ANALYSIS AND IMPROVEMENT OF MULTIPLE OPTIMAL LEARNING FACTORS FOR FEED-FORWARD NETWORKS

by

PRAVEEN JESUDHAS

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2010

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Manry for supervising my thesis work. Throughout my research, he has spent a lot of lab hours in patiently explaining even the most basic of questions. I also would like to thank him for providing me with good knowledge and a clear perspective of neural networks through his courses. I strongly believe he is the best supervising professor one could hope for.

I would also like to thank my thesis committee members Dr. Wei-Jen Lee and Dr. Saibun Tjuatja for reviewing my work.

I express my utmost gratitude to my parents, Mr. Peter and Mrs. Prabha and my sister, Priyanka for all their love and support. I dedicate this thesis to them.

November 18, 2010

ABSTRACT

ANALYSIS AND IMPROVEMENT OF MULTIPLE OPTIMAL LEARNING FACTORS FOR FEED FORWARD NETWORKS

Praveen Jesudhas, M.S

The University of Texas at Arlington, 2010

Supervising Professor: Michael T. Manry

The effects of transforming the net function vector in the Multilayer Perceptron (MLP) are analyzed. The use of optimal diagonal transformation matrices on the net function vector is proved to be equivalent to training the MLP using multiple optimal learning factors (MOLF). A method for linearly compressing large ill-conditioned MOLF Hessian matrices into smaller well-conditioned ones is developed. This compression approach is shown to be equivalent to using several hidden units per learning factor. The technique is extended to large networks. In simulations, the proposed algorithm performs almost as well as the Levenberg Marquardt (LM) algorithm with the computational complexity of a first order training algorithm.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Neural Networks and research

Neural networks have found wide acceptance in a variety of applications ranging from parameter estimation [1][2], document analysis and recognition [3], finance, manufacturing [4] and datamining [5]. One of the oldest and most useful types of neural network is the multilayer perceptron (MLP)[37]. The basis functions of the MLP are dynamically tuned through training. The MLP's approximation capabilities [6,7] make it an effective tool for classification and approximation. The relatively short evaluation time make it a better choice than many other classification and approximation methods such as support vector machines (SVM) [35]. The universal approximation [9] property of the MLP along with its ability to mimic Bayes discriminant[10], optimal L2 norm estimates[8] and maximum a-posteriori(MAP)[11] estimates give it a strong theoretical foundation.

Though neural networks have a variety of applications and solid theoretical validity, they also have limitations due to a lack of scalable and effective training algorithms .The error functions used with neural net training are not quadratic. Hence many iterations are required for training [24]. MLP training algorithms can get trapped in local minima [34]. The presence of dependent inputs [33] can adversely affect neural net training. In certain applications online training of neural network is employed [30, 31]. The convergence proofs for online training algorithms [32] are not as strong as those for batch training algorithms where a fixed set of training data is used. In this thesis we exclusively investigate batch training algorithms.

Batch training algorithms for the MLP can be briefly classified as first and second order algorithms. First order algorithms like back-propagation (BP) [12], conjugate gradient (CG) [13]

and output weight optimization-backpropagation (OWO-BP) require fewer multiplies and data passes and hence take less time per iteration. However, they are sensitive to input means and gains [14]. Second order methods on the other hand, are not quite as sensitive to input means and gains, but they do have their own inherent limitations. Second order algorithms related to Newton's method often have non-positive definite or singular Hessian matrices [15][16] which result in unstable training. Hence the Levenberg-Marquardt (LM) algorithm [17][18] is used normally. The LM algorithm is very computationally intensive due to the large size of the Hessian matrix so its use in training large networks is often limited.

When an optimal learning factor (OLF) calculated using the Gauss-Newton approximation [19] is used with OWO-BP, the resulting training algorithm is observed to work better than BP. The process of using a separate OLF for each of the hidden unit net function and combining them with OWO-BP, denoted as MOLF-BP [20] has produced better results than OWO-BP. However, the MOLF-BP approach still has certain limitations. The MOLF Hessian can be ill-conditioned. The size of the MOLF Hessian matrix $\mathbf{H}_{molf}$ can also become prohibitively large for larger networks, resulting in scalability problems.

This thesis gives a strong theoretical foundation to the basic MOLF algorithm by relating to optimally transforming the net function of MLP. The scalability issues of the basic MOLF algorithm for large networks are dealt with by collapsing the MOLF Hessian matrix to reduce the number of optimal learning factors so that the Hessian size is reduced. This process will also decrease the chances of the Hessian matrix to be singular.

1.2 Thesis organization

This thesis is organized as follows. Chapter 2 provides a review of the existing batch training algorithms. A review of the basic MOLF training algorithm is provided in chapter 3 along with an analysis of the problems present in it. The effects of optimally transforming the net

function vector of an MLP are analyzed for the MOLF algorithm in chapter 4. Chapter 5 includes a detailed description of the collapsing procedure for the MOLF Hessian matrix and its use in rectifying the problems of the basic MOLF algorithm. Experimental results comparing the improved MOLF algorithm with LM and other algorithms are made in Chapter 6.

CHAPTER 2

REVIEW OF BATCH TRAINING ALGORITHMS

2.1 MLP Notation

In the fully connected MLP of Figure 1, input weights w(k,n) connect the n^{th} input to the k^{th} hidden unit. Output weights $w_{oh}(m,k)$ connect the k^{th} hidden unit's activation $o_p(k)$ to the m^{th} output $y_p(m)$, which has a linear activation. The bypass weight $w_{oi}(m,n)$ connects the n^{th} input to the m^{th} output. The training data, described by the set $(\mathbf{x}_p,\mathbf{t}_p)$ consists of N-dimensional input vectors $\mathbf{x}_p$ and M-dimensional desired output vectors, $\mathbf{t}_p$. The pattern number p varies from 1 to N_v where N_v denotes the number of training vectors present in the data set.

In order to handle thresholds in the hidden and output layers, the input vectors are augmented by an extra element $x_p(N+1)$ where, $x_p(N+1) = 1$, so $\mathbf{x}_p = [x_p(1), x_p(2),\ldots., x_p(N+1)]^T$. Let N_h denote the number of hidden units. The dimensions of the weight matrices **W**, $\mathbf{W_{oh}}$ and $\mathbf{W_{oi}}$ are respectively N_h by (N+1), M by N_h and M by (N+1) .The vector of hidden layer net functions, $\mathbf{n}_p$ and the actual output of the network, $\mathbf{y}_p$ can be written as

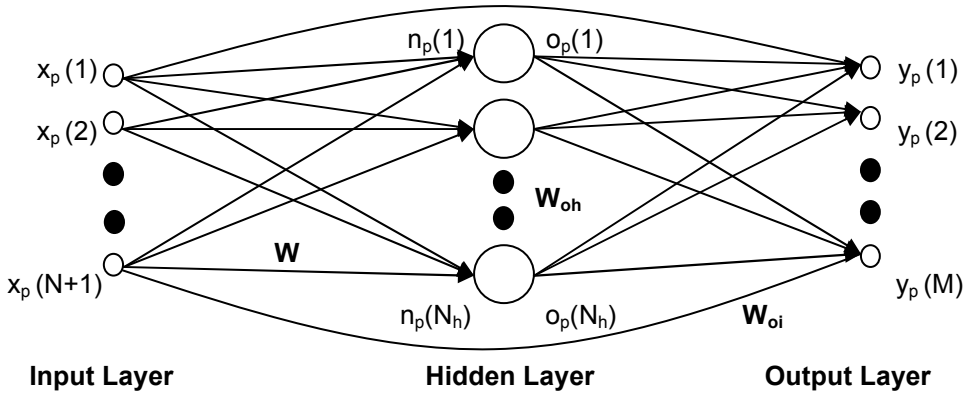

Figure 2.1 A fully connected Multilayer Perceptron

$$\mathbf{n_p} = \mathbf{W} \cdot \mathbf{x_p} \tag{1}$$

$$\mathbf{y_p} = \mathbf{W_{oi}} \cdot \mathbf{x_p} + \mathbf{W_{oh}} \cdot \mathbf{o_p}$$

where the k^{th} element of the hidden unit activation vector $\mathbf{o_p}$ is calculated as $o_p(k) = f(n_p(k))$ and f(.) denotes the hidden layer activation function. Training an MLP typically involves minimizing the mean squared error between the desired and the actual network outputs, defined as

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [\, t_p(i) - y_p(i) \,]^2 = \frac{1}{N_v} \sum_{p=1}^{N_v} E_p, \tag{2}$$

$$E_p = \sum_{i=1}^{M} [\, t_p(i) - y_p(i) \,]^2$$

where E_p is the cumulative squared error for pattern p.

2.2 Discussion of Back-Propagation

BP training [12, 22, 23, 24] is a gradient based learning algorithm which involves changing system parameters so that the output error is reduced. For a MLP, the expression for actual output found from input is provided in (1). The weights of the MLP are changed by BP so that the error E of equation (2), which computes the sum of the squared errors between the actual and desired outputs, decreases. The weights are updated using gradients of the error with respect to the corresponding weights.

The required error gradients are calculated using the chain rule utilizing delta functions, which are negative gradients of E_p with respect to net functions. For the p^{th} pattern, output and hidden layer delta functions [24] are respectively found as,

$$\delta_{po}(i) = 2 \left(t_p(i) - y_p(i) \right) \tag{3}$$

$$\delta_p(k) = f'\left(n_p(k) \right) \sum_{i=1}^{M} \delta_{po}(i) w_{oh}(i,k)$$

Now, the negative gradient of E with respect to w(k,n) is,

$$g(k,n) = \frac{-\partial E}{\partial w(k,n)} = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p(k) x_p(n) \tag{4}$$

The matrix of negative partial derivatives can therefore be written as,

$$\mathbf{G} = \frac{1}{N_v} \sum_{p=1}^{N_v} \boldsymbol{\delta}_p \cdot \mathbf{x}_p^T \tag{5}$$

where $\boldsymbol{\delta}_p = [\delta_p(1), \delta_p(2) \ldots, \delta_p(N_h)]^T$ [12]. If steepest descent is used to modify the hidden weights, $\mathbf{W}$ is updated in a given iteration as,

$$\mathbf{W} \leftarrow \mathbf{W} + z \cdot \mathbf{G} \tag{6}$$

$$\Delta \mathbf{W} = z \cdot \mathbf{G}$$

where z is the learning factor.

2.3 Discussion of Output-Weight-Optimization-Back Propagation

Here, we describe OWO-BP, a first order algorithm which trains two subsets of network weights [25] separately. In a given iteration of OWO-BP, we alternately (i) find the output weight matrices, $\mathbf{W_{oh}}$ and $\mathbf{W_{oi}}$ connected to the network outputs and (ii) separately train the input weights, $\mathbf{W}$ using BP.

Output weight optimization (OWO) is a technique finding weights connected to the actual outputs of the network. Since the outputs have linear activations, finding the weights connected to the outputs is equivalent to solving a system of linear equations. The expression for the actual outputs given in (1) can be re-written as

$$\mathbf{y_p} = \mathbf{W_o} \cdot \bar{\mathbf{x}}_\mathbf{p}$$

where $\bar{\mathbf{x}}_\mathbf{p} = [\mathbf{x_p}, \mathbf{o_p}]^T$ is the augmented input vector of size N_u where N_u equals $N + N_h + 1$, $\mathbf{W}_o$ is formed as $[\mathbf{W}_{oi} : \mathbf{W}_{oh}]$ and has a size of M by N_u. The output weights can be solved for by setting $\partial E / \partial \mathbf{W}_o = 0$ which leads to a set of linear equations given by,

$$\mathbf{C_a} = \mathbf{R_a} \cdot \mathbf{W_o^T} \tag{7}$$

where,

$$\mathbf{C_a} = \frac{1}{N_v} \sum_{p=1}^{N_v} \bar{\mathbf{x}}_\mathbf{p} \cdot \mathbf{t_p^T}$$

$$\mathbf{R_a} = \frac{1}{N_v} \sum_{p=1}^{N_v} \bar{\mathbf{x}}_\mathbf{p} \cdot \bar{\mathbf{x}}_\mathbf{p}^\mathbf{T}$$

Equation (7) is most easily solved using orthogonal least squares (OLS) [21].

In the second half of an OWO-BP iteration, the input weight matrix **W** is updated as in (6), where **G** is the generic representation of a direction matrix that contains information about the direction of learning and z, the learning factor that contains information about the step length to be taken in the direction **G**. The learning factor z could be specified heuristically or found optimally. The optimal learning factor (OLF)[19] used in this thesis is derived as,

$$z = \frac{-\partial E / \partial z}{\partial^2 E / \partial z^2} \tag{8}$$

The weight update $z \cdot \mathbf{G}$ in equation (6) can also be denoted as,

$$z \cdot \mathbf{G} = \Delta \mathbf{W} \tag{9}$$

For backpropagation, the direction matrix is nothing but the N_h by (N+1) negative input weight Jacobian matrix **G** computed as in equation (5).

A brief description of OWO-BP is given below. For every training iteration,

i. Find the negative Jacobian matrix **G** described in equation (5)

ii. Find the OLF and update the input weights, **W**, using equation (6)

iii. Solve the system of linear equations in (7) using OLS and update the output weights, $\mathbf{W_o}$.

This method is attractive for several reasons. First, the training is faster, since BP is not being used to find $\mathbf{W_o}$. Second, it helps avoid some local minima. Third, the method exhibits improved training performance compared to using only BP to update all the weights in the network.

2.4 Second Order Methods

Second order training methods for the MLP, related to Newton's method [26, 27, 28, 29], involve quadratic modeling of the error function. The subsequent update of the weights is derived by minimizing this model error function.

Let $\mathbf{w_f}$ be the vector containing all elements of the known weight matrices $\mathbf{W_{oi}}$, $\mathbf{W_{oh}}$ and $\mathbf{W}$. Let $\mathbf{e} = \mathbf{w_f}' - \mathbf{w_f}$ be the unknown weight change vector, where $\mathbf{w_f}'$ is the desired or improved version of $\mathbf{w_f}$ that is to be found. $\mathbf{H}$ and $\mathbf{g}$ respectively denote the Hessian matrix and negative gradient vector of the error E with respect to weight vector $\mathbf{w_f}$. For the Hessian matrix $\mathbf{H}$, the m^{th} row, n^{th} column element is given by,

$$h(m,n) = \frac{\partial^2 E}{\partial w_f(m) \partial w_f(n)} \tag{10}$$

Similarly the m^{th} element of the negative gradient vector $\mathbf{g}$ is,

$$g(m) = \frac{-\partial E}{\partial w_f(m)} \tag{11}$$

The multivariate Taylor's theorem says that E', which is E as a function of $\mathbf{e} = \mathbf{w_f}' - \mathbf{w_f}$, can be approximated as,

$$E' \approx E - \mathbf{e}^T\mathbf{g} + \tfrac{1}{2}\, \mathbf{e}^T \cdot \mathbf{H} \cdot \mathbf{e} \tag{12}$$

Setting $\partial E'/\partial \mathbf{e} = \mathbf{0}$, we get

$$\mathbf{H} \cdot \mathbf{e} = \mathbf{g} \tag{13}$$

so $\mathbf{e} = \mathbf{H}^{-1} \cdot \mathbf{g}$. The weight vector is then updated as,

$$\mathbf{w_f}' = \mathbf{w_f} + \mathbf{e} \tag{14}$$

The vector **e** can be found using OLS or other methods. However, for the MLP, the Hessian matrix **H** can be singular or ill-conditioned, so Newton's algorithm is not normally used for neural network training. A popular method which effectively rectifies this limitation is the Levenberg-Marquardt (LM) algorithm. LM [17,18] effectively interpolates between the Gauss-Newton form of Newton's algorithm and steepest descent according to the nature of the Newton Hessian matrix **H**. In LM the weights are updated as,

$$\mathbf{w_f} = \mathbf{w_f} + \mathbf{H_{LM}}^{-1}\mathbf{g} \tag{15}$$

$$\mathbf{H_{LM}}\,\mathbf{e} = \mathbf{g}$$

$$\mathbf{H_{LM}} = [\mathbf{H} + \lambda \cdot \mathrm{diag}(\mathbf{H})]$$

where diag(**H**) is **H** with off-diagonal elements set to 0 and $\mathbf{H_{LM}}$ is the Hessian matrix of the LM algorithm. In (15) if E increases, λ is increased. The weight vector is then updated as in (15), making the algorithm closer to steepest descent. Otherwise, λ is decreased. The LM algorithm gives good results, but unfortunately doesn't scale well due to large size of its Hessian matrix $\mathbf{H_{LM}}$.

2.5 Optimal Learning Factor

Calculation of the optimal learning factor (OLF) [19] z used with BP or OWO-BP is a second order algorithm. It also forms the basis for the MOLF algorithm [20].

The choice of learning factor z in the input-hidden weight update equation (6) has a direct effect on the convergence rate of OWO-BP. Using Taylor's series for the error E in (2), a non-heuristic optimal learning factor (OLF) for OWO-BP can be derived[19] as in (8), where the numerator and denominator derivatives are evaluated at z=0. The expression for the second derivative of the error with respect to the OLF is found using (2) as,

$$\frac{\partial^2 E}{\partial z^2} = \sum_{k=1}^{N_h}\sum_{j=1}^{N_h}\sum_{n=1}^{N+1} g(k,n) \sum_{i=1}^{N+1} \frac{\partial^2 E}{\partial w(k,n)\partial w(j,i)} g(j,i) = \sum_{k=1}^{N_h}\sum_{j=1}^{N_h} \mathbf{g}_k^T \mathbf{H}_R^{kj} \mathbf{g}_j \qquad (16)$$

where column vector $\mathbf{g}_k$ contains elements g(k,n) of **G**, for all values of n. $\mathbf{H}_R$ is the input weight Hessian with N_{iw} rows and columns, where $N_{iw} = (N + 1)\cdot N_h$ is the number of input weights. Clearly, $\mathbf{H}_R$ is reduced in size compared to **H**, the Hessian for all weights in the entire network. $\mathbf{H}_R^{k,j}$ contains elements of $\mathbf{H}_R$ for all input weights connected to the jth and kth hidden units and has size (N+1) by (N+1). When Gauss-Newton [15] updates are used, elements of $\mathbf{H}_R$ are computed as

$$\frac{\partial^2 E}{\partial w(j,i)\partial w(k,n)} = \frac{2}{N_v} u(j,k) \sum_{p=1}^{N_v} x_p(i) x_p(n) o'_p(j) o'_p(k) \qquad (17)$$

$$u(j,k) = \sum_{m=1}^{M} w_{oh}(m,j) w_{oh}(m,k)$$

where $o'_p(k)$ indicates the first partial derivative of $o_p(k)$ with respect to its net function. Because (17) represents the Gauss-Newton approximation to the Hessian, it is positive semi-definite. Equation (16) shows that

(i) The OLF can be obtained from elements of the Hessian $\mathbf{H}_R$

(ii) $\mathbf{H}_R$ contains useful information even when it is singular

(iii) A smaller non-singular Hessian $\partial^2 E/\partial z^2$ can be constructed using $\mathbf{H}_R$.

Therefore, it may be profitable to construct Hessian matrices of intermediate size between 1 by 1 and N_w by N_w for use in Newton's algorithm [20].

CHAPTER 3

THE MOLF ALGORITHM

3.1 Review of basic MOLF algorithm

In this section, the basic MLP-MOLF algorithm [20] is reviewed, where a vector **z** of optimal learning factors is computed. The MLP notation provided in Chapter 2 is followed in the equations.

A MLP is assumed to be trained using OWO-BP. Let **z** be the vector containing the optimal learning factors. Let z_k represent the OLF used to update each of the hidden unit input weights, w(k,n).The error function to be minimized is given by (2). The predicted output $y_p(m)$ is given by,

$$y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k)f\left(\sum_{i=1}^{N+1}(w(k,i)+z_k g(k,i)x_p(i)\right) \qquad (18)$$

where, g(k,i) is again an element of the negative Jacobian matrix **G** and f() denotes the hidden layer activation function. The negative first partial of E with respect to an OLF z_j is,

$$g_{molf}(j) = \frac{-\partial E}{\partial z_j} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}\left[\bar{t}_p(m) - \sum_{k=1}^{N_h} w_{oh}(m,k)o_p(z_k)\right] w_{oh}(m,j)o'_p(j)\Delta n_p(j) \qquad (19)$$

where,

$$\bar{t}_p(m) = t_p(m) - \sum_{n=1}^{N+1} w_{oi}(m,n)x_p(n)$$

$$\Delta n_p(j) = \sum_{n=1}^{N+1} x_p(n)g(j,n)$$

$$o_p(z_k) = f\left(\sum_{n=1}^{N+1}(w(k,n)+ z_k\, g(k,n))x_p(n)\right)$$

Using Gauss-Newton updates, the second partial derivative elements of the Hessian $\mathbf{H}_{molf}$ are derived as,

$$h_{molf}(k,j) \approx \frac{\partial^2 E}{\partial z_k \partial z_j} = \frac{2}{N_v} \sum_{m=1}^{M} w_{oh}(m,k) w_{oh}(m,j) \sum_{p=1}^{N_v} o'_p(k) o'_p(j) \Delta n_p(k) \Delta n_p(j) \tag{20}$$

$$= \sum_{i=1}^{N+1} \sum_{n=1}^{N+1} \left[\frac{2}{N_v} u(k,j) \sum_{p=1}^{N_v} x_p(i) x_p(n) o'_p(k) o'_p(j) \right] g(k,i) \cdot g(j,n)$$

where u(k,j) is as defined in equation (17).

The Gauss-Newton update guarantees that $\mathbf{H_{molf}}$ is non-negative definite [15]. Given the negative gradient vector, $\mathbf{g_{molf}} = [\ -\partial E/\partial z_1\ ,\ -\partial E/\partial z_2\ \ldots,\ -\partial E/\partial z_{N_h}\ ]^T$ and the Hessian $\mathbf{H}_{molf}$, the error E is minimized with respect to the vector **z** using Newton's method. For every iteration in the training algorithm, the steps are as follows:

i. Calculate the negative input weight Jacobian **G** using BP.

ii. Calculate **z** using Newton's method and update the input weights as

$$w(k,n) \leftarrow w(k,n) + z_k \cdot g(k,n) \tag{21}$$

iii. Solve linear equations for all output weights.

Thus the MOLF procedure has been successfully applied to OWO-BP, and the resulting algorithm is denoted as MOLF-BP. Similarly the MOLF procedure can also be used to improve other training algorithms.

3.2 The MOLF Hessian

In this sub section the MOLF Hessian is analyzed and compared to the Hessian matrix used in Newton's algorithm [20]. The effects of the dependent input and hidden units on the MOLF Hessian during training are also explored in this section [20].

The vector **z** can be computed by solving,

$$\mathbf{H_{molf}} \cdot \mathbf{z} = \mathbf{g_{molf}} \tag{22}$$

The MOLF Hessian $\mathbf{H_{molf}}$ of (20) can be expressed in terms of the Newton Hessian as,

$$h_{molf}(k,j) = \frac{\partial^2 E}{\partial z_k \partial z_j} = \sum_{i=1}^{N+1}\sum_{n=1}^{N+1}\left[\frac{\partial^2 E}{\partial w(k,i)\partial w(j,n)}\right] g(k,i)\cdot g(j,n) \tag{23}$$

The elements within the square brackets of equation (23) represent the Hessian $\mathbf{H_R}$ of the error E with respect to the input weights of the MLP. The size of $\mathbf{H_R}$, N_{iw} by N_{iw} with $N_{iw} = (N + 1)\cdot N_h$,is reduced compared to the size of **H** which is N_w by N_w, where $N_w = (M(N_h+N+1) + N_h(N+1))$. It can also be observed from (23) that the N_h by N_h MOLF Hessian $\mathbf{H_{molf}}$ leads to better scalability as compared to Newton's algorithms that use Hessians $\mathbf{H_R}$ or **H**.

3.3 Effects of linearly dependent signals

An analysis of the effect of linearly dependent input and hidden layers on the Hessian $\mathbf{H_{molf}}$ is presented here [20]. A linearly dependent input can be modeled as

$$x_p(N+2) = \sum_{n=1}^{N+1} b(n)\, x_p(n) \tag{24}$$

During OWO, the weights from the dependent input, feeding the outputs will be set to zero and the output weight adaptation will not be affected. During the input weight adaptation, the expression for gradient given by (19) can be re-written as,

$$g_{molf}(j) \equiv \frac{-\partial E}{\partial z_j} = \frac{2}{N_v}\sum_{p=1}^{N}\sum_{m=1}^{M}\left(t'_p(m) - \sum_{k=1}^{N_h} w_{oh}(m,k)\, o_p(z_k)\right)\cdot w_{oh}(m,j)\, o'_p(j)$$

$$\left[\Delta n_p(j) + g(k, N+2)\sum_{n=1}^{N+1} b(n) x_p(n)\right] \tag{25}$$

and the expression for an element of the Hessian in (20) can be re-written as,

$$\frac{\partial^2 E}{\partial z_k \partial z_j} = \frac{2}{N_v}\sum_{m=1}^{M} w_{oh}(m,k)\, w_{oh}(m,j) \sum_{p=1}^{N} o'_p(k) o'_j(j) \left[\Delta n_p(k)\Delta n_p(j) + \Delta n_p(k) g(j,N+2) \sum_{n=1}^{N+1} b(n) x_p(n) + \right.$$

$$\left. \Delta n_p(j) g(k,N+2) \sum_{i=1}^{N+1} b(i) x_p(i) + g(j,N+2) g(k,N+2) \sum_{n=1}^{N+1} \sum_{i=1}^{N+1} b(i) x_p(i) b(n) x_p(n) \right] \quad (26)$$

Let $\bar{\mathbf{H}}_{\mathbf{molf}}$ be the Hessian, when the extra dependent input $x_p(N+2)$ is included. Then,

$$\bar{h}_{molf}(k,j) = h_{molf}(k,j) + \frac{2}{N_v} u(k,j) \sum_{p=1}^{N_v} x_p(N+2) o'_p(k) o'_p(j) \left[g(k,N+2) \sum_{n=1}^{N+1} x_p(n) g(j,n) \right.$$

$$\left. g(j,N+2) \sum_{i=1}^{N+1} x_p(i) g(k,i) + x_p(N+2) g(k,N+2) g(j,N+2) \right] \quad (27)$$

On comparing equations (19) with (25) and (20) with (26) and (27), some additional terms are observed within the square brackets in the expressions for gradient and Hessian in the presence of linearly dependent input. Clearly, these extra terms will cause the training using MOLF to be different for the case of linearly dependent inputs. This leads to lemma 1.

Lemma 1: Linearly dependent inputs, when added to the network, do not force $\bar{\mathbf{H}}_{\mathbf{molf}}$ to be singular [20]. As seen in (26) and (27), each element $\bar{h}_{molf}(m,j)$ simply gains some first and second degree terms in the variables b(n).

Assume that some hidden unit activations are linearly dependent upon others, as

$$o_p(N_h+1) = \sum_{k=1}^{N_h} c(k) o_p(k) \quad (28)$$

Further assume that OLS is used to solve for output weights. The weights in the network are updated on each training iteration and it is quite possible that this could cause some hidden units to be linearly dependent upon each other. The dependence could manifest in hidden units being identical or a linear combination of other hidden unit outputs.

Consider one of the hidden units to be dependent. The autocorrelation matrix $\mathbf{R}_a$ in (7), will be singular and since we are using OLS to solve for output weights, all the weights connecting the dependent hidden unit to the outputs will be forced to zero. This will ensure that the dependent hidden unit does not contribute to the output [20]. To see if this has any impact on learning in MOLF we can look at the expression for the gradient and Hessian, given by (19) and (20) respectively. Both equations have a sum of product terms containing output weights. Since OWO sets the output weights for the dependent hidden unit to be zero, this will also set the corresponding gradient and Hessian elements to be zero. In general, any dependence in the hidden layer will cause the corresponding learning factor to be zero and will not affect the performance of MOLF. The above discussion on dependent hidden units leads to lemma 2.

Lemma 2: When OLS is used to solve (22), each hidden unit, dependent upon those already orthonormalized, results in zero-valued weights changes for that hidden unit [20].

3.4 Problems in MOLF

Lemmas 1 and 2 in the previous section show clearly that MOLF applied to OWO-BP still works reasonably even in the presence of dependent input and hidden units. The computational complexity is also quite small compared to that of Newton's algorithm due to the much smaller size of MOLF Hessian $\mathbf{H}_{\mathbf{molf}}$.

The MOLF Hessian, though small compared to $\mathbf{H}_{\mathbf{R}}$ and $\mathbf{H}$, can still be large for more complex networks having a lot of hidden units. Hence for large networks the MOLF algorithm may become unsuitable for training. According to lemma 2 for each dependent hidden unit, the corresponding row and column of $\mathbf{H}_{molf}$ is zero-valued. These zero valued elements do not contribute to effective training of the network, but they still influence the computational requirements of the MOLF algorithm.

In this thesis the MOLF algorithm is explored and analyzed in a more detailed manner using equivalent networks. Changes are also suggested to the MOLF algorithm for large networks and also when there are dependent units in the hidden layer.

CHAPTER 4

DETAILED ANALYSIS OF THE MOLF ALGORITHM

In this chapter a detailed analysis is made on the structure of the MLP using equivalent networks. The relationship between MOLF applied to OWO-BP and linear transformations is investigated of the net function vector.

4.1 Discussion of equivalent networks

Let MLP 1, have a net function vector $\mathbf{n_p}$ and output vector $\mathbf{y_p}$ after backpropagation training as discussed previously in section 2.2. In a second network called MLP 2 trained similarly, the net function vector and the output vector are respectively denoted by $\mathbf{n'_p}$ and $\mathbf{y'_p}$. The bypass and output weight matrix along with the hidden unit activation functions are considered to be equal for both the MLP 1 and MLP 2. Based on this information the output vectors for MLP 1 and MLP 2 are respectively,

$$y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)f\left(n_p(k)\right) \tag{29}$$

$$y'_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)f\left(n'_p(k)\right) \tag{30}$$

In order to make MLP 2 strongly equivalent to MLP 1 their respective output vectors $\mathbf{y_p}$ and $\mathbf{y'_p}$ have to be equal. Based on equations (29) and (30) this can be achieved only when their respective net function vectors are equal [37].

MLP 2 can be made strongly equivalent to MLP 1 by linearly transforming its net function vector $\mathbf{n'_p}$ before passing it as an argument to the activation function f(). This process is equivalent to directly using the net function $\mathbf{n_p}$ in MLP 2, denoted as,

$$\mathbf{n_p} = \mathbf{C} \cdot \mathbf{n'_p} \tag{31}$$

The linear transformation matrix **C** used to transform $\mathbf{n'_p}$ in equation (31) is of size N_h by N_h. This process of transforming the net function vector of MLP 2 is represented in Figure 4.1.

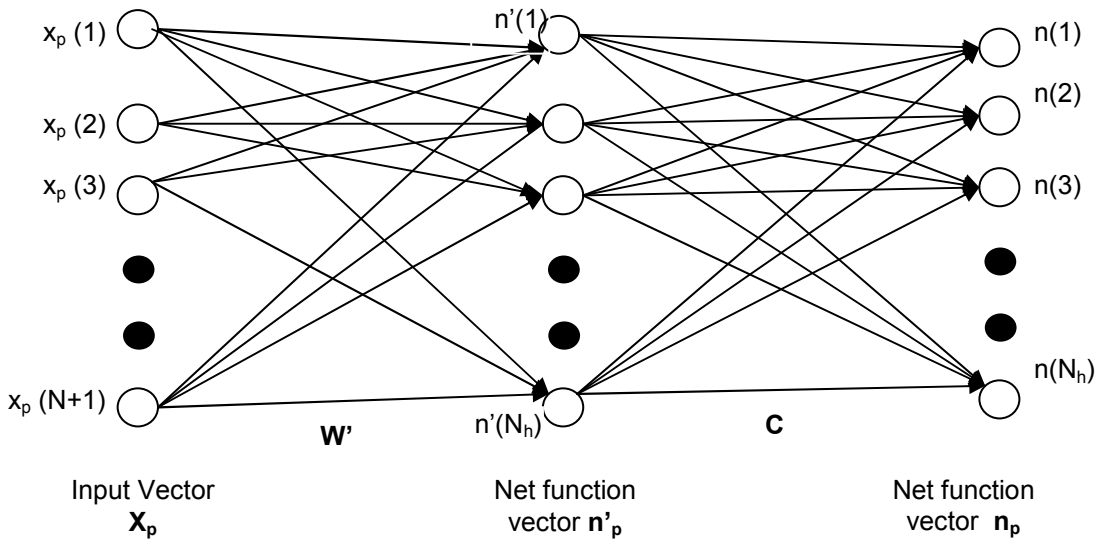

Figure 4.1 Graphical representation of net function vectors

On applying the linear transformation to the net function vector $\mathbf{n'_p}$ in equation (30), MLP 2 is made strongly equivalent to MLP1. We then get,

$$y'_p(i) = y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k) f\left(\sum_{m=1}^{N_h} c(m,k)\cdot n'_p(k)\right) \qquad (32)$$

After making MLP 2 strongly equivalent to MLP 1, the net function vector $\mathbf{n_p}$ can be related to its input weights **W** and the net function vector $\mathbf{n'_p}$ as,

$$n_p(k) = \sum_{n=1}^{N+1} w(k,m)x_p(m) = \sum_{m=1}^{N_h} c(k,m)\, n'_p(m) \qquad (33)$$

similarly the MLP 2 net function vector $\mathbf{n'_p}$ could be related to its weights **W'** as,

$$n''_p(k) = \sum_{n=1}^{N+1} w'(k,m)\, x_p(m) \qquad (34)$$

On substituting (34) into (33) we get,

$$n_p(k)= \sum_{n=1}^{N+1} w(k,m)\, x_p(m)= \sum_{n=1}^{N+1}\sum_{m=1}^{N_h} c(k,m)\, w'(m,n) x_p(n) \tag{35}$$

It is observed above that the net function of MLP 1 is found by linear transformation of the input weight matrix **W'** of MLP 2 given by,

$$\mathbf{W = C \cdot W'} \tag{36}$$

If the elements of the **C** matrix are found through an optimality criterion, then optimal input weights **W'** can be computed with the input weight matrix **W** of MLP 1 by inverting the **C** matrix in (36).

4.2 Optimal transformation of net function

The discussion of equivalent networks in section 4.1 suggests that, an optimal set of net functions can be obtained by linear transformation of the input weights. In this section, the input weight update equations are derived for OWO-BP training, using the linear transformation matrix **C** from MLP 2.

MLP 1 and MLP 2 are strongly equivalent as before. Then output of the two networks after transformation is given as in equation (32). The elements of the Jacobian matrix **G'** of MLP 2 found from equation (2) and (32) is defined as,

$$g'(u,v) = \frac{-\partial E}{\partial w'(u,v)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M} [\, t_p(i) - y_p(i)] \cdot \frac{\partial y_p(i)}{\partial w'(u,v)}, \tag{37}$$

$$\frac{\partial y_p(i)}{\partial w'(u,v)} = \sum_{k=1}^{N_h} w_{oh}(i,k)\, o'_p(k)\, c(k,u)\, x_p(v)$$

Rearranging terms in (37) results in,

$$g'(u,v) = \sum_{k=1}^{N_h} c(k,u) \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] w_{oh}(i,k) o'_p(k) x_p(v) \qquad (38)$$

$$= \sum_{k=1}^{N_h} c(k,u) \frac{-\partial E}{\partial w(k,v)}$$

which is abbreviated as

$$\mathbf{G'} = \mathbf{C^T G} \qquad (39)$$

The input weight update equation for MLP 2 based on its Jacobian vector **G'** is given by,

$$\mathbf{W'} = \mathbf{W'} + z \bullet \mathbf{G'}$$

On pre-multiplying this equation by **C** and using equations (36) and (39) we obtain

$$\mathbf{W} = \mathbf{W} + z \bullet \mathbf{G''} \qquad (40)$$

where,

$$\mathbf{G''} = \mathbf{R} \bullet \mathbf{G} \qquad (41)$$

$$\mathbf{R} = \mathbf{C}\,\mathbf{C^T}$$

Equation (40) is nothing but the weight update equation of MLP 1, which would result in the network being strongly equivalent to MLP 2. Thus using equation (40), the input weights of MLP 1 can be updated so that its net function is optimal as is MLP 2.

Lemma 3: For a given **R** matrix as defined in equation (41), there are an uncountably infinite number of **C** matrices.

The transformed Jacobian **G''** of MLP 1 is given in terms of the original Jacobian **G** as,

$$g''(u,v) = \sum_{k=1}^{N_h} r(u,k)\, g(k,v) \qquad (42)$$

$$g(k,v) = \frac{-\partial E}{\partial w(k,v)} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] w_{oh}(i,k) o'_p(k) x_p(v)$$

Equations (41) and (42) suggest that MLP 1 could be trained with optimal net functions using only the knowledge of the linear transformation matrix **R**.

4.3 Multiple Optimal Learning Factors for Input Weights

Equations (41) and (42) give a method for optimally transforming the net functions of an MLP by using the transformation matrix **R**. In this sub section, the MOLF approach is derived from equations (41) and (42). Let the **R** matrix in equation (41) be diagonal. Under this case equation (40) becomes,

$$w(k,n) = w(k,n) + z \bullet r(k) g(k,n) \tag{43}$$

On comparing equations (21) and (43), the expression for the different optimal learning factors z_k could be given as,

$$z_k = z \bullet r(k) \tag{44}$$

Equation (44) suggests that using the MOLF for training a MLP is equivalent to optimally transforming the net function using a diagonal transformation matrix.

It is proved that using MOLF algorithm is equivalent to optimally transforming the net function of the MLP with a diagonal transformation matrix.

CHAPTER 5

EFFECTS OF COLLAPSING THE MOLF HESSIAN

This chapter deals with the computational analysis of the existing MOLF algorithm followed by suggesting a method to collapse the MOLF Hessian matrix to create lesser number of optimal learning factors.

5.1 Computational Analysis of the MOLF algorithm

In the existing MOLF algorithm a significant amount of the computational burden is attributed to inverting the Hessian matrix to find the optimal learning factors, usually through the OLS procedure. The size of the Hessian Matrix which is directly based on the number of hidden units in the network, influences the computational load of the MOLF algorithm.

The gradient and Hessian equations respectively of the MOLF algorithm is provided in equation (19) and (20).The computation of the multiple optimal learning factor vector **z** requires that equation (22) be solved. The number of multiplies required for computing z through OLS is,

$$M_{ols\text{-}molf} = (N_h+1)[N_h(N_h+2)] \tag{45}$$

For networks having large values of N_h, this can be time consuming since solving (22) has to be done for every iteration of the training algorithm.

If some hidden units have linearly dependent outputs, the MOLF Hessian matrix $\mathbf{H}_{molf}$ can be ill conditioned as described by lemmas 1 and 2. This can affect MLP training.

Thus based on the above discussions, it can be inferred that having optimal learning factors equal to the number of hidden units result in higher computational load and also could lead to an ill-conditioned MOLF Hessian matrix.

5.2 Training with several hidden units per OLF

In this section, we modify the MOLF approach so that each OLF is assigned to one or more hidden units. The Hessian for this new approach is compared to $\mathbf{H_{molf}}$.

Let N_{OLF} be the number of optimal learning factors used for training in the proposed variable optimal learning factors (VOLF) method. N_{OLF} is selected such that it is less than or equal to N_h and also such that it divides N_h, with no remainder. Each optimal learning factor z_v applies to N_h/N_{OLF} hidden units. The MLP output based on these conditions is given by,

$$y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k)f\Big(\sum_{n=1}^{N+1}(w(k,n)+z_v g(k,n)x_p(n)\Big) \qquad (46)$$

$$v = \text{ceil}\,(\,k \bullet (N_{OLF}/N_h)\,)$$

where the function ceil() rounds the argument up to the next integer.

The negative gradient of the error in equation (2) with respect to each of the optimal learning factors z_v, denoted as $\mathbf{g_{volf}}$ is computed based on equation (46) as,

$$g_{molf1}(j) = \frac{-\partial E}{\partial z_j} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}\left[\bar{t}_p(m) - \sum_{k=1}^{N_h} w_{oh}(m,k)o_p(z_v,k)\right]\cdot \qquad (47)$$

$$\sum_{c=1}^{N_h/N_{OLF}}\left[w_{oh}(m,k(j,c))o'_p(k(j,c))\Delta n_p(k(j,c))\right]$$

where,

$$k(j,c) = c + (j-1)\cdot\frac{N_h}{N_{OLF}},$$

$$o_p(z_v,k) = f\Big(\sum_{n=1}^{N+1}(w(k,n)+ z_v\, g(k,n))x_p(n)\Big)$$

The Hessian matrix $\mathbf{H_{volf}}$ is derived from (2), (46) and (47) as,

$$h_{volf}(i,j)= \frac{\partial^2 E}{\partial z_i \partial z_j} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} \sum_{d=1}^{N_h/N_{OLF}} \left[w_{oh}(m,k(i,d)) o'_p(k(i,d)) \Delta n_p(k(i,d)) \right] \cdot$$

$$\sum_{c=1}^{N_h/N_{OLF}} \left[w_{oh}(m,k(j,c)) o'_p(k(j,c)) \Delta n_p(k(j,c)) \right] \quad (48)$$

The vector **z** of variable optimal learning factors is found from the Jacobian vector and Hessian matrix from the relation,

$$\mathbf{H_{volf}} \bullet \mathbf{z} = \mathbf{g_{volf}} \quad (49)$$

where $\mathbf{H_{volf}}$ is N_{OLF} by N_{OLF} and $\mathbf{g_{volf}}$ is of N_{OLF} by 1. Thus by varying the number of optimal learning factors required for training, the vector **z** is found using the method of section 3.1.

The computation required for solving equation (49) can be adjusted by choosing the number of optimal learning factors. When N_{OLF} equals one, the current procedure is similar to the OLF algorithm described in section 2.5. It requires less computation but the algorithm is also not very effective. When N_{OLF} equals N_h, then the algorithm reduces to the MOLF algorithm described in section 3.1. Thus by varying the number of optimal learning factors between one and N_h, the algorithm interpolates between the MOLF and OLF cases.

5.3 Collapsing the MOLF Hessian

Occasionally the MOLF approach can fail because of distortion in $\mathbf{H_{molf}}$ due to linearly dependent inputs or because it is ill-conditioned because of linearly dependent hidden units. In these cases we don't have to redo the current training iteration. Instead we can collapse $\mathbf{H_{molf}}$ and $\mathbf{g_{molf}}$ down to a smaller size and use the approach of the previous subsection. The elements of the original MOLF Hessian matrix $\mathbf{H_{molf}}$ for a MLP with N_h hidden units are denoted as,

$$\mathbf{H_{molf}} = \begin{pmatrix} h_{molf}(1,1) & h_{molf}(1,2) & h_{molf}(1,3) & \cdots & \cdots & h_{molf}(1,N_h) \\ h_{molf}(2,1) & h_{molf}(2,2) & h_{molf}(2,3) & \cdots & \cdots & \vdots \\ h_{molf}(3,1) & h_{molf}(3,2) & h_{molf}(3,3) & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ h_{molf}(N_h,1) & \cdots & \cdots & \cdots & \cdots & h_{molf}(N_h,N_h) \end{pmatrix} \tag{50}$$

On collapsing the Hessian matrix $\mathbf{H_{molf}}$ to a matrix $\mathbf{H_{molf1}}$ of size N_{OLF} by N_{OLF} we get,

$$\mathbf{H_{molf1}} = \begin{pmatrix} h_{molf1}(1,1) & h_{molf1}(1,2) & h_{molf1}(1,3) & \cdots & \cdots & h_{molf1}(1,N_{OLF}) \\ h_{molf1}(2,1) & h_{molf1}(2,2) & h_{molf1}(2,3) & \cdots & \cdots & \vdots \\ h_{molf1}(3,1) & h_{molf1}(3,2) & h_{molf1}(3,3) & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ h_{molf1}(N_{OLF},1) & \cdots & \cdots & \cdots & \cdots & h_{molf1}(N_{OLF},N_{OLF}) \end{pmatrix} \tag{51}$$

The relationship between the elements of $\mathbf{H_{molf1}}$ and $\mathbf{H_{molf}}$ is described by,

$$h_{molf1}(m,n) = \sum_{c=1}^{N_h/N_{OLF}} \sum_{d=1}^{N_h/N_{OLF}} h_{molf}(k(m,c),k(n,d)) \tag{52}$$

Equations (20 and (48) show that collapsing the MOLF Hessian matrix to size N_{OLF} by N_{OLF} results in a VOLF Hessian matrix of that size. Thus the $\mathbf{H_{molf1}}$ Hessian matrix of equation (51) is the equivalent to the VOLF Hessian matrix $\mathbf{H_{volf}}$ of equation (48).

On collapsing the Jacobian vector $\mathbf{g_{molf}}$ to a vector $\mathbf{g_{molf1}}$ having N_{OLF} elements we get,

$$\mathbf{g_{molf1}} = \left[g_{molf1}(1), g_{molf1}(2), g_{molf1}(3) \ldots\ldots\ldots\ldots g_{molf1}(N_{OLF})\right]^T \tag{53}$$

The relationship between $\mathbf{g_{molf1}}$ and $\mathbf{g_{molf}}$ is described by,

$$g_{molf1}(m) = \sum_{c=1}^{N_h/N_{OLF}} g_{molf}(k(m,c)) \tag{54}$$

Equations (19) and (47) show that collapsing the MOLF gradient vector to size N_{OLF} by 1 results in the VOLF gradient vector.

Therefore collapsing the MOLF Hessian matrix $\mathbf{H_{molf}}$ and gradient vector $\mathbf{g_{molf}}$ is equivalent to training the MLP with the VOLF algorithm.

5.4 Advantages of collapsing MOLF Hessian matrix

Sections 5.2 and 5.3 suggest that collapsing the MOLF Hessian matrix is a theoretically valid algorithm. Hence it can be put to practical use to overcome the limitations of the MOLF algorithm.

Figure 5.1 shows the number of multiplies required to solve equation (49) using OLS when the value of N_h is 40. The number of multiplies is observed to decrease cubically with decrease in N_{OLF}.

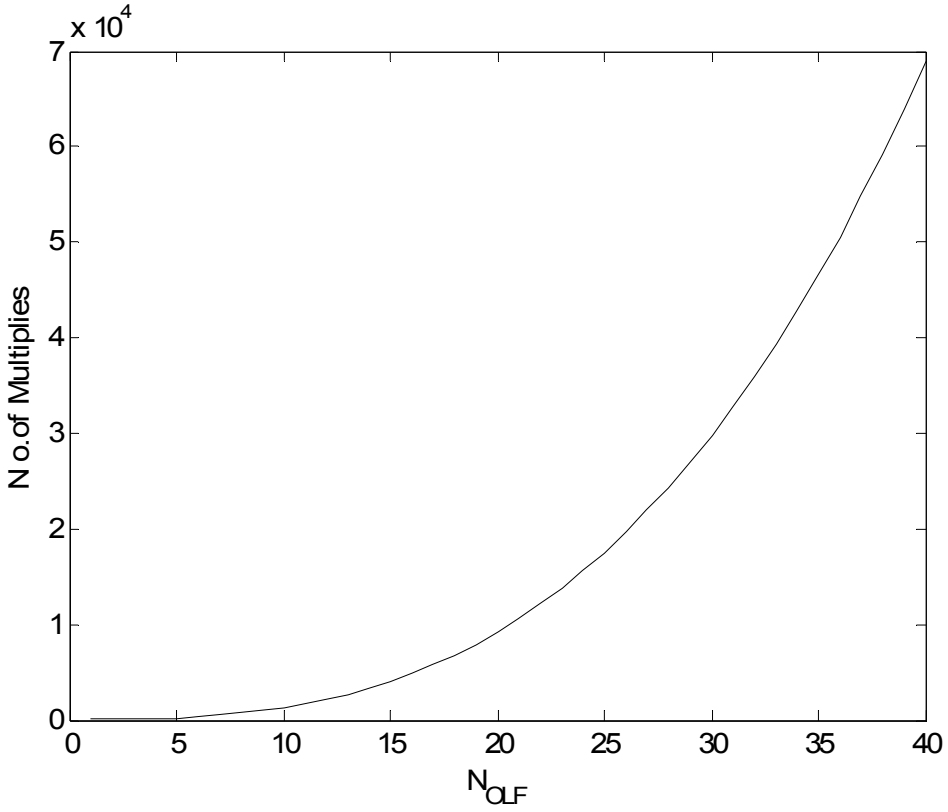

Figure 5.1 No. of multiplies vs. OLF's used

Thus collapsing the MOLF Hessian into a Hessian of smaller size significantly lowers the required number of multiplies for MLP training. It is clear from lemma 2 that the presence of

dependent hidden units makes the MOLF Hessian matrix ill-conditioned. Hence collapsing the MOLF Hessian is a reasonable solution.

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Computational Burden of different algorithms

In this section the computational burden for the different MLP training algorithms for comparison with the MOLF and VOLF algorithm is calculated. All the algorithms were implemented in Microsoft visual studio 2005.

Let $N_u = N + N_h + 1$ denote the number of weights connected to each output. The total number of weights in the network is denoted as $N_w = M(N + N_h + 1) + N_h(N + 1)$.The number of multiplies required to solve for output weights using Orthogonal Least Squares [22] is given by,

$$M_{ols}=N_u\left[M(N_u+2)+\frac{3}{4}N_u(N_u+1)\right] \tag{55}$$

The numbers of multiplies required per training iteration using BP, OWO-BP and LM are respectively given by,

$$M_{bp}= N_v[MN_u + 2N_h(N+1)+ M(N+6N_h+4)]+N_w \tag{56}$$

$$M_{owo\text{-}bp} = N_v\left[2N_h(N+2)+M(N_u+1)+ \frac{N_u(N+1)}{2}+ M(N+6N_h+4)\right] +M_{ols}+N_h(N+1) \tag{57}$$

$$M_{lm}= M_{bp} + N_v\left[MN_u\left(N_u+ 3N_h(N+1)\right)+ 4N_h^2(N+1)^2\right] + N_w^2 + N_w^3 \tag{58}$$

The multiplies required for computing the optimal learning factors of the MOLF is provided in equation (45). Similarly the number of multiplies required for the VOLF algorithm is given as,

$$M_{ols\text{-}volf} = (N_{OLF}+1)[N_{OLF}(N_{OLF}+ 2)] \tag{59}$$

The total number of multiplies required for the MOLF and VOLF algorithms are respectively given as,

$$M_{molf} = M_{owo\text{-}bp}+ N_v[N_h(N+4)- M(7N-N_h+4)]+ (N_h)^3 + M_{ols\text{-}molf} \tag{60}$$

$$M_{volf} = M_{owo\text{-}bp}+ N_v\left[N_h(N+4)- M(7N-N_h+4)\right]+ (N_h)^3 + M_{ols\text{-}volf} \tag{61}$$

Equations (60) and (61), show that the number of multiplies required in the MOLF and VOLF algorithms consists of the OWO-BP multiplies, multiplies to compute the Hessian and Jacobian, along with the number of multiplies to invert the Hessian matrix. The matrices $\mathbf{H}_{molf}$ and $\mathbf{H}_{volf}$ are respectively inverted in the MOLF and VOLF algorithms.

6.2 Experimental results and plots

Here the performance of VOLF is compared with those of MOLF, OWO-BP, LM and CG. In CG and LM, all weights are varied at each iteration. In MOLF, VOLF and OWO-BP we first solve linear equations for the output weights and subsequently update the input weights.

The data sets used for the simulations are listed below in Table 6.1. A detailed description of the different datasets is specified in Appendix A. The training for all the datasets are done on inputs normalized to zero mean and unit variance.

Table 6.1 Data set description

Data Set Name	No. of Inputs	No. of Outputs	No. of Patterns
Twod.tra	8	7	1768
Single2.tra	16	3	10000
Power12trn.tra	12	1	1414
Concrete Data Set	8	1	1030

The number of hidden units to be used in the MLP is determined by network pruning using the method of [36]. By this process the complexity of each of the data sets is analyzed and an appropriate number of hidden units is found. Then the k-fold validation procedure is used to obtain the average training and validation errors. In k-fold validation, given a data set, it is split into k non-overlapping parts of equal size, and (k − 1) parts are used for training and the remaining one part for validation. The procedure is repeated till all k combinations are exhausted. For the simulations the k value is chosen as 10. The number of iterations used for OWO-BP, CG, MOLF and VOLF is chosen as 4000. For the LM algorithm 300 iterations are chosen. For the VOLF algorithm $N_{OLF} = N_h/2$.

The average training error and the number of multiplies is calculated for every iteration in a particular dataset using the different training algorithms. These measurements are then plotted to provide a graphical representation of the efficiency and quality of the different training algorithms. These plots for different datasets are shown below.

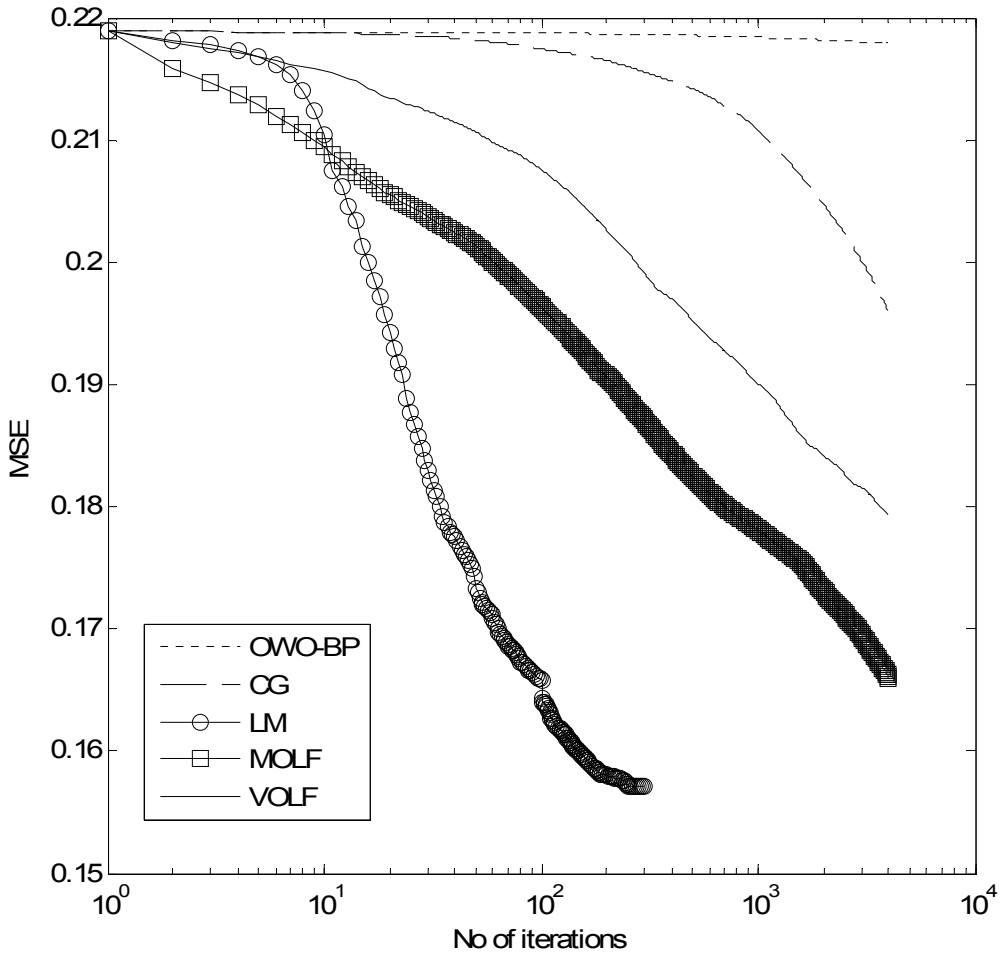

Figure 6.1 Twod.tra data set: Average error vs. iterations

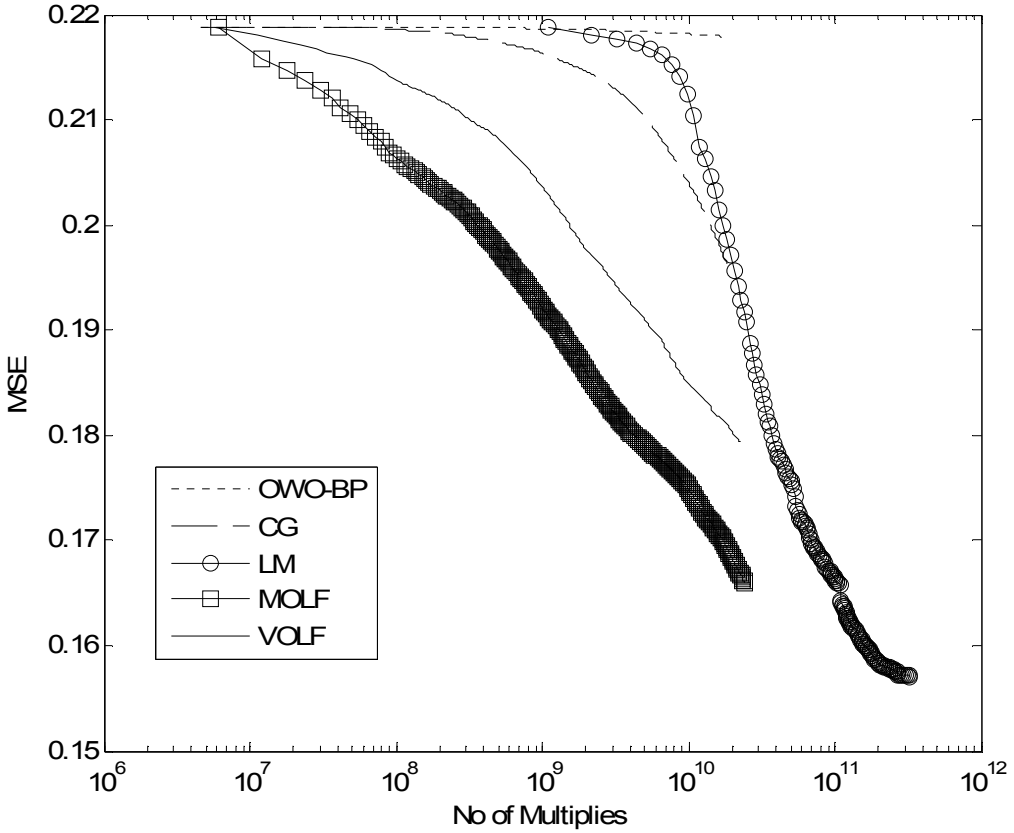

Figure 6.2 Twod.tra data set: average error vs. multiplies per iteration

For the Twod.tra data file [39], the MLP is trained with 30 hidden units. In Figure 6.1, the average mean square error (MSE) for training from 10-fold validation is plotted versus the number of iterations for each algorithm (shown on a log10 scale). In Figure 6.2, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a log10 scale).The LM algorithm show the lowest error of all the algorithms used for training, but its computational burden may make it unsuitable for training purposes. It is noted that the average error of VOLF algorithm lies between that of the OWO-BP and MOLF algorithms for every iteration.

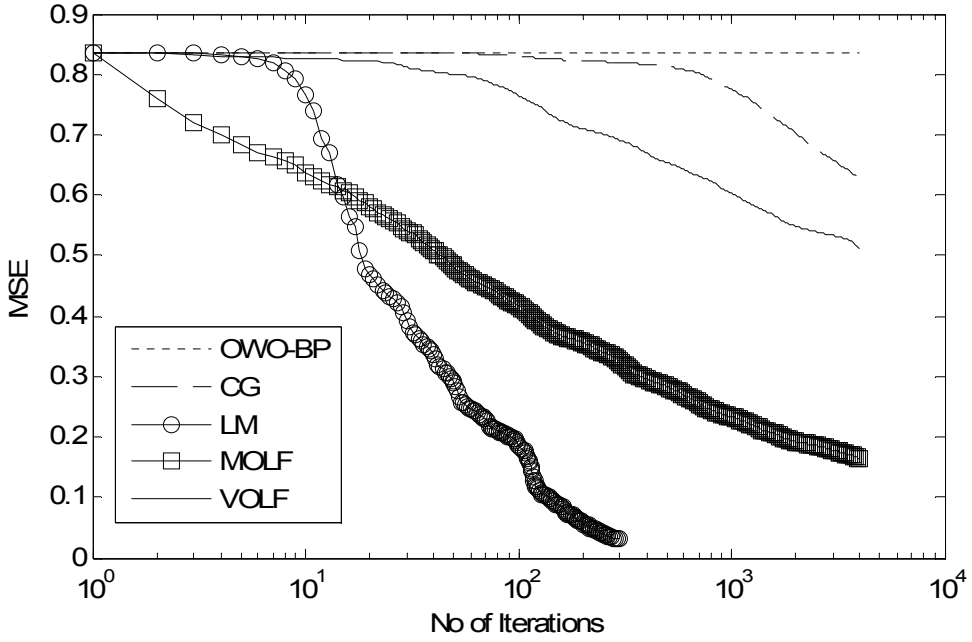

Figure 6.3 Single2.tra data set: Average error vs. iterations

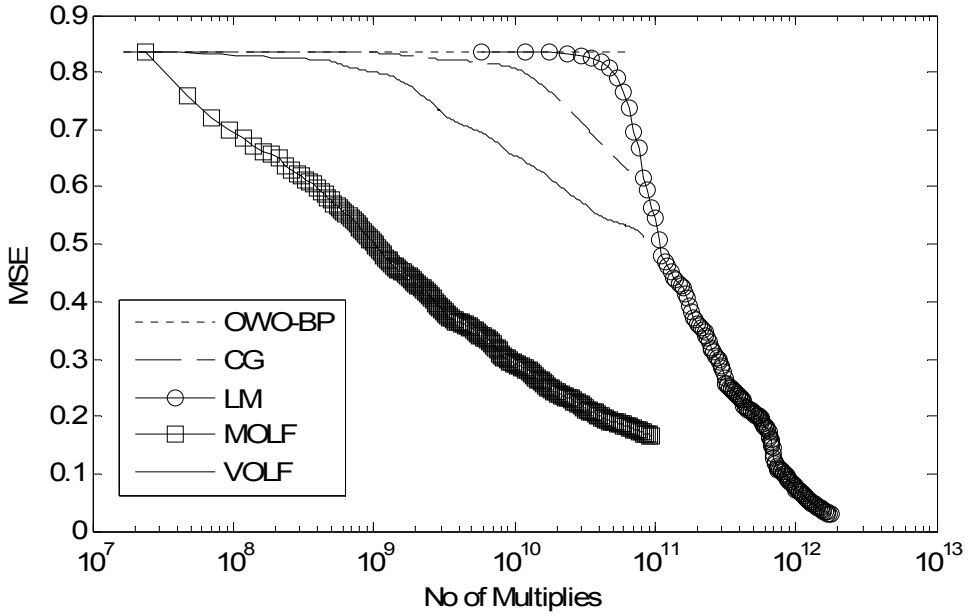

Figure 6.4 Single2.tra data set: average error vs. multiplies per iteration

For the Single2.tra data file [39], the MLP is trained with 20 hidden units. In Figure 6.3, the average mean square error (MSE) for training from 10-fold validation is plotted versus the number of iterations for each algorithm (shown on a log10 scale). In Figure 6.4, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a log10 scale).In the earlier iterations, the MOLF algorithm shows the least error. But overall, as for the previous dataset, the LM algorithm shows the lowest error of all the algorithms used for training. However its computational burden may make it unsuitable for training purposes. As before the average error of VOLF algorithm lie between that of the OWO-BP and MOLF algorithms for every iteration.

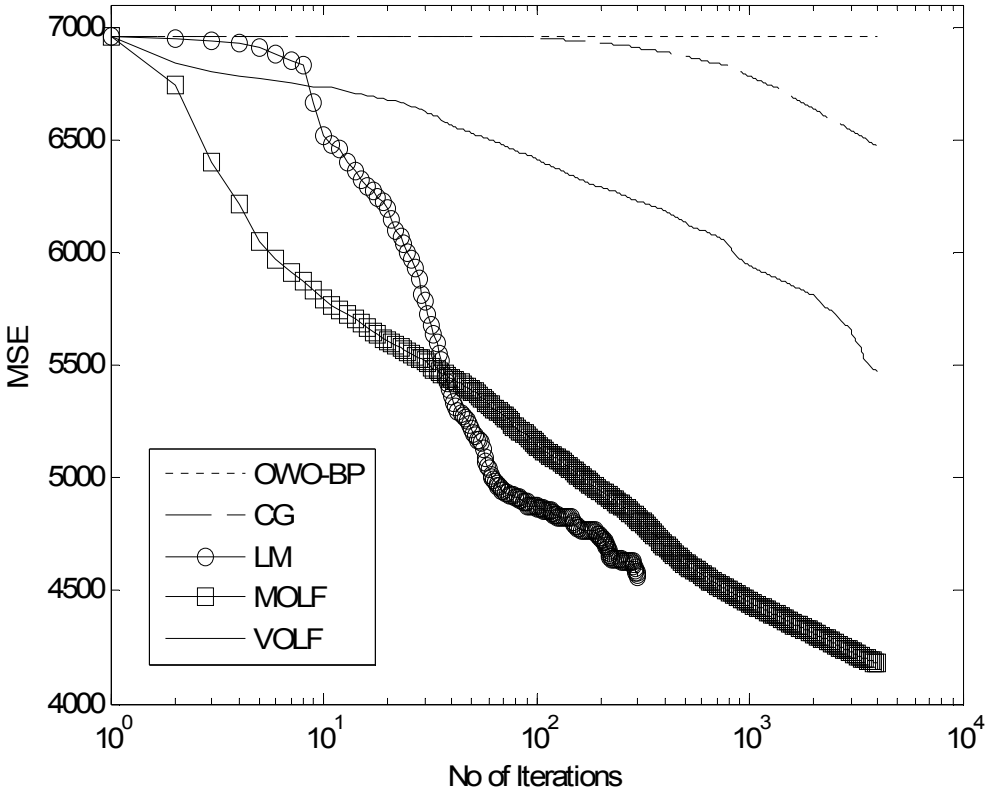

Figure 6.5 Power12trn.tra data set: Average error vs. iterations

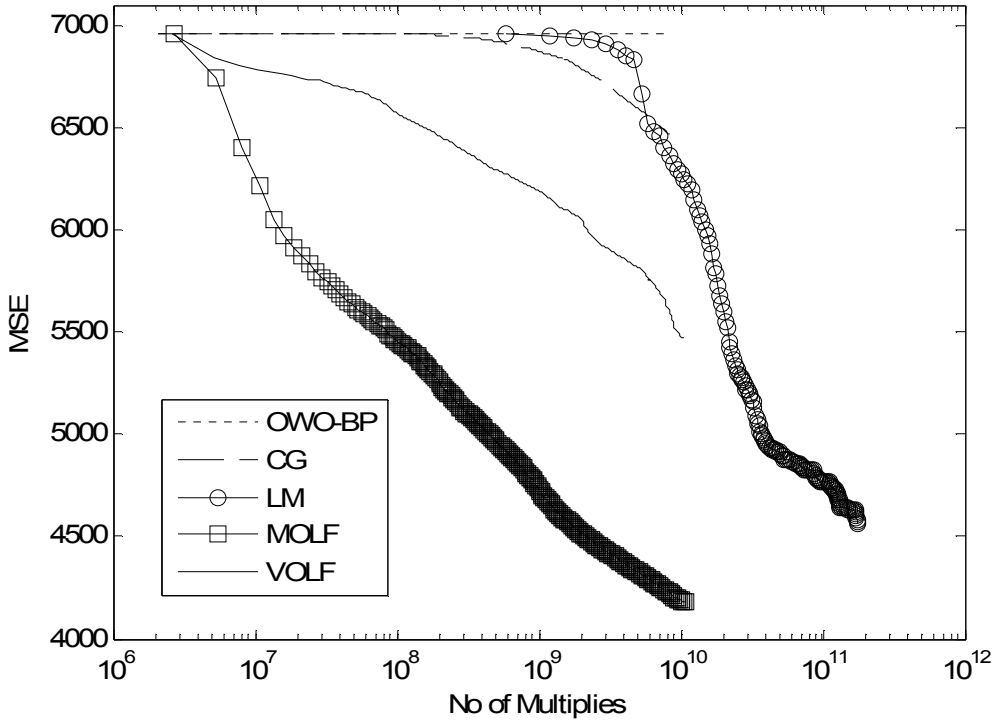

Figure 6.6 Power12trn.tra data set: Average error vs. multiplies per iteration

For the Power12trn.tra data file [39], the MLP is trained with 25 hidden units. In Figure 6.5, the average mean square error (MSE) for training from 10-fold validation is plotted versus the number of iterations for each algorithm (shown on a log10 scale). In Figure 6.6, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a log10 scale). For this dataset the MOLF algorithm performs better than all the other algorithms. As found previously, the computational requirements of the LM algorithm is very high, perhaps making it unsuitable for training. As before the average error of the VOLF algorithm lie between that of the OWO-BP and MOLF algorithms for each iteration.

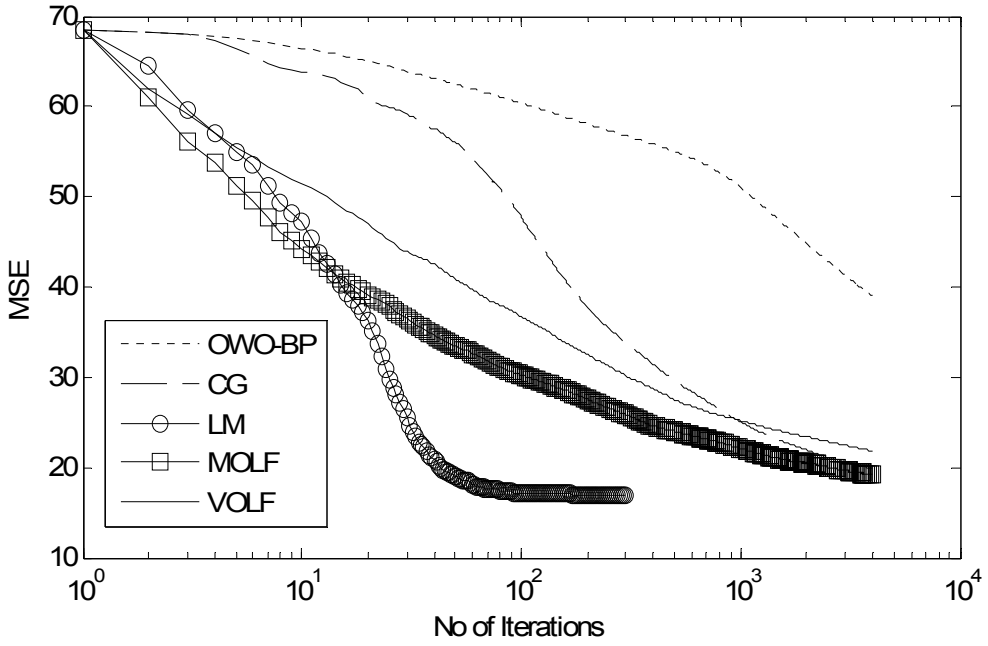

Figure 6.7 Concrete data set: Average error vs. iterations

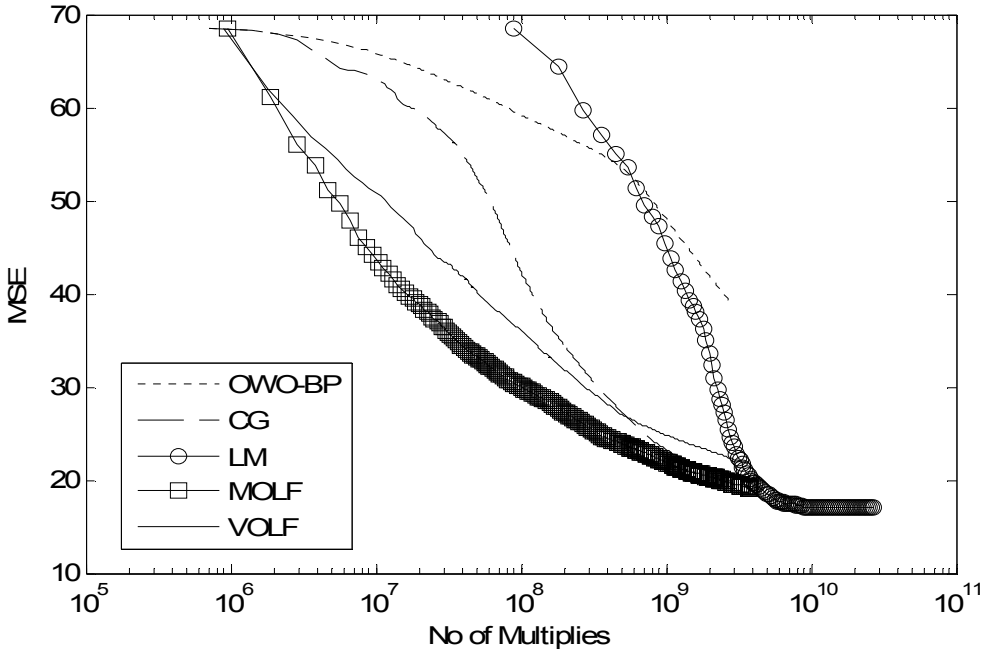

Figure 6.8 Concrete data set: average error vs. multiplies per iteration

For the Concrete data file [40], the MLP is trained with 15 hidden units. In Figure 6.7, the average mean square error (MSE) for training from 10-fold validation is plotted versus the number of iterations for each algorithm (shown on a log10 scale). In Figure 6.8, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a log10 scale).In the earlier iterations, the MOLF and VOLF algorithms shows lesser error than LM but towards the end LM has the lowest error. It is observed that in the later iterations, the error in the MOLF algorithm oscillates and doesn't decrease constantly. This could be due to the ill-conditioned nature of its Hessian matrix since the VOLF algorithm, which uses the collapsed Hessian doesn't have these characteristics. As before the average error of the VOLF algorithm lie between that of the OWO-BP and MOLF algorithms for every iteration.

Table 6.2 given below, compares the average training and validation errors of the MOLF and VOLF algorithms with the other algorithms for different data files. For each data set, the average training and validation errors are found after 10-fold validation.

Table 6.2 Average 10-fold training and validation error

Data Set		**BP-OLF**	**CG**	**MOLF**	**VOLF**	**LM**
Twod.tra	E_{trn}	0.217899	0.196059	0.166088	0.179418	0.157198
	E_{val}	0.252385	0.215986	0.198837	0.205892	0.168934
Single2.tra	E_{trn}	0.834977	0.629386	0.164867	0.512276	0.030426
	E_{val}	1.023428	0.992340	0.193423	0.644342	0.112323
Power12trn.tra	E_{trn}	6959.885485	6471.980875	4177.452590	5475.007808	4566.338420
	E_{val}	7937.212382	7132.132432	5031.234234	6712.343243	5423.535234
Concrete	E_{trn}	38.981067	19.886229	21.814297	21.560383	16.981587
	E_{val}	65.123234	56.123984	37.239453	37.212893	33.123498

From the plots and the Table presented, it can be inferred that the error found from the VOLF algorithm lies between the errors produced by the MOLF and OWO-BP algorithms. The VOLF and MOLF algorithms are also found to produce good results approaching that of the LM algorithm, with computational requirements only in the order of first order training algorithms.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The basic MOLF algorithm is reviewed, and a stronger theoretical foundation for it is given. It is now clear that MOLF is equivalent to optimally transforming the net function vector of the MLP. MOLF is found to have problems due to the presence of dependent input and hidden units, resulting in a distorted and ill-conditioned Hessian matrix. This problem is addressed by collapsing the Hessian matrix of the MOLF algorithm down to a smaller size. The collapsed version of the MOLF Hessian is proved to be equivalent to training the MLP with several hidden units per OLF. This process also results in reduced computational complexity compared to that of the MOLF algorithm. The proposed algorithm is found to interpolate between the OLF and MOLF algorithms based on the value of N_{OLF}. Simulation results show that the proposed algorithm works as stated.

Future work will include converting the proposed algorithm into a single–stage procedure where all the weights are updated simultaneously. This can be done by assigning OLF's to bypass and output weights. The algorithm can also be extended for use in MLP's with more than one hidden layer.

APPENDIX A

DESCRIPTION OF DATA SETS USED FOR TRAINING

TWOD.TRA: (8 Inputs , 7 Outputs, 1768 Training Patterns)

This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements.

The training data file contains 1768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform pdf.

SINGLE2.TRA: (16 Inputs, 3 Outputs, 10,000 Training Patterns)

This training data file consists of 16 inputs and 3 outputs and represents the training set for inversion of surface permittivity, the normalized surface rms roughness, and the surface correlation length found in back scattering models from randomly rough dielectric surfaces. The first 16 inputs represent the simulated back scattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarization. The remaining 8 are various combinations of ratios of the original eight values. These ratios correspond to those used in several empirical retrieval algorithms.

POW12TRN: (12 Inputs, 1 Output, 1414 Training Patterns)

This training file was generated using data obtained from TU Electric Company in Texas. The first ten input features are last ten minutes power load in megawatts for the entire

TU Electric utility, which covers a large part of north Texas. The output is power load fifteen minutes in the future from the current time. All powers were originally sampled every fraction of a second, and averaged over 1 minute to reduce noise. The last two inputs are respectively, the "True Area Control Error" (TACE) and the "Filtered Area Control Error" (FACE). The FACE is a combination of exponentially filtered TACE and moving average filtered TACE.

Concrete: (12 Inputs, 1 Output, 1414 Training Patterns)

This data file is available on the UCI Machine Learning Repository [40]. It contains the actual concrete compressive strength (MPa) for a given mixture under a specific age (days) determined from laboratory. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, super plasticizer, coarse aggregate, and fine aggregate. The data set consists of 8 inputs and one output per pattern, with a total of 1030 patterns.

REFERENCES

[1] R. C. Odom, P. Pavlakos, S.S. Diocee, S. M. Bailey, D. M. Zander, and J. J. Gillespie, "Shaly sand analysis using density-neutron porosities from a cased-hole pulsed neutron system," SPE Rocky Mountain regional meeting proceedings: Society of Petroleum Engineers, pp. 467-476, 1999.

[2] A. Khotanzad, M. H. Davis, A. Abaye, D. J. Maratukulam, "An artificial neural network hourly temperature forecaster with applications in load forecasting," IEEE Transactions on Power Systems, vol. 11, No. 2, pp. 870-876, May 1996.

[3] S. Marinai, M. Gori, G. Soda, "Artificial neural networks for document analysis and recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, No. 1, pp. 23-35, 2005.

[4] J. Kamruzzaman, R. A. Sarker, R. Begg, Artificial Neural Networks: Applications in Finance and Manufacturing, Idea Group Inc (IGI), 2006.

[5] L. Wang and X. Fu, Data Mining With Computational Intelligence, Springer-Verlag, 2005.

[6] K. Hornik, M. Stinchcombe, and H.White, "Multilayer Feedforward Networks Are Universal Approximators." Neural Networks, Vol. 2, No. 5, 1989, pp.359-366.

[7] K. Hornik, M. Stinchcombe, and H. White, "Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks," Neural Networks,vol. 3,1990, pp.551-560.

[8] Michael T.Manry, Steven J.Apollo, and Qiang Yu, "Minimum Mean Square Estimation and Neural Networks," Neurocomputing, vol. 13, September 1996, pp.59-74.

[9] G. Cybenko, "Approximations by superposition of a sigmoidal function," Mathematics of Control, Signals, and Systems (MCSS), vol. 2, pp. 303-314, 1989.

[10] D. W. Ruck et al., “The multi-layer perceptron as an approximation to a Bayes optimal discriminant function,” IEEE Transactions on Neural Networks, vol. 1, No. 4, 1990.

[11] Q. Yu, S.J. Apollo, and M.T. Manry, “MAP estimation and the multi-layer perceptron,” Proceedings of the 1993 IEEE Workshop on Neural Networks for Signal Processing, pp. 30-39, Linthicum Heights, Maryland, Sept. 6-9, 1993.

[12] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “Learning internal representations by error propagation,” in D.E. Rumelhart and J.L. McClelland (Eds.), Parallel Distributed Processing, vol. I, Cambridge, Massachusetts: The MIT Press, 1986.

[13] J.P. Fitch, S.K. Lehman, F.U. Dowla, S.Y. Lu, E.M. Johansson, and D.M. Goodman, "Ship Wake-Detection Procedure Using Conjugate Gradient Trained Artificial Neural Networks," IEEE Trans. on Geoscience and Remote Sensing, Vol. 29, No. 5, September 1991, pp. 718-726.

[14] Changhua Yu, Michael T. Manry, and Jiang Li, "Effects of nonsingular pre-processing on feed-forward network training ". International Journal of Pattern Recognition and Artificial Intelligence , Vol. 19, No. 2 (2005) pp. 217-247.

[15] S. McLoone and G. Irwin, “A variable memory Quasi-Newton training algorithm,” Neural Processing Letters, vol. 9, pp. 77-89, 1999.

[16] A. J. Shepherd Second-Order Methods for Neural Networks, Springer-Verlag New York, Inc., 1997.

[17] K. Levenberg, “A method for the solution of certain problems in least squares,” Quart. Appl. Math., Vol. 2, pp. 164.168, 1944.

[18] D. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” SIAM J. Appl. Math., Vol. 11, pp. 431.441, 1963.

[19] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, ”Improving the Convergence of the Backpropagation Algorithm Using Learning Rate Adaptation Methods,” Neural Computation, Vol. 11, No. 7, Pages 1769-1796, October 1, 1999.

[20] Sanjeev Malalur, M. T. Manry, "Multiple Optimal Learning Factors for Feed-forward Networks," accepted by The SPIE Defense, Security and Sensing (DSS) Conference, Orlando, FL, April 2010

[21] W. Kaminski, P. Strumillo, “Kernel orthonormalization in radial basis function neural networks,” IEEE Transactions on Neural Networks, vol. 8, Issue 5, pp. 1177 - 1183, 1997

[22] D.E Rumelhart, G.E. Hinton, and R.J. Williams, “Learning representations of back-propagation errors,” Nature, London, 1986, vol. 323, pp. 533-536

[23] D.B.Parker, “Learning Logic,” Invention Report S81-64,File 1, Office of Technology Licensing, Stanford Univ., 1982

[24] T.H. Kim “Development and Evaluation of Multilayer Perceptron Training Algorithms”, Phd. Dissertation, The University of Texas at Arlington, 2001.

[25] S.A. Barton, “A matrix method for optimizing a neural network,” Neural Computation, vol. 3, no. 3, pp. 450-459, 1991.

[26] Nocedal, Jorge & Wright, Stephen J. (1999). Numerical Optimization. Springer-Verlag

[27] Bonnans, J. F., Gilbert, J.Ch., Lemaréchal, C. and Sagastizábal, C.A. (2006), Numerical optimization, theoretical and numerical aspects.Second edition. Springer

[28] R. Fletcher, "Conjugate Direction Methods," chapter 5 in Numerical Methods for Unconstrained Optimization, edited by W. Murray, Academic Press, New York, 1972.

[29] P.E. Gill, W. Murray, and M.H. Wright, Practical Optimization, Academic Press, New York,1981.

[30] Charytoniuk, W.; Chen, M.-S.; "Very short-term load forecasting using artificial neural networks ," Power Systems, IEEE Transactions on , vol.15, no.1, pp.263-268, Feb 2000

[31] Ning Wang; Xianyao Meng; Yiming Bai;,"A fast and compact fuzzy neural network for online extraction of fuzzy rules," Control and Decision Conference, 2009. CCDC '09. Chinese , vol.,no.,pp.4249-4254,17-19 June 2009

[32] Wei Wu; Guorui Feng; Zhengxue Li; Yuesheng Xu;, "Deterministic convergence of an online gradient method for BP neural networks," Neural Networks, IEEE Transactions on , vol.16, no.3, pp.533-540, May 2005

[33] Saurabh Sureka, Michael Manry, A Functional Link Network With Ordered Basis Functions, Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007.

[34] Lee, Y.; Oh, S.-H,; Kim, M.W., "The effect of initial weights on premature saturation in back-propagation learning," Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on, vol.1, 1991 pp. 765-770 vol. 1

[35] P.L Narasimha, S. Malalur, and M.T. Manry, Small Models of Large Machines, Proceedings of the TwentyFirst International Conference of the Florida AI Research Society, pp.83-88, May 2008.

[36] Pramod L. Narasimha, Walter H. Delashmit, Michael T. Manry, Jiang Li, Francisco Maldonado, An integrated growing-pruning method for feedforward network training, Neurocomputing vol. 71, pp. 2831–2847, 2008.

[37] R.P Lippman, "An introduction to computing with Neural Nets," IEEE ASSP Magazine,April 1987.

[38] S. S. Malalur and M. T. Manry, Feedforward Network Training Using Optimal Input Gains, Proc.of IJCNN'09, Atlanta, Georgia, pp. 1953-1960, June 14-19, 2009.

[39] Univ. of Texas at Arlington, Training Data Files – http://wwwee.uta.edu/eeweb/ip/training_data_files.html.

[40] Univ. of California, Irvine, Machine Learning Repository - http://archive.ics.uci.edu/ml/

BIOGRAPHICAL INFORMATION

Praveen Jesudhas completed his bachelor's in Electrical engineering from SSN college of engineering, Anna University, Chennai, India in 2008. He completed his masters in Electrical engineering from the University of Texas at Arlington in 2010. He has worked as a software application developer at Inautix Technologies in Chennai, India and also completed an internship as a pattern recognition intern in FastVDO MD, USA. He is a graduate student member of the Tau Beta Pi honor society. His current research interests include image processing, pattern recognition and machine learning.