2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

An Efficient Conjugate Gradient Based Learning Algorithm For Multiple Optimal Learning Factors of Multilayer Perceptron Neural Network

Xun Cai
School of Computer Science and Technology
Shandong University
Jinan, P.R.China, 25010
caixunzh@sdu.edu.cn

Kanishka Tyagi
Department of Electrical Engineering,
The University of Texas at Arlington
Arlington, TX, USA, 76010
kanishka.tyagi@mavs.uta.edu

Michael T.Manry
Department of Electrical Engineering
The University of Texas at Arlington
Arlington, TX, USA, 76010
manry@uta.edu.

Zhi Chen
School of Computer Science and Technology
Shandong University
Jinan, Shandong, P.R.China, 25010
chenzhisdu@gmail.com

***Abstract*—In this paper, a second order learning algorithm based on *Conjugate Gradient* (CG) method for finding *Multiple Optimal Learning Factors* (MOLFs) of multilayer perceptron neural network is proposed in details. The experimental results on several benchmarks show that, compared with *One Optimal Learning Factor* algorithm with *Optimal Output Weights* (1OLF-OWO) and *Levenberg-Marquardt* learning algorithm (LM), our proposed CG based MOLF method with optimal output weights which is also called MOLFCG-OWO algorithm has not only significantly faster convergence rate than that of 1OLF and even super to that of LM learning algorithm for some datasets with much less computational time, but also more generalization capability than 1OLF-OWO. Thus, MOLFCG-OWO algorithm is suggested better choice for some practical applications.**

Keywords —Multilayer Perceptron Neural Network ; conjugate gradient method; multiple learning factors

I. INTRODUCTION

Multilayer Perceptron Neural Network (MLP NN) has several favorable properties of universal approximation and the ability to mimic Bayes discriminant [1] which make it a popular tool for many regression and classification applications including finance[2] and weather predication [3], automatic controller [4], image and documents retrieval [5,6] and clustering [7,8,9].

The fundamental theory of MLP NN is reviewed as following.

Given a dataset with N_v training patterns $\{(\boldsymbol{x}_p, \boldsymbol{t}_p)\}$, where p=1 to N_v, then the typical topology of a fully-connected feed-forward MLP consists of three layers including input layer, hidden layer and output layer , as shown in Fig.1.

In Fig.1, the numbers of units in input layer, hidden layer and output layer are denoted by N, N_h and M, respectively. The notation $\boldsymbol{x}_p$ =[x_p(1), x_p (2)…x_p (N+1)]^T denotes the input unit vector of pth pattern, where x_p (N+1) is an augmented input unit for convenience of handling the threshold of each hidden unit and is set to x_p (N+1)=1. The notation $\boldsymbol{t}_p$=[t_p(1), t_p (2)…t_p (M)]^T denotes the desired output vector of pth pattern corresponding to $\boldsymbol{x}_p$.

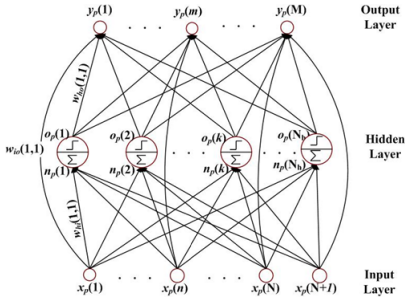


Fig. 1. The architecture of a fully connected feedforward MLP

For hidden layer, N_h denotes the number of hidden units and $\boldsymbol{W}_{ih}$={$w_{ih}(i,k)$} denotes all the hidden weights linking from input units to hidden units, where i is from 1 to N, k is from 1 to N_h.

For output layer, $\boldsymbol{W}_{ho}$={$w_{ho}(k,m)$} represents all the output weights linking from hidden units to output units with N_h by M dimension. For a full-connected feed-forward MLP, there is another kind of weight directly linking input units to output units which is called bypass weight $\boldsymbol{W}_{io}$={$w_{io}(i,m)$}, with N by M dimension.

For each hidden unit, each net function of hidden units is denoted by $n_p(k)$,which is given by

$$n_p(k) = \sum_{i=1}^{N+1} w_{ih}(i,k)\, x_p(i). \quad (1)$$

Correspondingly, each output of hidden units, $o_p(k)$, which is also called activation function is given by

$$o_p(k) = f\left(n_p(k)\right). \quad (2)$$

This work is sponsored by Innovation foundation of Shandong University (No. 2010TS001).

Normally, $f(\cdot)$ is taken as a sigmoid function which is shown by

$$f\left(n_p(k)\right) = \frac{1}{1+e^{-n_p(k)}}. \quad (3)$$

Based on the above notations, the actual output vector of MLP $\boldsymbol{y}_p$ is calculated by following expression

$$\boldsymbol{y}_p = (\boldsymbol{W}_{io})^{\mathrm{T}}\boldsymbol{x}_p + (\boldsymbol{W}_{ho})^{\mathrm{T}}\boldsymbol{o}_p. \quad (4)$$

If we choose batch mode training of MLP, then the typical error function E is the total mean squared error (MSE) of all output units and is given as follows

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}\left[t_p(m) - y_p(m)\right]^2. \quad (5)$$

For training hidden weights, the most basic algorithm is steepest descent back propagation [10]. In such case, the hidden weight matrix, $\boldsymbol{W}_{ih}$, is updated as

$$\boldsymbol{W}_{ih} \leftarrow \boldsymbol{W}_{ih} + z \cdot \boldsymbol{G}_{ih}, \quad (6)$$

where $\boldsymbol{G}_{ih}=\{g_{ih}(n,k)\}$ is an (N+1) by N_h negative gradient vector of error function with respect to each hidden unit , and is often calculated by the following equation

$$\boldsymbol{G}_{ih} = \frac{1}{N_v}\sum_{p=1}^{N_v}\boldsymbol{x}_p\left(\boldsymbol{\delta}_p\right)^{\mathrm{T}}, \quad (7)$$

where $\boldsymbol{\delta}_p = [\delta_p(1), \ldots, \delta_p(k) \ldots, \delta_p(N_h)]^{\mathrm{T}}$. $\delta_p(k)$ is the delta function corresponding to kth hidden unit and is calculated by using back propagation which is shown as follows

$$\delta_p(k) = f'\left(n_p(k)\right) \cdot \sum_{m=1}^{\mathrm{M}}\delta_o(m) \cdot w_{ho}(k,m), \quad (8)$$

where $\delta_o(m)$ denotes one of the element of the delta function of output units $\boldsymbol{\delta}_o$, and is normally given by

$$\boldsymbol{\delta}_o = 2(\boldsymbol{t}_p - \boldsymbol{y}_p). \quad (9)$$

If we combine the input vector and the output vector to form a new input vector $\boldsymbol{X}_p = [\boldsymbol{x}_p; \boldsymbol{o}_p]$ with (N+N_h+1) by 1 dimension, then the autocorrelation matrix of $\boldsymbol{X}_p$ which is denoted by $\boldsymbol{R}_o$, and the correlation matrix of $\boldsymbol{X}_p$ and $\boldsymbol{t}_p$, which is denoted by $\boldsymbol{C}_o$ can be given respectively by

$$\begin{cases} \boldsymbol{C}_o = \frac{1}{N_v}\sum_{p=1}^{N_v}\boldsymbol{y}_p\boldsymbol{X}_{\mathrm{p}}^{\mathrm{T}} \\ \boldsymbol{R}_o = \frac{1}{N_v}\sum_{p=1}^{N_v}\boldsymbol{X}_p\boldsymbol{X}_p^{\mathrm{T}} \end{cases}. \quad (10)$$

Hence, the output weights matrix, $\boldsymbol{W}_o$,can be calculated by the following linear equation

$$\boldsymbol{W}_o = \boldsymbol{R}_o\boldsymbol{C}_o^{\mathrm{T}}. \quad (11)$$

In this case, (11) can be solved by using orthogonal least square (OLS) [10,11].

A. The effects of learning factors

If steep descent algorithm is applied to minimize the error function in (6), the error function in (5) has to be zigzag to a minimum on the error surface, so steep descent algorithm tends to converge slowly.

In order to solve this cause, there are some heuristics that provide useful guidelines to accelerate the convergence of back-propagation learning, which includes: each adjustable weight should have its own individual learning factor and each learning factor should be updated after one iteration.

In light of above heuristics, the learning factor of each hidden weight should be given and updated individually after each iteration. Reference [12, 13] proposed a kind of multiple optimal learning factor (MOLF) algorithms which used Newton's method to evaluate the optimal learning factors. However, due to inversion of Hessian matrix, Newton's method has the computation complexity of O (n^2). Along with the total of number of multiple learning factors, N*N_h, grows, the computation of OLS will increase very fast which makes itself impractical for large scale network.

In order to solve this problem and to keep the optimal properties given by the second order optimization, CG is regarded as an intermediate between the method of steep descent and Newton's method which can accelerate the slow rate of convergence experienced with the method of steep descent, while avoiding the computational requirements associated with the evaluation, storage and inversion of the Hessian matrix in Newton's method [14,15,16]. Due to these advantages of CG, in this paper we propose to use conjugate gradient (CG) method instead of OLS to improve the training rate and convergence of MOLF.

The rest of paper is organized as follows: in the section II, we will introduce our CG-based MOLF algorithm in details. In section III , the procedure of the proposed MOLF-CG is given. In section IV, the experimental results for comparing the performance of our CG-based MOLF with a those of 1OLFBP-OWO and LM learning algorithms are given. Finally, the conclusions and some advance work are discussed in section V.

II. OUR WORK

The output weights of our proposed CG-based MOLF method are trained by *Output Weights Optimization* (OWO) method and the learning vector z_k for the kth hidden unit is being updated. Then, the predicted mth output is given by

$$y_p(m) = \sum_{i=1}^{N+1} w_{oi}(m,i)x_p(i) + \sum_{k=1}^{N_h} w_{oh}(m,k)\, f\left(\sum_{i=1}^{N+1}\left(w_{ih}(k,i) + z_k g(k,i)\right)x_p(i)\right). \quad (12)$$

For each iteration, all the hidden weights are updated respectively by

$$w_{ih}(k,i) \leftarrow w_{ih}(k,i) + z_k \cdot g(k,i). \quad (13)$$

If the negative derivative vector of error function to each learning factor z_k is denoted by $'(z_k)$, then the second derivative of the *j*th output error function with respect to *k*th learning factor , $E"(z_{m,k})$ can be rewritten as

$$E'(z_k) = \frac{-2}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}[t_p(m) - y_p(m)]w_{oh}(m,k)o_p(k)\Delta n_p(k). \quad (14)$$

$$E"(z_{m,k}) \approx \frac{-2}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}[w_{oh}(m,k)o'_p(k)\Delta n_p(k)]^2. \quad (15)$$

Also, if we use $\boldsymbol{G}_z$ to denote gradient vector of which each element is given by $E'(z_k)$ and use $\boldsymbol{H}_z$ to denote Hessian matrix of which each element is given by $E"(z_{m,k})$ then for he learning factor vector $\boldsymbol{z}$ of which each element is z_k , then by using Newton's method, $\boldsymbol{z}$ is calculated by the follow formula

$$\boldsymbol{z} = (\boldsymbol{H}_z)^{-1}\boldsymbol{G}_z. \quad (16)$$

Normally, the above linear equation is solved very efficiently by using Orthogonal Least Square (OLS) method as [13, 14] shown.

However, computation of Newton ' s method depends on the rank deficient of which of $\boldsymbol{H}_z$ ten suffers from the intrinsically ill-conditioned nature of neural network training problems. Moreover, the computation complexity of the (16) is O(n^3) which will be very expensive with the increasing size of network. To overcome some of these difficulties, it is widely known that the conjugate-gradient method is perhaps the only method that is applicable to large-scale problems, that is, problems with hundreds or thousands of adjustable parameters. In this paper, we derive an efficient conjugate gradient method for MOLF learning algorithm.

If the optimal learning factor vector $\boldsymbol{z}$ for *it*th iteration is denoted by $\boldsymbol{z}^{(it)}$, then its updating expression is

$$\boldsymbol{z}^{(it)} = \boldsymbol{z}^{(it-1)} + \Delta\boldsymbol{z}^{(it)}, \quad (17)$$

where *it* is the order of iterations which is from 0 to N_{it} and N_{it} is the total number of iterations. Based on the second order of Taylor series, the quadratic form of error function for $\boldsymbol{z}$ is rewritten as

$$E(\boldsymbol{z}^{(it)}) \approx E(\boldsymbol{z}^{(it-1)})^2 + (1/2)(\Delta\boldsymbol{z}^{(it)})^{\mathrm{T}}\boldsymbol{H}_z{}^{(it-1)}\Delta\boldsymbol{z}^{(it)} - (\boldsymbol{G}_z{}^{(it-1)})^{\mathrm{T}}(\Delta\boldsymbol{z}^{(it)}), \quad (18)$$

where

$$\boldsymbol{G}_z{}^{(it)} = E'(\boldsymbol{z}^{(it)}) \quad (19)$$

and

$$\boldsymbol{H}_z{}^{(it)} = E''(\boldsymbol{z}^{(it)}). \quad (20)$$

Thus, the gradient of such quadratic form is defined as

$$E'^{(\boldsymbol{z}^{(it)})} = (1/2)(\boldsymbol{H}_z{}^{(it-1)})^{\mathrm{T}}(\Delta\boldsymbol{z}^{(it)}) + (1/2)\boldsymbol{H}_z{}^{(it-1)}(\Delta\boldsymbol{z}^{(it)}) - (\boldsymbol{G}_z{}^{(it-1)})^{\mathrm{T}}. \quad (21)$$

Since $\boldsymbol{H}_z{}^{(it-1)}$ is symmetric, (21) can be simplified to

$$E'(\boldsymbol{z}^{(it)}) = (\boldsymbol{H}_z{}^{(it-1)})^{\mathrm{T}}(\Delta\boldsymbol{z}^{(it)}) - (\boldsymbol{G}_z{}^{(it-1)})^{\mathrm{T}}. \quad (22)$$

To minimize the quadratic function, the desired optimal value of $(\Delta\boldsymbol{z})^*$ will be

$$(\Delta\boldsymbol{z})^* = (\boldsymbol{H}_z{}^{(it-1)})^{-1}(\boldsymbol{G}_z{}^{(it-1)})^{\mathrm{T}} \quad (23)$$

and to obtain the optimal learning factor vector $(\Delta\boldsymbol{z})^*$, the new $\Delta\boldsymbol{z}$ should be chosen along a search direction $\boldsymbol{d}^{(it)}$, that is

$$\Delta\boldsymbol{z}^{(it)} = \Delta\boldsymbol{z}^{(it-1)} + \boldsymbol{\alpha}^{(it-1)}\boldsymbol{d}^{(it-1)}. \quad (24)$$

To find the optimal $\boldsymbol{\alpha}^{(it-1)}$, we set the value of $\partial\mathrm{E}/\partial\boldsymbol{\alpha}^{(it-1)}$ as follows

$$\partial\mathrm{E}/\partial\boldsymbol{\alpha}^{(it-1)} = 0. \quad (25)$$

Then, in light of chain rule, we can get

$$(\partial\mathrm{E}/\partial\Delta\mathbf{z}^{(it)})\,(\partial\Delta\mathbf{z}^{(it)}/\partial\boldsymbol{\alpha}^{(it-1)}) = 0. \quad (26)$$

and by (23), we can also get

$$\partial\Delta\boldsymbol{z}^{(it)}/\partial\boldsymbol{\alpha}^{(it-1)} = \boldsymbol{d}^{(it-1)}, \quad (27)$$

where $\Delta\boldsymbol{z}^{(it)}$ is expressed as

$$\boldsymbol{r}^{(it)} = -\boldsymbol{G}_z{}^{(it-1)} = -\left((\boldsymbol{H}_z{}^{(it-1)})(\Delta\boldsymbol{z}^{(it)}) - (\boldsymbol{G}_z{}^{(it-1)})^{\mathrm{T}}\right) \quad (28)$$

If we define $\boldsymbol{r}^{(it)}$equals to negative gradient of error function with respect to , we can get

$$(\boldsymbol{r}^{(it)})^{\mathrm{T}}\boldsymbol{d}^{(it-1)} = 0. \quad (29)$$

In (29), we can see that each successive gradient will be linear independent to the previous search direction during the process. In this way, if we define the differential vector $\boldsymbol{e}^{(it)}$ as the distance of current value of $\boldsymbol{z}$ from the solution, that is

$$\boldsymbol{e}^{(it)} = \Delta\boldsymbol{z}^{(it)} - \Delta\boldsymbol{z}^*, \quad (30)$$

then

$$\boldsymbol{e}^{(it)} = \boldsymbol{e}^{(it-1)} + \boldsymbol{\alpha}^{(it-1)}\boldsymbol{d}^{(it-1)}. \quad (31)$$

By (30), $\boldsymbol{r}^{(it)}$ in (28) can also be rewritten as follows

$$\boldsymbol{r}^{(it)} = -\boldsymbol{H}_z{}^{(it-1)}\left(\boldsymbol{e}^{(it)} + \Delta\boldsymbol{z}^*\right) + \boldsymbol{G}_z{}^{(it-1)} \quad (32)$$

$$= -\boldsymbol{H}_z{}^{(it-1)}\boldsymbol{e}^{(it)} + \left(\boldsymbol{G}_z{}^{(it-1)} - \boldsymbol{H}_z{}^{(it-1)}\Delta\boldsymbol{z}^*\right). \quad (33)$$

Since from (23), it is shown that

$$\boldsymbol{G}_z{}^{(it-1)} - \boldsymbol{H}_z{}^{(it-1)}\Delta\boldsymbol{z}^* = 0 \quad (34)$$

and

$$\boldsymbol{r}^{(it)} = -\boldsymbol{H}_z{}^{(it-1)}\boldsymbol{e}^{(it)} . \quad (35)$$

By (29), we can get

$$\left(\boldsymbol{H}_z{}^{(it-1)}\boldsymbol{e}^{(it)}\right)^{\mathrm{T}}\left(\boldsymbol{d}^{(it-1)}\right) = 0 . \quad (36)$$

From (36), it is known that $\boldsymbol{d}^{(it-1)}$ is $\boldsymbol{H}_z{}^{(it-1)}$-conjugate to $\boldsymbol{e}^{(it)}$, that is

$$\left(\boldsymbol{e}^{(it)}\right)^{\mathrm{T}}\left(\boldsymbol{H}_z{}^{(it-1)}\right)^{\mathrm{T}}\boldsymbol{d}^{(it-1)} = 0, \quad (37)$$

which means that if we can find a set of $\boldsymbol{H}_z{}^{(it)}$-conjugate direction vector , then we can minimize the error function to get the optimal value of $\boldsymbol{z}$.

However, to construct a set of $\boldsymbol{H}_z{}^{(it)}$-conjugate search direction $\boldsymbol{d}^{(it)}$, we need a set of linear independent vectors. From (29), we can see that if we choose the successive previous residual to be orthogonal to the previous residuals, which is described as

$$\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\left(\boldsymbol{r}^{(k)}\right) = 0, \text{ for all } k < it + 1, \quad (38)$$

then the successive residuals $\{\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)}, \cdots, \boldsymbol{r}^{(it)}\}$ can be used to construct a set of searching direction vector $\{\boldsymbol{d}^{(0)}, \boldsymbol{d}^{(1)}, \cdots, \boldsymbol{d}^{(it)}\}$, which the conjugated gradient method is named. In this way, the expression of $\boldsymbol{d}^{(it)}$ is given by

$$\boldsymbol{d}^{(it)} = \boldsymbol{r}^{(it)} + \sum_{k=0}^{it-1`} \beta^{(it,k)}\boldsymbol{d}^{(k)}, \text{for } it = 1 \ldots Nit - 1. \quad (39)$$

By (29) and (33), assuming $\boldsymbol{H}_z{}^{(it)} \approx \boldsymbol{H}_z{}^{(it-1)}$, we can get

$$\boldsymbol{r}^{(it)} \approx \boldsymbol{r}^{(it-1)} - \boldsymbol{\alpha}^{(it-1)}\boldsymbol{H}_z{}^{(it-1)}\boldsymbol{d}^{(it-1)}. \quad (40)$$

By taking inner product $\left(\boldsymbol{d}^{(it-1)}\right)^{\mathrm{T}}$ at both side of (40) and from (36), $\boldsymbol{\alpha}^{(it-1)}$ can be calculated as

$$\boldsymbol{\alpha}^{(it-1)} = \frac{\left(\boldsymbol{d}^{(it-1)}\right)^{T}\left(\boldsymbol{r}^{(it-1)}\right)}{\left(\boldsymbol{d}^{(it-1)}\right)^{T}\boldsymbol{H}_z{}^{(it-1)}\left(\boldsymbol{d}^{(it-1)}\right)}. \quad (41)$$

Also, by taking inner product of $\boldsymbol{r}^{(it)}$ at both side of (40), (41) can be rewritten as

$$\boldsymbol{\alpha}^{(it-1)} = \frac{-\left(\boldsymbol{r}^{(it)}\right)^{T}\left(\boldsymbol{r}^{(it)}\right)}{\left(\boldsymbol{r}^{(it)}\right)^{T}\boldsymbol{H}_z{}^{(it-1)}\left(\boldsymbol{d}^{(it-1)}\right)}. \quad (42)$$

Also, by taking inner product of $\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}$ and from (37), we can get

$$\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\boldsymbol{d}^{(it)} = \left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\boldsymbol{r}^{(it)}. \quad (43)$$

To get $\boldsymbol{d}^{(it)}$ in (43), we have to get value $\boldsymbol{\beta}^{(it,k)}$ in (39). If we multiply $\left(\boldsymbol{H}_z{}^{(it)}\boldsymbol{d}^{(it)}\right)^{\mathrm{T}}$ on the both sides of (35), then to satisfy the $\boldsymbol{H}_z{}^{(it)}$-conjugate condition, we can get

$$\boldsymbol{\beta}^{(it,it-1)} = -\frac{\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\boldsymbol{H}_z{}^{(it)}\boldsymbol{d}^{(it-1)}}{\left(\boldsymbol{d}^{(it-1)}\right)^{\mathrm{T}}\boldsymbol{H}_z{}^{(it)}\left(\boldsymbol{d}^{(it-1)}\right)}. \quad (44)$$

With the initial condition $\boldsymbol{d}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{G}_z{}^{(0)}$ and orthogonal condition in (29), it is known that

$$\boldsymbol{r}^{(it+1)}\left(\boldsymbol{r}^{(it)} + \sum_{k=0}^{it-1`} \boldsymbol{\beta}^{(it,k)}\boldsymbol{d}^{(k)}\right)^{\mathrm{T}} = 0 . \quad (45)$$

Then , by taking inner product of $\left(\boldsymbol{r}^{(it+1)}\right)^{\mathrm{T}}$ on the both sides of (39) ,we can get that $\boldsymbol{\beta}^{(it,k)} = 0$,for all $k < it$.

Hence, (39) can be simplified as

$$\boldsymbol{d}^{(it)} = \boldsymbol{r}^{(it)} + \boldsymbol{\beta}^{(it,it-1)}\boldsymbol{d}^{(it-1)}. \quad (46)$$

$\boldsymbol{\beta}^{(it,it-1)}$ can be rewritten based on (40) and (41) as follows

$$\boldsymbol{\beta}^{(it,it-1)} = \frac{1}{\boldsymbol{\alpha}^{(it-1)}}\frac{-\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\left(\boldsymbol{r}^{(it)}\right)}{\left(\boldsymbol{d}^{(it-1)}\right)^{\mathrm{T}}\boldsymbol{H}_z{}^{(it)}\left(\boldsymbol{d}^{(it-1)}\right)} \quad (47)$$

$$= \frac{-\left(\boldsymbol{r}^{(it)}\right)^{\mathrm{T}}\left(\boldsymbol{r}^{(it)}\right)}{\left(\boldsymbol{r}^{(it-1)}\right)^{T}\left(\boldsymbol{r}^{(it)-1}\right)}. \quad (48)$$

III. PROCEDURE

In this paper, the procedure for training MOLFCG-OWO is shown as follows:

Step 1	Initialize input vector to be zero means.
Step 2	By using normal distribution function and net control, initialize input weight matrix $\boldsymbol{W}_{ih}{}^{(0)}$.
Step2	Use OWO to get the initial optimal output weight matrix, $\boldsymbol{W}_o{}^{(0)}$ by (11).
Step 3	By (7) and (8), get the initial negative Jacobian matrix $\boldsymbol{G}_{ih}{}^{(0)}$.
Step 4	By (29) ~ (33) , obtain $\boldsymbol{G}_z{}^{(0)}$and $\boldsymbol{H}_z{}^{(0)}$.
Step 5	Set $\boldsymbol{d}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{G}_z{}^{(0)}$.
Step 6	Start *it*th iteration, for $0 \le it < \boldsymbol{N}_{it}$.
Step 7	Get $\boldsymbol{\alpha}^{(it)}$, by (41).
Step 8	Update $\boldsymbol{z}^{(it+1)}$ by (24).

Step 9	Get $\boldsymbol{H}_z^{(it)}$ by (20) and update $\boldsymbol{r}^{(it+1)}$ by (40).
Step 10	Update $\boldsymbol{\beta}^{(it,k)}$ by (48).
Step 11	Update $\boldsymbol{d}^{(it-1)}$ by (46).
Step 12	Get $\boldsymbol{G}_{ih}{}^{(it+1)}$ by (7) and update $\boldsymbol{W}_{ih}{}^{(it+1)}$ by (6).
Step 13	Get optimal $\boldsymbol{W}_o{}^{(it+1)}$ by OWO as (11) shows.
Step 14	Combine new $\boldsymbol{W}_{ih}{}^{(it+1)}$ and $\boldsymbol{W}_o{}^{(it+1)}$ to get new actual output vector $\boldsymbol{y}^{(it+1)}$ by (4) and the new training error E, by (5).
Step 15	Update $\boldsymbol{r}^{(it-1)}$ by (40), and return to step (6).

IV. EXPERIMENTAL RESULTS

In order to compare the performance of our proposed MOLFCG-OWO with those of 1OLFBP-OWO[11], and traditional Levenberg-Marquardt (LM) [17] , we implemented many experiments on several real-life datasets and bench marks. All these simulations ran on Intel Core2 Duo P8700. For the convenience of comparison, the number of training iterations for each simulation was fixed to 100, that is, N_{it}=100 and all the synaptic weights were initialized by the same way to guarantee that all the training errors start at the same point.

A. Description of Datasets

In this paper, we take the four datasets as examples, which are called "Twod", "Single2", "Mattrn" and "Concrete", respectively. All these datasets used for simulation are publicly available and the input units of all these datasets have been normalized to be zero-mean and unit variance.

The first three are taken from some real-life application [18] and the last one is a benchmark from UCI Machine Learning Repository [19, 20]. Among these four datasets, "Twod" is used for inversion the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, "Mattrn" is for inversion of random two-by-two matrices and "Single2" is for inversion of surface permittivity of the normalized surface , rms roughness and the surface correlation length found in back scattering models from randomly rough dielectric surfaces. The benchmark of "Concrete" dataset is used for approximation of the nonlinear function of age and ingredients of concrete compressive strength.

The numbers of patterns, input units, output units and hidden units of the four datasets which are denoted as N_v, N, M, and N_h, as well as the co-linearity of each dataset for investigating the effects of co-linearity for training are listed respectively in TABLE 1.

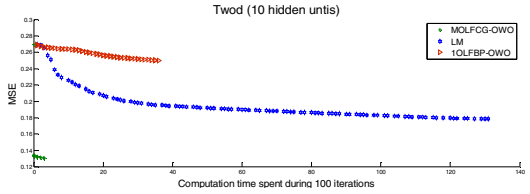
B. Training error and computational time

Under the above training conditions, each training MSE and computational time spent per iteration for MOLF, 1OLFBP-OWO and LM is plotted, respectively, as Fig.2 shows.

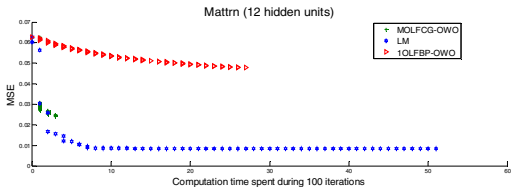
TABLE I. THE STRUCTURES OF NEURATL NETWORKS FOR FOUR DATASETS

Datasets	N_v	N	M	N_h	*Co linearity*
Twod	1768	8	7	10	high
Single2	10000	16	3	8	high
Mattrn	2000	4	4	12	high
Concrete	1030	8	1	15	low

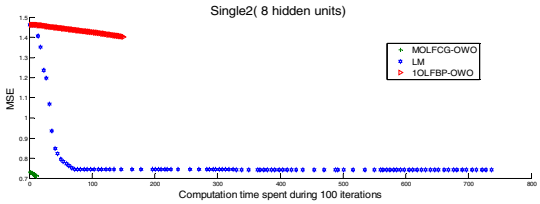
Fig.2(a)~(d) shows the plots of average mean square errors and computational time spent of 1OLFBP-OWO, LM and MOLFCG-OWO algorithms during each iteration. It obviously shows that MOLFCG-OWO has much faster converging rates than that of 1OLFBP-OWO , even rivals LM for "twod" and "single2" datasets as (a) and (c) shown. Moreover, the total computational training time of MOLF is much less than those of 1OLFBP-OWO and LM.It can also easily be seen that for high co-linearty datasets, such as "Twod", "Mattrn" and "Single2", MOLFCG-OWO has the best training convergence as Fig 2.(a)~(c) shows.



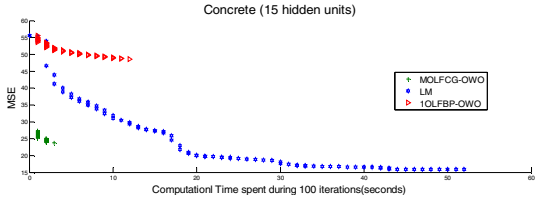
(a) "Twod"



(b)"Mattrn"



(c)"Single2"



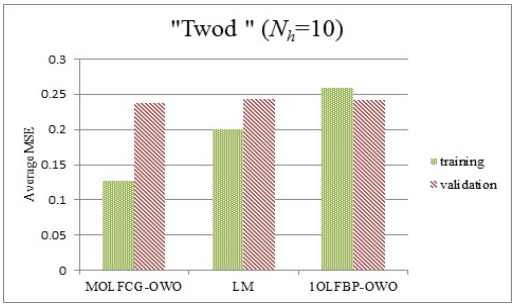
(d)"Concrete"

Fig. 2. The training error and computational time spent on the four datasets

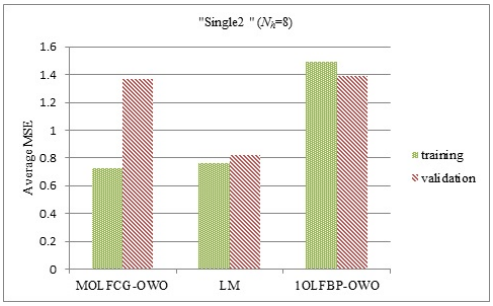
C. K-fold cross-validation

In order to learn the generalization capability of our proposed MOLFCG-OWO algorithm, we implement 10-fold cross-validation method on those four datasets receptively, and both the training errors and validation errors of them are compared in Fig. 3(a)~(d). From Fig.

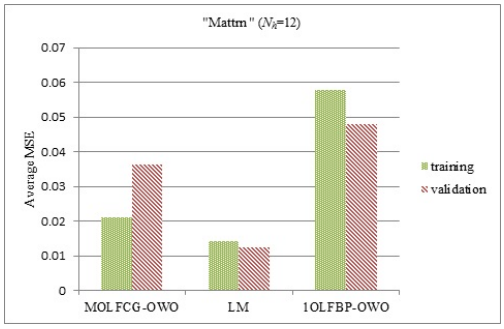
3, we can find that both the training errors and validation errors of our proposed algorithm are much better than 1OLF-OWO algorithm, but not always better than LM, only the errors for "twod" is lower and others are higher. The reason is that LM is a method which can adapt itself to be prone to the first order learning method or to the second order learning method based on the training errors by tuning the parameter λ, which makes it more possible to get more generalization capability by getting out of local optimal points of learning progress for some applications.



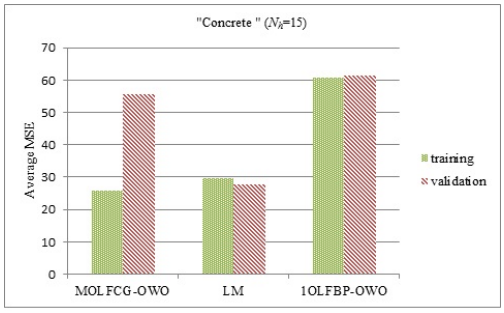
(a) "Twod"



(b) "Single2"



(c) "Mattrn"



(d) "Concrete"

Fig. 3. The 10-fold cross- validation errors of the four datasets

V. CONCLUSION

In this paper, we investigate an efficient conjugate gradient method for optimizing multiple learning factors of MLP which is called MOLFCG-OWO algorithm. The experimental results on some real-life datasets and benchmarks show that the proposed algorithm not only greatly improves the convergence of 1OLFBP-OWO, especially for high co-linearity datasets, but also even rivals LM with much less computational time. Hence, MOLFCG-OWO is obviously good choice for practical applications of MLP training algorithms.

Since LM algorithm still is a good choice for better generalization capability, in order to low its high computational time, our future work will focus on updating Newton's method into CG method to find all the optimal weights.

REFERENCES

[1] Dennis W. Ruck et al., "The mulitlayer perceptron as an approximation to a Bayes optimal discriminant function," IEEE Transactions on Neural Networks, vol. 1, No. 4, pp.256-298, 1990.

[2] Meesomsarn, K., Chaisricharoen, R., Chipipop, B. and Yooyativong, T. "Forecasting the effect of stock repurchase via an artificial neural network," ICROS-SICE International Joint Conference 2009, pp.2573 - 2578, Aug 2009.

[3] Ramón Velo, Paz López, Francisco Maseda. "Wind speed estimation using multilayer perceptron, " Energy Conversion and Management vol. 81, pp.1-9, 2014.

[4] Martin T. Hagan, Howard B. Demuth and Orlando De Jesús, "An introduction to the use of neural networks in control systems," International Journal Of Robust And Nonlinear Control. vol. 12, n.11,pp:959-985, 2002.

[5] Sangjae Leea, Joon Yeon Choehb, "Predicting the helpfulness of online reviews using multilayer perceptron neural networks," Expert Systems with Applications , vol.41, pp.3041–3046, 2014.

[6] Jyothi B.V., Eswaran K., "Comparative study of neural networks for image retrieval," International Conference on Intelligent Systems, Modelling and Simulation, pp. 199-203, Jan . 2010.

[7] K.-L, Du. "Clustering: A neural network approach," Neural Networks, vol. 23, n. 1, pp. 89-107, 2011.

[8] Melacci, S., Maggini M.and Sarti L., "Semi-supervised clustering using similarity neural networks," International Joint Conference on Neural Networks, (Atlanta, GA, June 14-19, 2009), pp.2065 – 2072, June 2009.

[9] Rumelhart D.E., Hinton G.E.and Williams R.J., "Learning internal representations by error propagation," Parallel distributed processing: explorations in the microstructure of cognition, Mit Press Computational Models Of Cognition And Perception Series, Cambridge, MA, USA, vol. 1: foundations.1, pp. 318 – 362, 1986.

[10] F. J. Maldonado, M. T. Manry and Tae-Hoon Kim,. "Finding optimal neural network basis function subsets using the Schmidt procedure" In Proceedings of the International Joint Conference on Neural Networks,(Portland, Oregon,2003),1, pp.444 – 449, July , 2003.

[11] H-H Chen, M.T. Manry, and H. Chandrasekaran, "A neural network training algorithm utilizing multiple sets of linear equations", Neurocomputing, vol. 25, pp. 1-3, 55-72, 1999.

[12] Sanjeev S. Malalur and Michael T. Manry, "Multiple optimal learning factors for feed-forward networks", The SPIE Defense, Security and Sensing (DSS) Conference, Orlando, FL, April 2010.

[13] Praveen Jesudhas, Michael T Manry, Sanjeev Malalur,, "Analysis and improvement of multiple optimal learning factors for feedforward networks", International Joint Conference on Neural Networks,(San, Joes, CA.) , pp. 2593 – 2600, July 31-Aug. 5, 2011.

[14] R.Fletch. “Practical methods of optimization”. A Wiley-Interscience Pulication, 2000.

[15] Simon Haykin, “Neural networks: a comprehensive foundation”, Pretice-Hall, Inc.2008.

[16] Jonathan Richard Shewchuk, “An introduction to the conjugate gradient method without the agonizing pain”, Technical report. School of Computer Science, Carnegie Mellon University, August 1994.

[17] Fun., M,H., and Hagan., M,T., “ Levenberg-Marquardt training for modular networks,” Proc. Of IEEE International Conference on Neural Networks’96, Washington DC,pp.468-473, 1996.

[18] http://www-ee.uta.edu/eewb/ip/training_data_files.htm.

[19] http://archive.ics.uci.edu/ml/.

[20] Cheng Yeh., “ Modeling of strength of high performance concrete using artificial neural networks,” Cement and Concrete Research,vol. 28,n.12, pp.1797-1808,1998.