A NEW DAMPING STRATEGY OF LEVENBERG-MARQUARDT ALGORITHM FOR MULTILAYER PERCEPTRONS

Young-tae Kwak, Ji-won Hwang, Cheol-jung Yoo[*]

Abstract: In this paper, a new adjustment to the damping parameter of the Levenberg-Marquardt algorithm is proposed to save training time and to reduce error oscillations. The damping parameter of the Levenberg-Marquardt algorithm switches between a gradient descent method and the Gauss-Newton method. It also affects training speed and induces error oscillations when a decay rate is fixed. Therefore, our damping strategy decreases the damping parameter with the inner product between weight vectors to make the Levenberg-Marquardt algorithm behave more like the Gauss-Newton method, and it increases the damping parameter with a diagonally dominant matrix to make the Levenberg-Marquardt algorithm act like a gradient descent method. We tested two simple classifications and a handwritten digit recognition for this work. Simulations showed that our method improved training speed and error oscillations were fewer than those of other algorithms.

Key words: *Levenberg-Marquardt algorithm, damping parameter, Gauss-Newton method, error backpropagation*

Received: January 6, 2011
Revised and accepted: July 12, 2011

1. Introduction

The error backpropagation (EBP) algorithm for multi-layer perceptions (MLP) has been used in many applications since it was proposed by many researchers in the 1980s [1, 2]. Because it utilizes a gradient descent method for minimizing a mean squared error, it can take a long time to escape from some flat error surfaces that make EBP slow. Recently, many people have proposed some heuristic modifications to improve the training speed of EBP. In these methods, there are adding momentum, variable learning rates [3, 4, 5, 6], scaling parameters [7, 8], variable step search algorithms [9] and so on. Besides, some layer-by-layer algorithms that train each hidden layer after defining an error function of its layer have

[*]Young-tae Kwak, Ji-won Hwang, Cheol-jung Yoo
Department of Information Technology, Chonbuk National University, Dukjin-dong, Dukjin-gu, Jeonju, Chonbuk, 561-756, Korea, E-mail: {ytkwak, hwangj, cjyoo}@jbnu.ac.kr

been suggested [10, 11, 12]. However, these heuristic approaches need additional parameters, and selecting improper parameters can influence the performance of EBP in complicated applications.

To overcome the slow convergence of EBP and its variations, some researchers have adapted numerical optimizations [13, 14] used in optimization theory which calculate the second derivatives of error functions. There are the Conjugate Gradient (CG) method [15], the Quasi-Newton method [16], the Gauss-Newton method and the Levernberg-Marquardt (LM) algorithm [17] in such optimization. The numerical optimizations can train MLP faster than EBP does owing to approximating the second derivatives instead of calculating them directly. However, the CG algorithm has to conduct golden section search and be reset after passing a fixed number of iterations. The LM algorithm must also calculate a Jacobian matrix to approximate the second derivatives. Saving the Jacobian matrix and calculating its inverse have been critical problems. To solve these problems, Costa [18] restricted the norm of weight vectors to speed the LM algorithm up. Xu [19] used the rank deficiency of a Jacobian matrix to prune MLP with the LM algorithm. Lera [20, 21] also proposed a way to train local nodes of MLP to save both memory required and expensive operations of the LM algorithm.

Generally, the LM algorithm can be regarded as a combination of a gradient descent method and the Gauss-Newton method. The alternation between a gradient descent method and the Gauss-Newton method is called a damping strategy and is controlled by a damping parameter. If the damping parameter is large, it makes the Jacobian matrix diagonally dominant, and the LM algorithm updates weights like a gradient descent method. But the magnitude of updated weights is so small that it takes a long time to train MLP. On the other hand, if the damping parameter is very small, the LM algorithm updates weights by solving the normal equations of a Jacobian matrix like the Gauss-Newton method. So the LM algorithm can train MLP faster than EBP algorithm. This property is especially needed to converge quickly when a weight vector is close to its target vector. Thus, the damping parameter becomes an important factor to speed up the LM algorithm and reduce error oscillations. However, the methods for altering a damping parameter are not rich, so we propose a new strategy altering a damping parameter.

Until now, only a few methods for a damping parameter have been proposed. Lampton [22] used both additive damping and multiplicative damping. He showed that additive damping can get better results than multiplicative damping. Hagan [17, 23] also proposed the LM algorithm using additive damping and implemented Matlab Toolbox with it. But Hagan's method still makes error oscillations and Matlab Toolbox does not include the LM's trial weight updates in the number of iterations. Chen [24] proposed a variable decay rate considering the linear changes to the logarithm of error function, but its speed is slower in the beginning of training. Amir [25] proposed a very simple method that a damping parameter is multiplied by a squared error function, but it makes error oscillations more frequent. Yamashita [26] and Fan [27] showed that if $\parallel F(x) \parallel$ provides a local error bound for the system of nonlinear equation $F(x) = 0$, the damping sequences generated by $\mu = \parallel F(x) \parallel^2$ or $\mu = \parallel F(x) \parallel^{[1,2]}$ converge to its solution quadratically. These papers proved that a damping parameter should be varied along both error function and the distance between a current error and a local minimum.

Therefore, we will propose a new damping strategy that uses the inner product of weights to decrease a damping parameter, and employs a diagonally dominant matrix to increase it. The inner product of weight vectors can prevent error function from oscillating in the early training, especially when the direction of a weight vector is different from its target weight direction. The diagonally dominant matrix can save iterations by making a Jacobian matrix to be positive definite. Unlike the traditional damping parameter with a fixed decay rate, our approach uses a variable decay rate adjusted by the direction of weight vectors.

In our simulation, we tested both two simple classifications and a somewhat complicated recognition. The result of simulations confirmed that our approach could improve training speed and reduce error oscillations. Especially in the problems with huge weight space, our method performed a fast and high-quality convergence with fewer error oscillations. The rest of this paper is organized as follows. Section 2 reviews the LM algorithm and introduces related works. In Section 3, we propose a new damping strategy for converging the LM algorithm quickly. The experimental results and analyses are shown in Section 4. Finally, we will give our conclusion in Section 5.

2. Problem Formulation

This section briefly describes the LM algorithm and its problem. We will also review some existing methods on damping parameter.

2.1 Levenberg-Marquardt algorithm

In spite of the large memory required and expensive operations, the LM algorithm is estimated to be much faster than other algorithms if the size of MLP is not very large. The error function to be minimized, the sum of squared errors for a weight vector $\mathbf{w}$, is defined as

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

where $\mathbf{w} = [w_1, w_2, \cdots, w_N]^T$ 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. Eq. (1) can be written as

$$\begin{gathered} F(\mathbf{w}) = \mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w}) \qquad (2)\\ \mathbf{e} = [e_{11}\cdots e_{K1}, e_{12}\cdots e_{K2}, \cdots, e_{1P}\cdots e_{KP}]^T \\ e_{kp} = d_{kp}-o_{kp},\;\; k=1\cdots K,\;\; p=1\cdots P. \end{gathered}$$

When considering a quadratic approximation of the error function, the weight update of a Newton's method is

$$\Delta\mathbf{w} = -[\mathbf{H}(\mathbf{w})]^{-1}\mathbf{g}(\mathbf{w}), \tag{3}$$

where $\mathbf{H}(\mathbf{w})$ is the Hessian matrix and $\mathbf{g}(\mathbf{w})$ is the gradient vector in relation to a weight vector $\mathbf{w}$. With the approximation to the Hessian matrix and the gradient vector, the LM method is deduced:

$$\begin{aligned} \mathbf{H}(\mathbf{w}) &= \mathbf{J}(\mathbf{w})^T\mathbf{J}(\mathbf{w}) + S(\mathbf{w}) \\ \mathbf{g}(\mathbf{w}) &= \mathbf{J}(\mathbf{w})^T\mathbf{e}(\mathbf{w}), \end{aligned} \tag{4}$$

where $\mathbf{J}(\mathbf{w})$ is the Jacobian matrix and $S(\mathbf{w})$ is defined as

$$S(\mathbf{w}) = \mathbf{e}(\mathbf{w})^T \cdot \bigtriangledown^2\mathbf{e}(\mathbf{w}),\ \ \bigtriangledown^2 e_{kj} = \frac{\partial^2 e_{kp}}{\partial w_k \partial w_j}. \tag{5}$$

In the Gauss-Newton method, the term $S(\mathbf{w})$ can be left unconsidered. Finally, the LM's weight update is obtained as a modification of the Gauss-Newton method.

$$\begin{aligned} \Delta\mathbf{w} &= -\left[\mathbf{J}(\mathbf{w})^T\mathbf{J}(\mathbf{w}) + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{w})\mathbf{e}(\mathbf{w}) \\ &= -\left[\mathbf{D}(\mathbf{w})\right]^{-1}\mathbf{g}(\mathbf{w}), \end{aligned} \tag{6}$$

where $\mu > 0$ is called a damping parameter and $\mathbf{I}$ is the identity matrix. The damping parameter is adjusted according to the evaluation of $F(\mathbf{w})$. If $F(\mathbf{w})$ is lower than a trial error $F(\mathbf{w}_{trial})$ that is calculated by the new weight, μ is divided by a factor $\beta(0 < \beta < 1)$ called decay rate. Reversely, if $F(\mathbf{w})$ is higher, μ is multiplied by β and the new weight update is accepted.

$$\mu = \begin{cases} \dfrac{\mu}{\beta} & \text{if } F(\mathbf{w}_{trial}) \geq F(\mathbf{w}) \\ \mu \cdot \beta & \text{if } F(\mathbf{w}_{trial}) < F(\mathbf{w}) \end{cases}. \tag{7}$$

The traditional damping strategies used until now have used a fixed decay rate to control the damping parameter. That is, after a decay rate is initialized as a constant, it does not change. Hagan's algorithm and Matlab Toolbox also use the fixed decay rate that set $\beta = 0.1$ as a default value. The fixed decay rate does not reflect the direction and agreement between weight vectors to a damping parameter. It just depends on an error function. Like Eq. (7), the decreasing μ by only evaluating errors can make error function oscillate when the error function is partially approximated to a quadratic error especially in the beginning of training. The value of μ close to 0 can also get an error function to oscillate except for the end of training. These error oscillations take more time to converge and raise a possibility of training failure.

The LM algorithm can be thought of as a combination of a gradient descent method and the Gauss-Newton method like Fig. 1. The LM algorithm with a large μ trains like a gradient descent method so that it can guarantee to converge but its speed is slow. In the case that μ is small, the LM algorithm becomes the Gauss-Newton method that converges quickly. Thus, we expect the LM algorithm to ideally train a nonlinear problem as follows. The algorithm plays the role of a gradient descent method in the beginning of training when the direction of weight vectors is not consistent or when the error function is not perfectly quadratic, and gradually performs the Gauss-Newton method when the weight vector is close to the neighborhood of a target vector. For the ideal LM algorithm, we need a

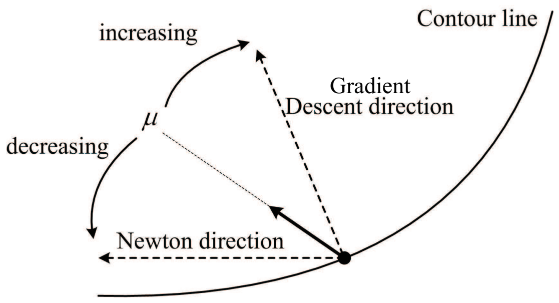

Fig. 1 *Combination of gradient descent method and Gauss-Newton method.*

new method that can decrease the damping parameter a little in the beginning of training rather than in the end of training. Therefore, we propose a variable decay rate for a damping parameter to adapt according to how much a training progresses. The proposed method also uses a diagonally dominant matrix for increasing the damping parameter instead of a fixed decay rate.

2.2 Review of related work

Chen [24] proposed a variable decay rate to prevent $F(\mathbf{w})$ from oscillating in the end of training when the error function $F(\mathbf{w})$ approached to a target error. Here F is the current error, F_0 the initial error, and F_{min} the target error.

$$\beta = \frac{\log(F) - \log(F_{min})}{\log(F_0) - \log(F_{min})} \times 0.8 + 0.1. \tag{8}$$

Chen's disadvantage is that the training speed is slow when it is at the start. We expect the LM algorithm to train like a gradient descent method at its initial training so that β has to be small to increase μ rapidly. But in the initial case $F = F_0$, β is 0.9 and larger than Hagan's $\beta(0.1)$, this β makes the LM algorithm slow and we could find these results in our simulation.

Yamashita and Fukushima [26] established an attractive quadratic convergence result $\mu = \parallel F(\mathbf{w}) \parallel^2$ for the LM algorithm without nonsingularity assumption. Fan and Yuan [27] also extended the result of Yamashita and Fukushima to using $\mu = \parallel F(\mathbf{w}) \parallel^{\delta}$, where $\delta \in [1, 2]$ as a damping parameter. These methods converge to a solution quickly when a current weight vector approximates closely to a target vector. But it is difficult to establish a criterion on whether a current error is close to its target or not.

Amir [25] considered that a damping parameter shows the state of training and proposed Eq. (9). In the beginning of training when the LM algorithm does not train enough, $\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w})$ is so large that it makes the damping parameter too large. In the end of training when the target error is close to 0, $\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w})$ is so small that it makes the parameter too small. So, Amir's method rapidly changes its parameters along $\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w})$ and induces irregular damping parameters. It has also some difficulty in setting the maximum and minimum of damping parameters.

$$\mu = \mu\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w}) \tag{9}$$

3. Strategy of Damping Parameter

We will describe our method by dividing two cases, increasing a damping parameter and decreasing a damping parameter.

3.1 Decreasing damping parameter

We expect the LM algorithm to train like a gradient descent method just after it starts, because in general the solutions of error function are far from starting points or unknown. There is also a possibility for a Jacobian matrix not to approximate error function exactly. In this beginning of training, decreasing a damping parameter with a fixed decay rate like Eq. (7) invokes an oscillating error function. Even though a trial error is reduced, we cannot be sure that the weight direction is optimal. Therefore, we introduce the weight inner product between a current weight vector and a trial weight vector to maintain a weight direction to be constant.

We define the weight inner product as

$$\begin{aligned} wip &= \frac{\mathrm{dot}(\mathbf{w}_{trial}, \mathbf{w})}{\parallel \mathbf{w}_{trial} \parallel \parallel \mathbf{w} \parallel} && (-1 \leq wip \leq 1) \\ \mu &= \mu \cdot \beta^{wip} && \text{if } F(\mathbf{w}_{trial}) < F(\mathbf{w}), \end{aligned} \tag{10}$$

where $\mathbf{w}_{trial}$ is the new weight vector updated by Eq. (6), and β is fixed and generally set as 0.1 in many algorithms such as Hagan's method. Though β is fixed, the term, β^{wip}, plays the role of a variable decay rate because wip is varied to the weight inner product. The weight inner product has the following properties: in the beginning of training, the wip can be very small because the direction of an initialized weight vector might be completely different from that of a trial weight vector. This small wip increases μ and makes the LM algorithm behave like a gradient descent method especially in the early training. However, as the training evolves, the direction of an initial weight vector gradually accords with the direction of its target weight vector. So, the wip approaches to 1. The fact that wip is equal to about 1 means that μ is gradually close to 0, and the LM algorithm can perform the Gauss-Newton method.

Our decay rate, β^{wip}, is a variable decay rate depending on the weight inner product rather than the fixed decay rate like Hagan's method. Replacing Eq. (7) with Eq. (10) can prevent a damping parameter from decreasing when the error function is not quadratic, namely, the error is far from its target error. If wip is negative, that is, when the direction of a current weight vector is opposite to the direction of an updated weight vector, our method can increase a damping parameter unlike Hagan's method even though the error function evaluated by a trial weight vector is reduced.

3.2 Increasing damping parameter

Let us consider the case when we have to increase a damping parameter because a trial error is greater than the current error. When $\mathbf{D}(\mathbf{w})$ in Eq. (6) is singular or not positive definite, the error function increases and we have to increase the damping parameter. Therefore, we have to increase μ until $\mathbf{D}(\mathbf{w})$ is not singular or positive definite. To solve these problems, we use a strictly dominant matrix. A matrix is said to be strictly diagonally dominant if in every row of the matrix the magnitude of the diagonal entry in a row is larger than the sum of the magnitude of all other (non-diagonal) entries in that row. More precisely, the matrix **A** is diagonally dominant if

$$\mathbf{A} = a_{ij} \quad \text{if } |a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{for all } i. \tag{11}$$

A strictly diagonally dominant matrix is non-singular. This result is known as the Levy-Desplanques theorem. A Hermitian (Hessian) diagonally dominant matrix with real non-negative diagonal entries is positive semi-definite. So we will make $\mathbf{D}(\mathbf{w})$ in Eq. (6) diagonally dominant.

Firstly, we check whether $\mathbf{D}(\mathbf{w})$ is positive definite or not by Cholesky factorization. If it is positive, we continue Hagan's method. Otherwise, we make $\mathbf{D}(\mathbf{w})$ diagonally dominant like Eq. (11) and choose the minimum of diagonal entities as a new μ. This procedure can save the iterations needed to increase a damping parameter until $\mathbf{D}(\mathbf{w})$ is positive definite.

```
if $\mathbf{D}(\mathbf{w})$ is positive definite
    $\mu = \mu/\beta$                                              (12)
else
    make $\mathbf{D}(\mathbf{w})$ diagonally dominant
    $\mathbf{D} = d_{ij}, \quad d_{ii} = \sum_{j \neq i} |d_{ij}| \quad \text{for all } i$
    $\mu = \min(d_{ii}) \quad \text{for all } i$
```

Though $\mathbf{D}(\mathbf{w})$ of Eq. (6) is positive definite, the reduction of error function cannot be fully guaranteed because $\mathbf{D}(\mathbf{w})$ may not be modeling error functions exactly. So we need to increase μ repeatedly.

The proposed LM algorithm can be summarized as follows:

1. Initialize the weights and parameter μ, β
   (set $\mu = 0.1, \beta = 0.1$ like Hagan's fixed rate)
2. Stop if the number of iteration exceeds the maximum iteration or $F(\mathbf{w})$ is less than a desired error
3. After passing all training data, compute the sum of squared errors over all inputs, $F(\mathbf{w})$
4. Compute the Jacobian matrix $\mathbf{J}(\mathbf{w})$
5. Solve Eq. (6) to obtain the weight change $\Delta\mathbf{w}$
6. Recompute the sum of squared errors $F(\mathbf{w}_{trial})$ using $\mathbf{w}_{trial} = \mathbf{w} + \Delta\mathbf{w}$ after passing all training data again, and judge
   IF $F(\mathbf{w}_{trial}) < F(\mathbf{w})$ in step 3 THEN

```
            $wip = \frac{\mathrm{dot}(\mathbf{w}_{trial}, \mathbf{w})}{\| \mathbf{w}_{trial} \| \| \mathbf{w} \|} \quad (-1 \leq wip \leq 1)$
            $\mu = \mu \cdot \beta^{wip}$    if $F(\mathbf{w}_{trial}) < F(\mathbf{w})$
go back to step 2
ELSE
            if $\mathbf{D}(\mathbf{w})$ is positive definite
                $\mu = \mu / \beta$
            else
                make $\mathbf{D}(\mathbf{w})$ diagonally dominant
                $\mathbf{D} = d_{ij}, \quad d_{ii} = \sum_{j \neq i} |d_{ij}|$   for all $i$
                $\mu = \min(d_{ii})$   for all $i$
go back to step 2
```

4. Experimental results

We tested two simple classifications and a more complicated problem: an iris classification, a wine classification and a handwritten digit recognition. Iris and wine classifications were cited from UCI Machine Learning Repository [28]. A handwritten digit recognition was done with CEDAR data from Hull [29]. We used MLP with one hidden layer and *tanh* function for activation function instead of logistic sigmoid function.

Our LM algorithm was implemented with Matlab scripts not using Matlab NN Toolbox. There are three main reasons for not using the Toolbox. It does not count trial iterations. That is, if a trial training error, $F(\mathbf{w}_{trial})$, is greater than the current one $F(\mathbf{w})$, the Toolbox disregards (cancels) the weight vectors updated by trial weights until it obtains the correct weight vectors that do not increase the training error. Thus, we cannot measure how many epochs were spent for increasing the damping parameter. In the following results, all iterations included the number of trial weight updates.

Secondly, to compare our method with two other methods, we had to set the initial weight vectors randomly and equally. But the Toolbox does not have a function to load the saved initial weights for making the same conditions of other algorithms. So we had to implement our method with Matlab scripts language. Thirdly, when using the Toolbox, we first have to create a feed-forward network with **newff** function. To speed training up, the **newff** function uses two default functions (**traingdx**, **learngdm**) that update weights and biases according to gradient descent momentum and adaptive learning rate. However, our implementation did not use these techniques because we could not find out how such functions affect damping parameters.

To measure each algorithm's performance, we used mean squared error function like the following Eq. (13), where P is the number of input patterns.

$$F(\mathbf{w}) = \frac{1}{P}\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w}) \tag{13}$$

Number of	Average epoch			Average time (second)			Number of failures		
hidden nodes	Hagan	Our Way	Chen	Hagan	Our Way	Chen	Hagan	Our Way	Chen
5	156.87	88.4	177.23	16.66	7.84	17.8	11	4	11
6	122	100.3	174.83	12.08	9.58	17.3	8	6	11
7	89.7	70.07	138.57	7.8	6.07	12.87	3	3	7
8	115.03	61.07	143.4	11.31	5	13.13	7	2	6
9	93.03	72.6	161.5	8.77	6.32	16.28	5	2	10

Tab. I *Iris Results.*

Number of	Average epoch			Average time (second)			Number of failures		
hidden nodes	Hagan	Our Way	Chen	Hagan	Our Way	Chen	Hagan	Our Way	Chen
4	14.47	12.73	64.1	0.93	0.88	4.39	0	0	1
5	38.6	26.4	88.57	3.86	2.42	7.91	2	1	4
6	28.03	24.9	75.57	2.61	2.43	6.23	1	1	2
7	33.43	22.67	78.53	3.48	2.26	6.3	2	1	1
8	33.17	14.07	80.97	3.52	1.04	7.43	2	0	3

Tab. II *Wine Results.*

In the results of Tab. I, II and III, we tried MLP for 30 runs according to each number of hidden nodes to get more general results. Our stopping criterion is not classification accuracy but training error. The criterion to stop training was that the maximum epoch was 300 or $F(\mathbf{w})$ in Eq. (13) was less than 0.01. If the iteration of a training exceeded the maximum, we considered the training as a failure like Tables. To compare fairly with other methods, we equally initialized the initial weights. In addition, our simulation was run in standard alone without any running program to measure exact time.

4.1 Iris and wine classifications

An iris classification is a problem that MLP gets 4 input data (sepal length and width, petal length and width) and classifies three types of iris. So MLP consisted of 4 input nodes, 3 output nodes and 5-9 hidden nodes. We used 50 input patterns for each iris type and ended up with 150 input patterns.

Tab. I shows the iris results according to the number of hidden nodes. In Tab. I, average epoch and average time are the averaged results of 30 runs. This table tells us that our approach outperformed other methods in training time as well as in the number of failures. On the whole, the proposed method spent about 68% of the iterations of Hagan and about 49% of the iterations of Chen. Especially, our strategy got the minimum number of failures. Hagan's failure rate was 2 times higher than ours and Chen's was 2.6 times higher. The cause was that our method made the direction of weights consistent by the weight inner product. In addition, a diagonally dominant Jacobian matrix could enhance the possibility that the matrix could be positive definite.

In Fig. 2, (a) indicates the training epochs obtained by MLP with 5 hidden nodes for 30 runs. As the figure shows, our approach got better results than both Hagan's and Chen's methods. (b) illustrates the comparison of training errors on the 21$^{\text{st}}$ run among the results of (a). We can also find that our method saved training epochs best, and Hagan's method was next. As we pointed out before, Chen's method took more training time because it needed a lot of iterations to increase the damping parameter in the beginning of training, which we could see in (c). The figure (c) shows the changes of damping parameters about the result of (b). The y axis is a log scale here. Our approach had a few oscillations in the damping parameter. But Hagan's method was still oscillating until the end of training.

As the second classification, the wine problem classifies three types of wine after taking thirteen chemical attributes as inputs. Thus, we constructed MLP with 13 input nodes and 3 output nodes, and trained it by changing 4-8 hidden nodes. The total 178 input patterns were used, and other conditions for training were the same as those of the iris problem. Tab. II shows the results of wine classification. We expected this problem to be more complicated because the number of input nodes was large. But we knew that the wine problem was easier than the iris problem because the number of failures was low and the training time was shorter. We can also find that the training time of our method was shorter than those of others as well as the possibility of failures was lower.

Fig. 3 (a) indicates the results acquired by MLP with 7 hidden nodes, and both (b) and (c) are the results of the 23$^{\text{rd}}$ run. In (a) and (b), both Hagan's method and our method showed similar results. Our method, however, saved a few iterations when we referred to Tab. II. In (c), our damping parameter was changing like our expectation that it is to rise to behave like a gradient descent method in the beginning and lessen to do like the Gauss-Newton method in the end. On the other hand, Hagan's parameter was oscillating even in the end of training.

4.2 Handwritten digit recognition

To test a more complicated problem and get more general results, we applied our strategy to a handwritten digit recognition. We used 1000 input patterns, 100 patterns of each digit. Each digit was 12x12 pixels and each pixel had a hexadecimal value by gray level. MLP had 144 input nodes and 10 output nodes, and we trained it by changing 10-14 hidden nodes. These MLPs cost a lot of time because of huge weights. Table III shows the results of CEDAR data. We can see that our strategy obtained better results than other methods in training time and the number of failures. Our method needed about 37% of the iterations of Hagan and about 41% of the iterations of Chen. In addition, Hagan's failure was 4 times higher than ours and Chen's failure was 2.3 times higher. It is noteworthy that Chen's failure was lower than Hagan's in this intricate problem.

In Fig. 4, (a) is the results of MLP used 13 hidden nodes. (b) and (c) represent the evolution of the 28$^{\text{th}}$ run among 30 runs. Here, we can also find that our approach had few oscillations in the end of training so it could train CEDAR data faster than others. Using the weight inner product and a diagonally dominant matrix brought us these good results. Our strategy can lead to better results in

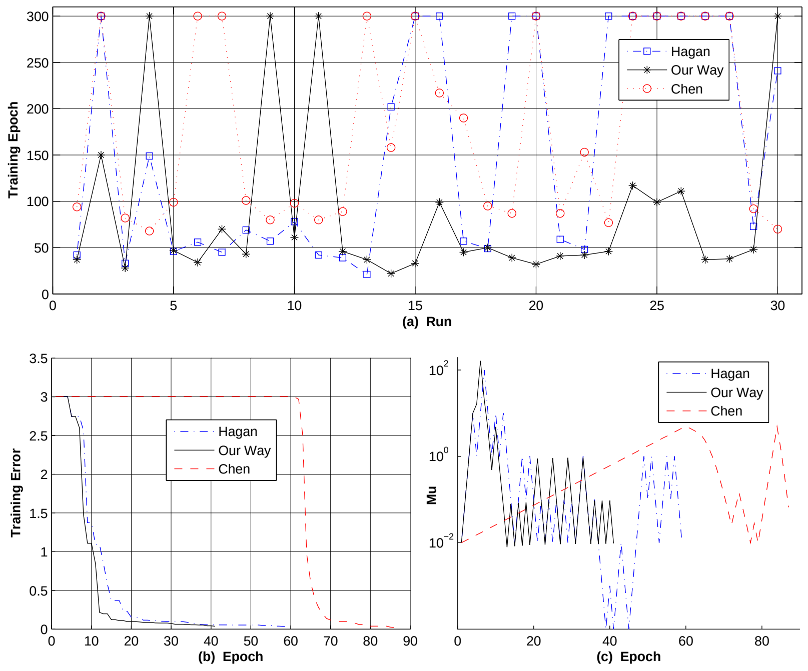

Fig. 2 *Comparison of results on iris data.*

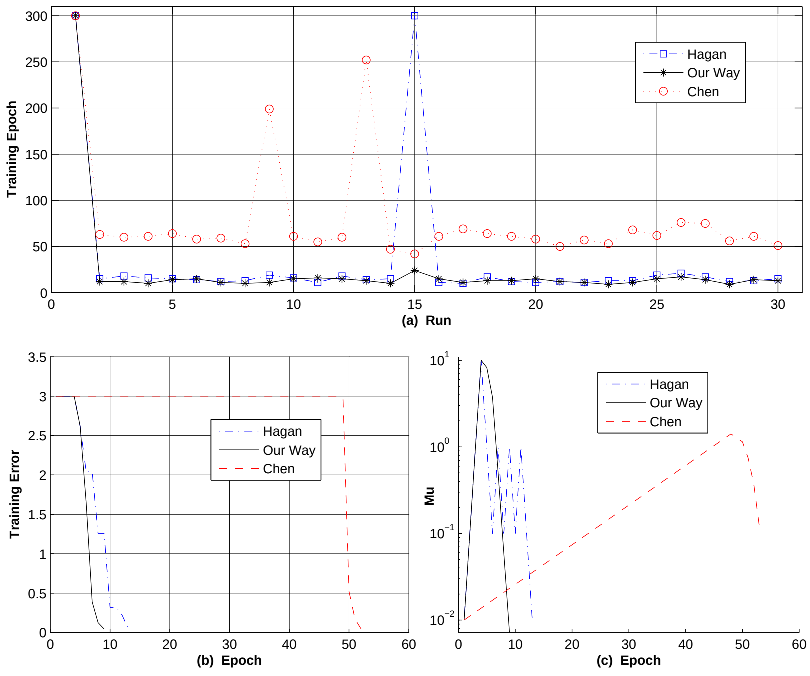

Fig. 3 *Comparison of results on wine data.*

Number of	Average epoch			Average time (second)			Number of failures		
hidden nodes	Hagan	Our Way	Chen	Hagan	Our Way	Chen	Hagan	Our Way	Chen
10	212.37	107.7	179.37	2317.02	1209.82	1952.68	20	9	14
11	151.2	52.27	158.37	2062.73	732.68	2152.55	13	3	11
12	120.27	46	136.87	1966.78	773.51	2235.38	10	2	8
13	102.5	34.23	74.77	1920.21	661.27	1393.57	8	1	0
14	117.8	20.77	86.53	2662.19	481	1946.95	10	0	2

Tab. III *CEDAR Results.*

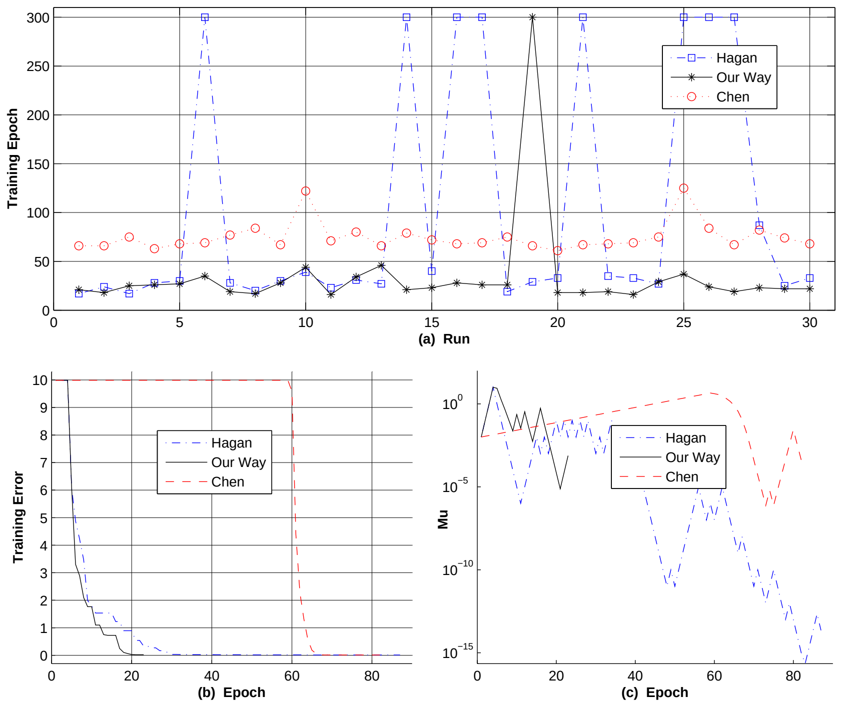

Fig. 4 *Comparison of results on CEDAR data.*

the problems with huge weight space like this complicated problem. Especially, the weight inner product plays an important role in searching for a target weight vector.

We got other noticeable simulations, even though we did not present them in this paper. We tested the same three problems by either the weight inner product (decreasing μ) or a diagonally dominant matrix (increasing μ). As a result, the results with decreasing μ by the weight inner product got better performance than those with only increasing μ. Therefore, we can recommend using only the weight inner product when you want to implement our approach simply. On the whole, the range of μ changed by our approach was smaller than that of Hagan's method and similar to that of Chen's method. It means that the proposed method makes μ

increase or decrease flexibly according to both training situation and the direction between a current weight vector and its target weight vector.

5. Conclusion

The LM algorithm is a combination of a gradient descent method and the Gauss-Newton method, and its damping parameter switches between two methods in this algorithm. Changing the damping parameter by a fixed decay rate makes training errors oscillate and takes much time to finish training work. Therefore, we proposed the weight inner product to decrease damping parameter and a diagonally dominant matrix to increase damping parameter. The proposed method strengthens a gradient descent method when a weight vector is far away from its target vector, and the Gauss-Newton method when it is close to its target vector. In our experiment, we have found that our approach converges faster than other methods and is less likely to fail in trainings. The improved damping strategy can save iterations and raise the rate of training success in problems with large weight space because it can keep weight directions toward its target weight direction.

Acknowledgements

This paper was supported by research funds of Chonbuk National University in 2010.

References

[1] Lippman R. P.: An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, **4**, 2, 1987, pp. 4–21.

[2] Rumelhart D. E., McClelland J. L.: Parallel Distributed Processing, MIT Press, Cambridge, MA, 1986, pp. 318–362.

[3] Vogl T. P., Mangis J. K., Zigler A. K., Zink W. T., Alkon D. L.: Accelerating the Convergence of the Back-Propagation method, Biological Cybernetics, 59, 1988, pp. 256–263.

[4] Xiao-Hu Yu, Guo-an Chen, Shi-Xin Cheng: Dynamic Learning Rate Optimization of the Backpropagation Algorithm, IEEE Trans. on Neural Networks, **6**, 3, 1995, pp. 669–677.

[5] Nied A., Seleme Jr. S. I., Parma G. G., Menezes B. R.: On-line neural training algorithm with sliding mode control and adaptive learning rate, Neurocomputing, 70, 2007, pp. 2687–2691.

[6] Magoulas G. D., Plagianakos V. P., Vrahatis M. N.: Globally Convergent Algorithms With Local Learning Rates, IEEE Trans. on Neural Networks, **13**, 3, 2002, pp. 774–779.

[7] Behera L., Kumar S., Patnaik A.: On Adaptive Learning Rate That Guarantees Convergence in Feedforward Networks, IEEE Trans. on Neural Networks, **17**, 5, 2006, pp. 1116–1125.

[8] Behera L., Kumar S., Patnaik A.: Corrections to On Adaptive Learning Rate That Guarantees Convergence in Feedforward Networks, IEEE Trans. on Neural Networks, **19**, 6, 2008, pp. 1141.

[9] Kordos M., Duch W.: Variable step search algorithm for feedforward networks, Neurocomputing, 71, 2008, pp. 2470–2480.

[10] Chen H. H., Manry M. T., Chandrasekaran H.: A Neural Network Training Algorithm Utilizing Multiple Sets of Linear Equations, Neurocomputing, **25**, 1-3, 1999, pp. 55–72.

[11] Ergezinger S., Thomsen E.: An accelerated learning algorithm for multilayer perceptrons optimization Layer by Layer, IEEE Trans. on Neural Networks, **6**, 1, 1995, pp. 31–42.

[12] Sang-Hoon Oh, Soo-Young Lee: A New Error Function at Hidden Layers for Fast Training of Multilayer Perceptrons, IEEE Trans. on Neural Networks, **10**, 4, 1999, pp. 960–964.

[13] Nocedal J., Wright S.: Numerical Optimization, Springer 2nd edition, 2006.

[14] Nocedal J.: Theory of algorithms for unconstrained optimization, Acta Numerica, 1, 1992, pp. 199–242.

[15] Charalambous C.: Conjugate gradient algorithm for efficient training of artificial neural networks, IEEE Proceedings, **139**, 3, 1992, pp. 301–310.

[16] Setiono R., Chi Kwong Hui L.: Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm, IEEE Trans. on Neural Networks, **6**, 1, 1995, pp. 273–277.

[17] Hagan M. T., Menhaj M.: Training feedforward networks with the Marquardt algorithm, IEEE Trans. on Neural Networks, **5**, 6, 1994, pp. 989–993.

[18] Costa M. A., Braga A. P., Menezes B. R.: Improving generation of MLPs with sliding mode control and the Levenberg-Marquardt algorithm, Neurocomputing, 70, 2007, pp. 1342–1347.

[19] Xu J., Ho D. W. C.: A new training and pruning algorithm based on node dependence and Jacobian rank deficiency, Neurocomputing, 70, 2006, pp. 544–558.

[20] Lera G., Pinzolas M.: Neighborhood Based Levenberg-Marquardt Algorithm for Neural Network Training, IEEE Trans. on Neural Networks, **13**, 5, 2002, pp. 1200–1203.

[21] Toledo A., Pinzolas M., Ibarrola J. J., Lera G.: Improvement of the Neighborhood Based Levenberg-Marquardt Algorithm by Local Adaptation of the Learning Coefficient, IEEE Trans. on Neural Networks, **16**, 4, 2005, pp. 988–992.

[22] Lampton M.: Damping-undamping strategies for the Levenberg-Marquardt nonlinear least-squares method, Computers in Physics, **11**, 1, 1977, pp. 110–115.

[23] Hagan M. T., Demuth H. B., Beale M.: Neural Network Design, PWS Publishing Company, 1995.

[24] Tai-cong Chen, Da-jian Han, Francis T. K Au, Tham L. G.: Acceleration of Levenberg-Marquardt Training of Neural Networks with Variable Decay Rate, International Joint Conference on Neural Networks Proceedings, Portland, Oregon, 3, July 2003, pp. 1873–1878.

[25] Amir Abolfazl Suratgar, Mohammad Bagher Tavakoli, Abbas Hoseinabadi: Modified Levenberg-Marquardt Method for Neural Networks Training, Proc. of World Academy of Science, Engineering and Technology, 6, June 2005, pp. 46–48.

[26] Nobuo Yamashita, Masao Fukushima: On the rate of Convergence of the Levenberg-Marquardt Method, Computing (Suppl. 15), 2001, pp. 237–249.

[27] Jin-yan Fan, Ya-xiang Yuan: On the Quadratic Convergence of the Levenberg-Marquardt Method without Nonsingularity Assumption, Computing, 74, 2005, pp. 23–39.

[28] Aha D., Asuncion A., Newman D.: UCI Machine Learning Repository, Availabe: http://archive.ics.uci.edu/ml/index.html

[29] Hull J. J.: A Database for Handwritten Text Recognition Research, IEEE Trans. on Pattern Analysis and Machine Intelligence, **16**, 5, 1994, pp. 550–554.