(\underline{w}_k)$ is the Hessian matrix given by

$$\underline{H} = \begin{bmatrix} \frac{\partial^2 F(\underline{w}_k)}{\partial w_1^2} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_1 \partial w_M} \\ \frac{\partial^2 F(\underline{w}_k)}{\partial w_2 \partial w_1} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_2^2} & \dots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_2 \partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\underline{w}_k)}{\partial w_M \partial w_1} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_M \partial w_2} & \dots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_M^2} \end{bmatrix} \quad (7)$$

and $\underline{g}(\underline{w}_k)$ is the gradient vector given by

$$\underline{g}(\underline{w}_k) = \left[\frac{\partial F(\underline{w}_k)}{\partial w_1} \quad \frac{\partial F(\underline{w}_k)}{\partial w_2} \quad \dots \quad \frac{\partial F(\underline{w}_k)}{\partial w_M} \right]^T \quad (8)$$

The gradient vector and the Hessian matrix can be written in terms of the Jacobian matrix as

$$\underline{g}(\underline{w}_k) = 2\underline{J}^T(\underline{w}_k)\underline{e}_k \quad (9)$$

and

$$\underline{H}(\underline{w}_k) = 2\underline{J}^T(\underline{w}_k)\underline{J}(\underline{w}_k) + \text{neglected terms} \quad (10)$$

where $\underline{J}(\underline{w}_k)$ is the $RQ \times M$ Jacobian matrix given by

$$\underline{\underline{\mathbf{J}}}(\underline{\mathbf{w}}_k) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \dots & \frac{\partial e_{1,1}}{\partial w_M} \\ \frac{\partial e_{2,1}}{\partial w_1} & \frac{\partial e_{2,1}}{\partial w_2} & \dots & \frac{\partial e_{2,1}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{R,1}}{\partial w_1} & \frac{\partial e_{R,1}}{\partial w_2} & \dots & \frac{\partial e_{R,1}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{1,Q}}{\partial w_1} & \frac{\partial e_{1,Q}}{\partial w_2} & \dots & \frac{\partial e_{1,Q}}{\partial w_M} \\ \frac{\partial e_{2,P}}{\partial w_1} & \frac{\partial e_{2,P}}{\partial w_2} & \dots & \frac{\partial e_{2,P}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{R,Q}}{\partial w_1} & \frac{\partial e_{R,Q}}{\partial w_2} & \dots & \frac{\partial e_{R,Q}}{\partial w_M} \end{bmatrix} \quad (11)$$

Afterwards, substitution of (9) and (10) into (6) yields the update rule for Gauss-Newton method, and the weight updates are calculated by

$$\Delta \underline{\mathbf{w}}_k = -[\underline{\underline{\mathbf{J}}}^T(\underline{\mathbf{w}}_k)\underline{\underline{\mathbf{J}}}(\underline{\mathbf{w}}_k)]^{-1}\underline{\underline{\mathbf{J}}}^T(\underline{\mathbf{w}}_k)\underline{\mathbf{e}}_k \quad (12)$$

Whereas the neglected terms in Eq.(10) may cause some accuracy errors in the calculation of the Hessian, the most significant advantage of the Gauss-Newton method over Newton's method is the elimination of the necessity of calculation of the second derivatives. Furthermore, the fact that the matrix $[\underline{\underline{\mathbf{J}}}^T(\underline{\mathbf{w}}_k)\underline{\underline{\mathbf{J}}}(\underline{\mathbf{w}}_k)]$ may be singular constitutes the main disadvantage of the Gauss-Newton method.

In the Levenberg-Marquardt algorithm, the singularity problem in the Gauss-Newton method is overcome by introducing an additional term, which as well provides a good switching between the Steepest Descent and the Gauss-Newton method. For standard Levenberg-Marquardt algorithm, adjustable parameters are updated by

$$\Delta \underline{\mathbf{w}}_k = -[\underline{\underline{\mathbf{J}}}^T(\underline{\mathbf{w}}_k)\underline{\underline{\mathbf{J}}}(\underline{\mathbf{w}}_k) + \mu_k \underline{\underline{\mathbf{I}}}]^{-1}\underline{\underline{\mathbf{J}}}^T(\underline{\mathbf{w}}_k)\underline{\mathbf{e}}_k \quad (13)$$

where μ is the learning rate, $\underline{\underline{\mathbf{I}}}$ is identity matrix [19]. During training process the learning rate μ is incremented or decremented by a scale at weight updates. As the learning rate draws closer to zero, the Levenberg-Marquardt algorithm approaches the Gauss-Newton method, while it approaches the Steepest Descent algorithm as the learning rate takes large values. This is shown in Figure 1 [20].

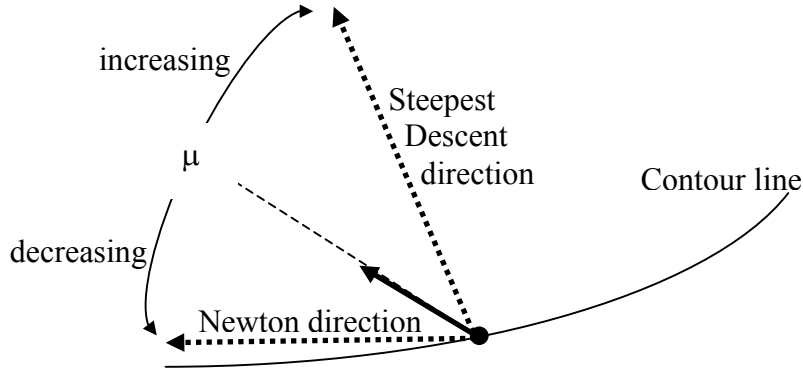


Figure 1. Transition between the Steepest Descent and the Gauss-Newton

Although the Levenberg-Marquardt algorithm gives a good compromise between those methods, its main disadvantage, as can be seen from Eq.(13), is the necessity of computation of $[\underline{\underline{J}}^T(\underline{\underline{w}}_k)\underline{\underline{J}}(\underline{\underline{w}}_k) + \mu_k\underline{\underline{I}}]^{-1}$ square matrix at every weight updates, the dimension of which is $M \times M$.

In [18], one modification on the performance index is proposed in order to reduce the above-mentioned computational complexity, where the performance index given by Eq.(4a) is replaced with the performance index given by Eq.(14),

$$F(\underline{\underline{w}}) = \sum_{r=1}^R \left[\sum_{q=1}^Q (d_{r,q} - a_{r,q})^2 \right]^2 \quad (14)$$

The new performance index can also be written in a quadratic form :

$$F(\underline{\underline{w}}) = \underline{\underline{\hat{e}}}^T \underline{\underline{\hat{e}}} \quad (15)$$

where $\underline{\underline{\hat{e}}}$ is the new error vector $\underline{\underline{\hat{e}}} = [\hat{e}_1 \hat{e}_2 \dots \hat{e}_R]^T$, and $\hat{e}_r = \sum_{q=1}^Q (d_{r,q} - a_{r,q})^2$, for $r = 1, \dots, R$.

It can be observed that the continuity requirements are still preserved and that the new measure can well be considered as a measure of similarity between the desired and the produced patterns. By this modification, proposed in [18], the new update rule is then written by,

$$\Delta \underline{\underline{w}}_k = - \left[\frac{1}{\mu_k} \underline{\underline{I}} - \frac{1}{\mu_k^2} \underline{\underline{J}}^T(\underline{\underline{w}}_k) \left(\underline{\underline{I}} + \frac{1}{\mu_k} \underline{\underline{J}}(\underline{\underline{w}}_k) \underline{\underline{J}}^T(\underline{\underline{w}}_k) \right)^{-1} \underline{\underline{J}}(\underline{\underline{w}}_k) \right] \underline{\underline{J}}^T(\underline{\underline{w}}_k) \underline{\underline{\hat{e}}} \quad (16)$$

where $\underline{\underline{J}}$ is the new Jacobian matrix, the size of which is now $R \times M$. Consequently, in the new update rule the size of the matrix to be inverted becomes $R \times R$. In most neural network applications R is less than M .

Another modification investigated during the study is on the gradient computation of the sigmoidal activation function, which is proposed in [17]. This modification aims at improving the slow asymptotic convergence rate of the error-back propagation algorithm by using the slope of the line connecting the output value and the desired value instead of using derivative of the activation function as the gradient information. In the limit case that the output value approaches the desired value, the calculated slope becomes very near to the calculated derivative of the activation function, and then both algorithms become identical [17].

3. EXAMPLES AND SIMULATION RESULTS

The modifications in [17-18] have 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. Figures 2 to 5 illustrate the error of standard Levenberg-Marquardt and modified Levenberg-Marquardt for the same initial weights for the benchmark problems. Table I summarizes the performance of the modified algorithm compared to standard Levenberg-Marquardt algorithm with respect to computational complexity, while Table II shows a comparison of algorithms with respect to average number of iterations for convergence. As can be seen from Table II, using the slope instead of the derivative yields better performance in the number of average iterations for meeting the prescribed convergence criterion.

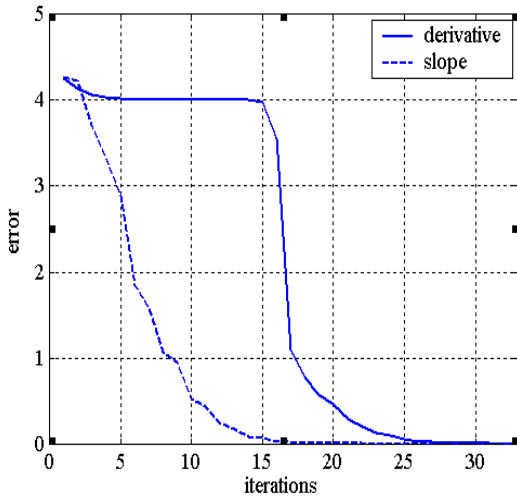


Figure 2. XOR problem with 2 hidden neurons

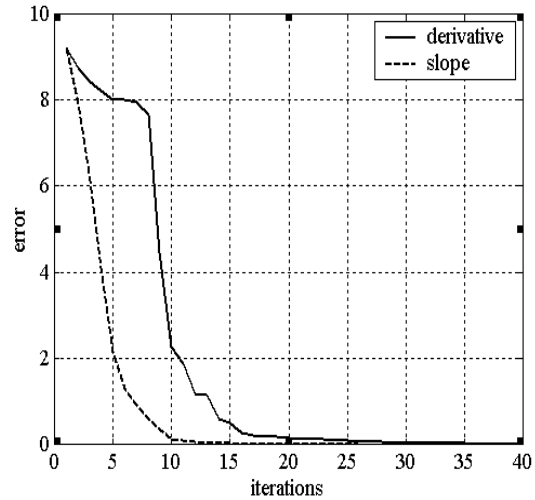


Figure 3. Parity 3 problem with 2 hidden neurons

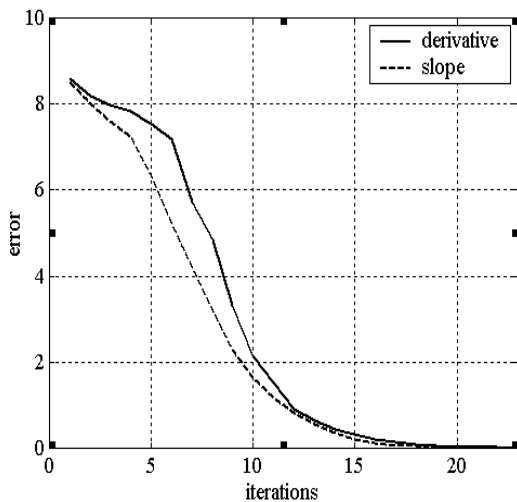


Figure 4. Parity 3 problem with 3 hidden neurons

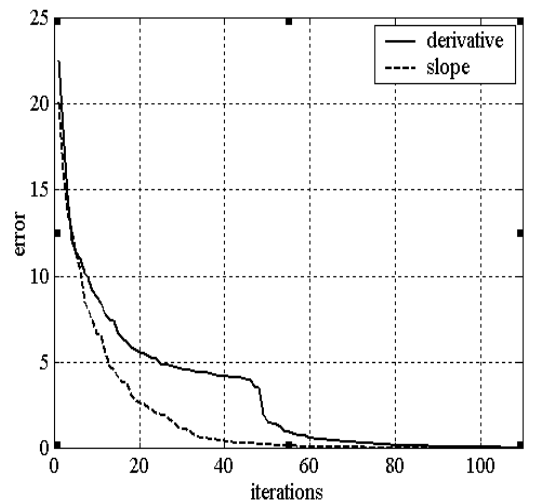


Figure 5. Parity 4 problem with 6 hidden neurons

Table I. Comparison of the algorithms with respect to average number of floating points in MatLab for convergence

		Standard LM	Modified LM
XOR	# of hidden neurons	2	2
	Average # of flops	168972	116991
Parity-3	# of hidden neurons	2	2
	Average # of flops	507397	336113
	# of hidden neurons	3	3
	Average # of flops	631883	370205
Parity-4	# of hidden neurons	6	6
	Average # of flops	16.6913 10 ⁶	6.63997 10 ⁶

Table II. Comparison of the algorithms with respect to average number of iterations for convergence

		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

6. CONCLUSIONS

In this study, some modifications on training feed-forward neural networks have been investigated and have been tested on some benchmark problems. One modification [18] significantly reduces computational complexity by introducing a different performance index, while other one [17] improves convergence rate with respect to the number of average iterations for convergence by introducing a different tool for gradient computation.

REFERENCES

- [1] Bello, M. G., "Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks", IEEE Trans. on Neural Networks, 3, 864, 1992.
- [2] Samad, T., "Back-propagation Improvements Based on Heuristic Arguments", Proceedings of International Joint Conference on Neural Networks, Washington, 1, 565, 1990.
- [3] Oh, S.-H. and S.-Y. Lee, "A New Error Function at Hidden Layers for Fast Training of Multilayer Perceptrons", IEEE Tr. on Neural Networks, 10 (4), 960, 1999.

pp. 960-963. IEEE Transactions on Neural Networks, volume 10 (1999), number 4

- [4] Sperduti, A. and Starita, A., "Speed up Learning and Network Optimization with Extended Back-propagation", *Neural Networks*, 6, 365, 1993.
- [5] Van Ooten, A. and Nienhuis, B., "Improving the Convergence of the Back-propagation Algorithm", *Neural Networks*, 5, 465, 1992.
- [6] Reidmiller, M. and Braun, H., "A Direct Adaptive Method for Faster Back-propagation Learning: The RPROP Algorithm", *Proc. ICNN*, San Francisco, 1993.
- [7] Jacobs, R. A., "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, vol. 1, no.4, pp. 295, 1988.
- [8] Tollenaere, T., "SuperSAB: Fast Adaptive Back-propagation with Good Scaling Properties", *Neural Networks*, 3, 561, 1990.
- [9] Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T. and Alkon, D. L., "Accelerating the Convergence of the Back-propagation Method", *Biological Cybernetics*, 59, 257, 1988.
- [10] Yu, X. H., Chen, G. A. and Cheng, S. X., "Dynamic Learning Rate Optimization of the Back-propagation Algorithm", *IEEE Tr. On Neural Networks*, 6 (3), 669, 1995.
- [11] Magoulas G.D., Vrahatis M.N. and Androulakis G.S., "Improving the convergence of the back-propagation algorithm using learning rate adaptation methods", *Neural Computation*, 11, 1769-1796, 1999.
- [12] Battiti, R., "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method", *Neural Computation*, 4(2), 141, 1992.
- [13] Charalambous, C., "Conjugate Gradient Algorithm for Efficient Training of Artificial Neural Networks", *IEE Proceedings*, 139(3), 301, 1992.
- [14] Hagan, M. T. and Menhaj, M., "Training Feed-forward Networks with the Marquardt Algorithm", *IEEE Tr. on Neural Networks*, 5(6), 989, 1994.
- [15] Johansson, E. M., Dowla, F. U. and Goodman, D. M., "Back-propagation Learning for Multi-layer Feed-forward Neural Networks Using the Gradient Conjugate Method", *Int.'l J. of Neural Systems*, 2, 291, 1992.
- [16] Kinsella, J. A., "Comparison and Evaluation of Variants of the Conjugate Gradient Methods for Efficient Training in Feed-forward Neural Networks with Backward Error Propagation", *Neural Networks*, 3(27), 1992.
- [17] Torvik, L. and Wilamowski, B. M., "Modification of the Back-propagation 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.
- [18] Wilamowski, B. M., Chen, Y. and Malinowski, A. "Efficient Algorithm for Training Neural Networks with One Hidden Layer", *International Joint Conference on Neural Networks'1999 – IJCNN'99*, Washington DC, July 10-16 1999, pp. 1725-1728, 1999.
- [19] Hagan, M. T., Demuth, H. B. and Beale, M., "Neural Network Design", PWS Publishing Company, Boston, MA, 1996.
- [20] Jang, J. S. R., Sun, C. T. and Mizutani, E., "Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence", *MatLab Curriculum*

Series, Prentice-Hall International Inc., NJ, 1997.