OPTIMAL OUTPUT GAIN ALGORITHM FOR FEED FORWARD NETWORK TRAINING

by

BABU HEMANTH KUMAR ASWATHAPPA

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

Copyright © by Babu Hemanth Kumar Aswathappa 2010

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr Michael T. Manry, for supervising me on this thesis. Throughout my research period he provided encouragement, sound advice, good teaching, good company, and lots of good ideas that helped make my research easy and very interesting. I am really thankful to him for patiently spending so many lab hours with me and even answering all my questions over the phone when I was doing my internship at Intel. I sincerely believe he is a very good counselor one can ever hope for. It is an honor for me to be a part of his IPNNL research group.

I would like thank Dr. Rao and Dr. Wei-Jen Lee for taking interest in my work and accepting to be a part of my thesis defense committee. I would like to thank my manager and my team at Intel Corporation, Chandler, Arizona. It is in their company that I gained a real-world industry exposure.

I must express my sincere gratitude to my family for all their love and support. My parents Mr. Aswathappa and Mrs. Suvarna, for all their love, faith and support. My sister and my brother, Mamatha and Gopal, for being there for me whenever I needed them. I dedicate this thesis to them, whose prayers made this possible.

I take this moment to thank all my friends Harish Prakash, Prashanth.R.V, Mahadevkirthi, Hemanth, Adarsh, Bharath, Rohan, Kashyap, Kiran, Manju, Shreyanka, Pavithra and Deepa for being with me through all the times of struggle and celebration.

I would never forget the support of my beloved friends back home in India - Balaji, Nithin, Karthik, Arun, Nikki, Chaithali, Nisha and Geetanjali. Thank you all so much for all the support across the miles.

October 29, 2010

ABSTRACT

OPTIMAL OUTPUT GAIN ALGORITHM FOR FEEDFORWARD NETWORK TRAINING,

Babu Hemanth Kumar Aswathappa, M.S.

The University of Texas at Arlington, 2010

Supervising Professor: Michael T Manry

A batch training algorithm for feed-forward networks is proposed which uses Newton's method to estimate a vector of optimal scaling factors for output errors in the network. Using this vector, backpropagation is used to modify weights feeding into the hidden units. Linear equations are then solved for the network's output weights. Elements of the new method's Gauss-Newton Hessian matrix are shown to be weighted sums of elements from the total network's Hessian. The effect of output transformation on training a feed-forward network is reviewed and explained, using the concept of equivalent networks. In several examples, the new method performs better than backpropagation and conjugate gradient, with similar numbers of required multiplies.

The method performs about as well as Levenberg-Marquardt, with several orders of magnitude fewer multiplies due to the small size of its Hessian.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

The multilayer perceptron (MLP) has been shown to have several properties that make it of interest to investigators. First, it can be trained using gradient approaches such as back propagation (BP) [1] and Levenburg-Marquardt (LM) [2]. It has the universal approximation property [3]. With proper training, the MLP approximates the Bayes classifier [4] or the minimum mean square error (mmse) estimator [5]. The MLP has found use in many applications including character recognition [6][7], power load forecasting[8], prognostics [9], well logging [10], and data mining[11].

Unfortunately, MLP training is sensitive to many parameters of the network and its training data, including the input means and the initial network weights. In addition, MLP training is sensitive to the collinearity of its inputs [12] and outputs.

In this paper, a fast, convergent training algorithm is developed which attempts to compensate for output collinearity. In section II, matrix-vector notation is introduced for BP in the MLP network's input weights, transforms of the network output vectors are analyzed, and a training algorithm incorporating backpropagation [1] (BP) is described. The Optimal Output Gain (OOG) algorithm which utilizes Newton's method is described in section III and analyzed in section IV. A combination of the optimal output gains and optimal learning factor (OLF) are then found, using Newton's method.

Computational burden for all the algorithms are also discussed. In section V numerical results for OOG and other training algorithms are presented for comparison. Conclusions and future work are shown in section VI.

CHAPTER 2

MULTILAYER PERCEPTRON

2.1 Introduction

In this chapter we introduce MLP notation and describe a training algorithm based upon BP [1].

2.2 MLP Notation

In the fully connected MLP of Fig 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.

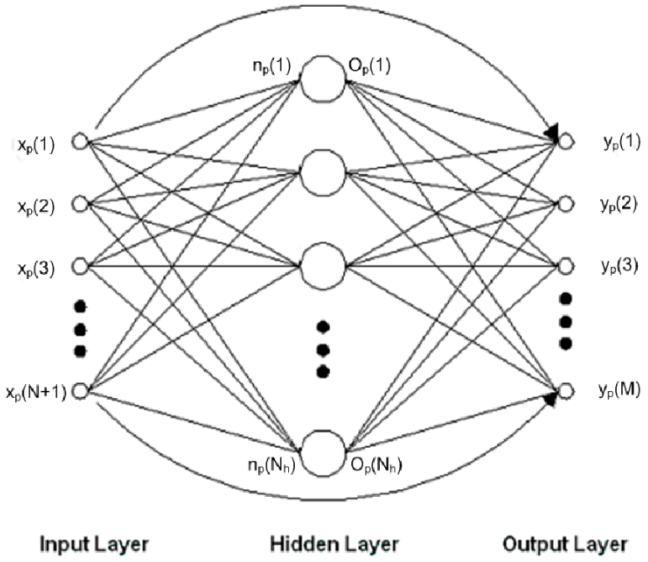

Fig 2.1: A fully connected multi-layer perceptron

In order to handle thresholds in the hidden and output layers, the input vectors are augmented by an extra element $x_p(N+1)$ which is equal to 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 in the network. The vector of hidden layer net functions, $\mathbf{n}_p$ and the actual network output vector $\mathbf{y}_p$ can be written as

$$\mathbf{n}_p = \mathbf{W} \cdot \mathbf{x}_p, \tag{2.1}$$

$$\mathbf{y}_p = \mathbf{W}_{oi} \cdot \mathbf{x}_p + \mathbf{W}_{oh} \cdot \mathbf{o}_p \tag{2.2}$$

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

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}[t_p(m) - y_p(m)]^2 \tag{2.3}$$

Here $t_p(m)$ and $y_p(m)$ respectively denote the m^{th} desired and actual outputs for the p^{th} pattern.

2.3 Training using OWO-BP

Here we introduce a training algorithm known as Output Weight Optimization-Backpropagation (OWO-BP). This algorithm alternately solves linear equations for the network's output weights and separately trains the input weights using BP.

Output Weight Optimization (OWO) is a technique to solve for weights connected to the actual outputs of the network [12]. 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 (2.2) can be re-written as

$$\mathbf{y}_p = \mathbf{W}_o \cdot \mathbf{x}_p \tag{2.4}$$

where $\mathbf{x}_p = [\mathbf{x}_p^T, \mathbf{o}_p^T]$ is the augmented input vector and $\mathbf{W}_o = [\mathbf{W}_{oi} : \mathbf{W}_{oh}]$ denotes all the output weights. $\mathbf{x}_p$ is a column vector of size N_u where $N_u = N + N_h + 1$ and $\mathbf{W}_o$ is M by N_u. The output weights can be solved by setting $\partial E / \partial \mathbf{W}_o = 0$ which leads to a set of linear equations given by

$$\mathbf{C}_a = \mathbf{R}_a \cdot \mathbf{W}^T{}_o \tag{2.5}$$

where,

$$\mathbf{C}_a = \frac{1}{N_v}\sum_{p=1}^{N_v}\mathbf{x}_p^T t^T{}_p, \tag{2.6}$$

$$\mathbf{R}_a = \frac{1}{N_v}\sum_{p=1}^{N_v}\mathbf{x}_p\mathbf{x}_p^T \tag{2.7}$$

Equation (2.5) is most easily solved using orthogonal least squares [13] (OLS) which is equivalent to using the QR decomposition [14] . In the second half of an OWO-BP iteration, the input weight matrix **W** is updated as

$$\mathbf{W} \leftarrow \mathbf{W} + z\cdot\mathbf{G} \tag{2.8}$$

where **G** is the generic representation of a *direction matrix* that contains information about the direction of learning and z, the learning factor contains information about the step length to be taken in the direction **G**. The weight update $z\cdot\boldsymbol{G}$ in equation (2.8) can also be denoted as

$$z\cdot\mathbf{G} = \Delta\mathbf{W} \tag{2.9}$$

For backpropagation [1] , the direction matrix is nothing but the N_h by (N+1) negative input weight Jacobian matrix computed as

$$\mathbf{G} = \frac{1}{N_v}\sum_{p=1}^{N_v}\boldsymbol{\delta}_p\cdot\mathbf{x}_p^T \tag{2.10}$$

Here $\boldsymbol{\delta}_p = [\delta_p(1)\ ,\ \delta_p(2)\ldots\ ,\ \delta_p(N_h)\ ]^T$ is the N_h by 1 column vector of hidden unit delta functions [1]. A description of OWO-BP is given below. For every training epoch

i. Solve the system of linear equations in (2.6) and (2.7) using OLS and update the output weights, $\mathbf{W}_o$

ii. Find the negative Jacobian matrix **G** described in equation (2.10)

iii. Update the input weights, **W**, using equation (2.8)

This method is attractive for several reasons. First, the training is faster than ordinary BP since weights connected to the outputs are found by solving linear equations. Second, it helps us avoid some local minima. Third, the method's **G** matrix and learning factors can be changed without damaging the performance unlike conjugate gradient (CG) [15].

2.4 Optimal Learning Factor

The choice of learning factor z in equation (2.8) has a direct effect on the convergence rate of OWO-BP. Early steepest descent methods used a constant learning factor, which resulted in slow convergence. Later methods have used a heuristic scaling approach to modify the learning factor between iterations and thus speed up the rate of convergence. However, using Newton's method for one unknown, a non-heuristic *optimal learning factor* (OLF) for OWO-BP can be derived as,

$$z = \frac{-\partial E / \partial z}{\partial^2 E / \partial z^2} \tag{2.11}$$

where the numerator and denominator partial derivatives are evaluated at z=0. The expression for the second derivative of the error with respect to the OLF is found using (2.3) as,

$$\frac{\partial^2 E}{\partial z^2} = \sum_{k=1}^{N_h}\sum_{j=1}^{N_h}\sum_{n=1}^{N+1} g(k,n)\sum_{i=1}^{N+1}\frac{\partial^2 E}{\partial w(k,n)\partial w(j,i)} g(j,i) = \sum_{k=1}^{N_h}\sum_{j=1}^{N_h}\mathbf{g}_k^T \mathbf{H}_R^{k,j}\mathbf{g}_j \tag{2.12}$$

where column vector $\mathbf{g}_k$ contains elements g(k,n) of **G**, for all values of n. $\mathbf{H}_R$ is the reduced size input weight Hessian with N_{iw} rows and columns, where $N_{iw} = (N + 1)\cdot N_h$ is the number of input weights. $\mathbf{H}_R(k, j)$ 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 [11] updates are used, elements of $\mathbf{H}_R$ are computed as

$$\frac{\partial^2 E}{\partial w(j,i)\partial w(k,n)} = \frac{2}{N_v} u(j,k)\sum_{p=1}^{N_v} x_p(i)x_p(n)o'_p(j)o'_p(k) \tag{2.13}$$

where,

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

$o'_p(k)$ indicates the first partial derivative of $o_p(k)$ with respect to its net function. Because (2.13) represents the Gauss-Newton approximation to the Hessian, it is positive semi-definite. Equation (2.12) 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, and (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 for use in Newton's algorithm.

2.5 Equivalent Networks

Definition: Two networks, one trained on original data $\{\mathbf{x}_p, \mathbf{t}_p\}$ and the other on linearly transformed data $\{\mathbf{z}_p, \mathbf{t}'_p\}$ are **strongly equivalent** if the outputs of the two networks are identical and the transforms are invertible

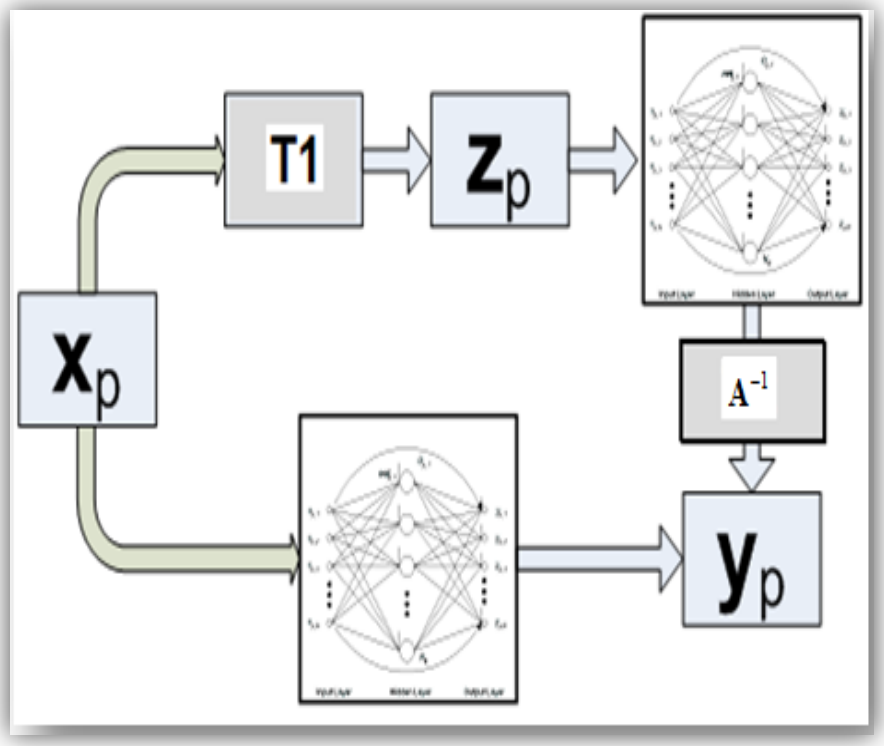

Fig 2.2. Equivalent Networks

2.6 Effects of output transformations

In this section we use the concept of equivalent networks to analyze the effects of transforming the desired output vectors $\mathbf{t}_p$

Given a network MLP-1 and its weights, $\mathbf{W}$, $\mathbf{W}_{oi}$, and $\mathbf{W}_{oh}$ we train the network by minimizing the error function E($\mathbf{W}$, $\mathbf{W}_{oi}$, $\mathbf{W}_{oh}$), which may be the standard MSE. Training data for this first network is $\{\mathbf{x}_p , \mathbf{t}_p\}$ for $1 \le p \le N_v$,

where dim($\mathbf{x}$) = (N+1) and

dim ($\mathbf{t}$) = M. Let MLP-1, be trained using the input vectors $\mathbf{x}_p$ and output vector $\mathbf{t}_p$ as described in section 2.2

MLP 1 has parameters $\mathbf{x}_p$, $\mathbf{t}_p$, $\mathbf{y}_p$, $\mathbf{R}$, $\mathbf{W}_o$, $\mathbf{C}$ and $\boldsymbol{\delta}_{po}$

Consider a second network denoted as MLP 2 , trained using data $\{\mathbf{x}_p , \mathbf{t}'_p\}$ where $\mathbf{t}'_p = \mathbf{A} \cdot \mathbf{t}_p$ and $\mathbf{A}$ is an M' by M rectangular transformation matrix and M' may be unequal to M. MLP 2 is equivalent to MLP 1 in the sense that its input weight matrix $\mathbf{W}'$ is equal to $\mathbf{W}$ in MLP 1 and its net and activation vectors $\mathbf{n}'_p$ and $\mathbf{O}'_p$ therefore satisfy

$$\mathbf{n}'_p = \mathbf{n}_p \tag{2.14}$$

$$\mathbf{O}'_p = \mathbf{O}_p \tag{2.15}$$

While training $\mathbf{W}'$ in *MLP 2* using BP its output delta function is given by

$$\boldsymbol{\delta}'_{po} = 2[\mathbf{t}'_p - \mathbf{y}'_p] \tag{2.16}$$

$$\boldsymbol{\delta}'_{po} = \mathbf{A} \cdot \boldsymbol{\delta}_{po} \tag{2.17}$$

For MLP 1 the delta function for each hidden unit is given by

$$\delta_p(k) = f'(n_p(k)) \sum_{i=1}^{M} W_{oh}(i,k) \cdot \delta_{po}(i) \tag{2.18}$$

which can be expressed in matrix vector form as

$$\boldsymbol{\delta}_p = \mathbf{T} \cdot \mathbf{W}_{oh}{}^T \cdot \boldsymbol{\delta}_{po} \tag{2.19}$$

$$\mathbf{T} = \begin{bmatrix} f'[n_p(1)] & 0 & 0\ldots & 0 \\ 0 & f'[n_p(2)] & 0\ldots & 0 \\ 0 & 0 & f'[n_p(3)] & 0 \\ \vdots & \vdots & 0 \quad \ddots & 0 \\ \vdots & \vdots & \vdots \quad \ddots & \\ 0 & 0 & 0 & f'[n_p(k)] \end{bmatrix} \tag{2.20}$$

The delta matrix for *MLP 2* can similarly be written as

$$\boldsymbol{\delta}'_p = \mathbf{T} \cdot \mathbf{W}'_{oh}{}^T \cdot \boldsymbol{\delta}'_{po} \tag{2.21}$$

$$= \mathbf{T} \cdot \mathbf{W}_{oh}{}^T \cdot (\mathbf{A} \cdot \mathbf{A}^T) \cdot \boldsymbol{\delta}_{po} \tag{2.22}$$

which leads to

$$\boldsymbol{\delta}'_p = (\mathbf{A} \cdot \mathbf{A}^T) \cdot \boldsymbol{\delta}_p \tag{2.23}$$

$$\boldsymbol{\delta}'_p = \mathbf{R} \cdot \boldsymbol{\delta}_p$$

Where

$$\mathbf{R} = \mathbf{A} \cdot \mathbf{A}^T \tag{2.24}$$

The negative Jacobian matrix for training input weights in *MLP-1* is given in (2.10).

The negative Jacobian for training input weights in *MLP-2* is then

$$\mathbf{G}' = \frac{1}{N_v} \sum_{p=1}^{N_v} \boldsymbol{\delta}'_p \cdot \mathbf{x}_p^T \tag{2.25}$$

$$= \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{R} \cdot \boldsymbol{\delta}_p \cdot \mathbf{x}_p^T \tag{2.26}$$

$$= \mathbf{R} \cdot \frac{1}{N_v} \sum_{p=1}^{N_v} \boldsymbol{\delta}_p \cdot \mathbf{x}_p^T \tag{2.27}$$

$$= \mathbf{R} \cdot \mathbf{G} \tag{2.28}$$

2.6.1 Lemma 1

A square matrix $\mathbf{A}$ has no effect on training, if it is Orthogonal.

If $\mathbf{A}$ is orthogonal, then

$$\mathbf{A} \cdot \mathbf{A}^T = \mathbf{R} = \mathbf{I} \tag{2.29}$$

Hence we get

$$\boldsymbol{\delta}'_p = \boldsymbol{\delta}_p \tag{2.30}$$

$$\mathbf{G}' = \mathbf{G} \tag{2.31}$$

If the delta functions and the negative gradients are same then the training is no different than in *MLP-1*. Hence $\mathbf{A}$ does not improve training. Orthogonal transform matrices are therefore useless. This also proves that the conventional backpropagation does not optimize effects of desired outputs on training.

2.6.2 Lemma 2

If $\mathbf{A}$ is Non-Orthogonal, training of *MLP 1* and *MLP 2* diverge in the sense that they become no longer strictly equivalent

If $\mathbf{A}$ is a *M' by M* rectangular transformation matrix, for some *M' > M* (The extra rows are not zero rows) then $\mathbf{R} \neq \mathbf{I}$

This implies that $\boldsymbol{\delta}'_p \neq \boldsymbol{\delta}_p$ and $\mathbf{G}' \neq \mathbf{G}$. Hence there is an effect on *MLP* training.

From Lemma 2 it is clear that $\mathbf{A}$ should be a non-orthogonal matrix

2.6.3 Lemma 3

Given **R**, the number of A matrices is uncountably infinite

$$\mathbf{R} = \mathbf{A} \cdot \mathbf{A}^T$$

From the above equation for one value of R, we can find infinitely many solutions

Here we look for **R** and not for **A** . There might be only one value of **R**

2.6.4 Lemma 4

For the case M=1, OOG behaves same as OWO-BP algorithm

CHAPTER 3

OPTIMAL OUTPUT GAIN ALGORITHM

In this section, we discuss the OOG derivation, the utility of non-singular, non-orthogonal **A** matrices and steps for the algorithm. We also discuss the OOG-HWO algorithm which is an improvement to OOG algorithm.

3.1 OOG Derivation

The matrix notation for the mean-squared error for *MLP-2* is given by

$$\mathrm{E}' = \frac{1}{N_v}\sum_{p=1}^{Nv}\left\|\mathbf{t}'_p - \mathbf{y}'_p\right\|^2 \tag{3.1}$$

$$= \frac{1}{N_v}\sum_{p=1}^{Nv}(\mathbf{t}_p - \mathbf{y}_p)^T \cdot \mathbf{R} \cdot (\mathbf{t}_p - \mathbf{y}_p) \tag{3.2}$$

Let $\mathbf{R}$ be a diagonal matrix with diagonal elements $r(i)$, Then

$$\mathrm{E}' = \frac{1}{N_v}\sum_{p=1}^{Nv}\sum_{i=1}^{M} r(i)[\mathbf{t}_p(i) - \mathbf{y}_p(i)]^2 \tag{3.3}$$

where the gains r(i) are usually positive.

For the pth pattern, output and hidden layer delta functions are respectively found as

$$\delta_{po}(k) = 2r(i)[t_p(i) - y_p(i)] = r(i)\delta_{po}(i) \tag{3.4}$$

This can also be written as,

$$\delta'_p(k) = f'(n_{pk}) \sum_{i=1}^{M} \delta'_{po}(i) w_{oh}(i,k) \tag{3.5}$$

Using equation (3.4), the above equation can be expressed as,

$$= \sum_{i=1}^{M} r(i) f'(n_{pk}) \delta_{po}(i) w_{oh}(i,k) \tag{3.6}$$

$$= \sum_{i=1}^{M} r(i) \delta_p(i,k) \tag{3.7}$$

From equation (3.6) and (3.7), the delta function can be written as,

$$\delta_p(i,k) = f'(n_{pk}) \delta_{po}(i) w_{oh}(i,k) \tag{3.8}$$

Now, the negative gradient of E' is

$$g'(k,n) = \frac{-\partial E'}{\partial w(k,n)} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} r(i) \delta_p(i,k) x_p(n) \tag{3.9}$$

$$= \sum_{i=1}^{M} r(i) \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p(i,k) x_p(n) = \sum_{i=1}^{M} r(i) g_i(k,n), \tag{3.10}$$

$$g_i(k,n) = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p(i,k) x_p(n), \tag{3.11}$$

where $g(k,n) = g'(k,n)$ for r(i) =1.

The matrix of negative partial derivatives can be written as

$$\mathbf{G}' = \sum_{i=1}^{M} r(i) \mathbf{G}_i \tag{3.12}$$

Since $\mathbf{G}'$ is used to change hidden units identical to those in the original network, we get

$$\mathbf{W} \longleftarrow \mathbf{W} + z \cdot \mathbf{G}' \tag{3.13}$$

So,

$$\Delta \mathrm{W} = z \cdot \mathbf{G}' \tag{3.14}$$

where z is the learning factor.

The error function being minimized with respect to the r(i)s is

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2 \tag{3.15}$$

where

$$y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n) x_p(n) + \sum_{k=1}^{Nh} w_{oh}(i,k) f\left(\sum_{n=1}^{N+1} (w(k,n) + \sum_{j=1}^{M} r(j) g_j(k,n) x_p(n))\right) \tag{3.16}$$

The first partial of E with respect to r is a vector $\mathbf{g}$ which is defined as

$$\mathbf{g} = \frac{\partial E}{\partial r} \tag{3.17}$$

Therefore the elements of $\mathbf{g}$ can be found as,

$$g(m) \equiv \frac{\partial E}{\partial r(m)} = \frac{-2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] \frac{\partial y_p(i)}{\partial r(m)} \tag{3.18}$$

$$\frac{\partial y_p(i)}{\partial r(m)} = \sum_{k=1}^{N_h} w_{oh}(i,k) f'(n_p(k)) \sum_{n=1}^{N+1} g_m(k,n) x_p(n) \tag{3.19}$$

$$\frac{\partial y_p(i)}{\partial r(m)} = v_p(i,m) \tag{3.20}$$

The second partials comprising the M by M Hessian matrix are

$$h_{og}(m,u) \equiv \frac{\partial^2 E}{\partial r(m)\partial r(u)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial r(m)}\frac{\partial y_p(i)}{\partial r(u)} \tag{3.21}$$

Using the equation (3.19), we can express the Hessian as,

$$= \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\sum_{k=1}^{N_h} w_{oh}(i,k) f'(n_p(k))\sum_{n=1}^{N+1} x_p(n) g_m(k,n)\sum_{v=1}^{N_h} w_{oh}(i,v) f'(n_p(v))\sum_{j=1}^{N+1} x_p(j) g_u(v,j) \tag{3.22}$$

by rearranging the terms we get,

$$= \sum_{k=1}^{N_h}\sum_{v=1}^{N_h}\sum_{n=1}^{N+1}\sum_{j=1}^{N+1} g_m(k,n) g_u(v,j) \frac{2}{N_v}\sum_{p=1}^{N_v} x_p(n) x_p(j) f'(n_p(v)) f'(n_p(k))\sum_{i=1}^{M} w_{oh}(i,v) w_{oh}(i,k) \tag{3.23}$$

$$= \sum_{k=1}^{N_h}\sum_{v=1}^{N_h}\sum_{n=1}^{N+1}\sum_{j=1}^{N+1} g_m(k,n) g_u(v,j) \frac{\partial^2 E}{\partial w(k,n)\partial w(v,j)} \tag{3.24}$$

Let $\mathbf{d}_m$ denote the mth column of the N_h by (N+1) input weight change matrix such as **G**. Let $h_{mv}(j,u)$ denote the element of the input weight Hessian $\mathbf{H}_R$ that relates input weights w(j,m) and w(u,v). Similarly, $h_{mv}(j,u)$ is an element of the N_h by N_h matrix $\mathbf{H}_R$. Then

$$h_{og}(m,v) = \mathbf{d}_m{}^T \mathbf{H}_R \mathbf{d}_v \tag{3.25}$$

The vector **r** is found as the solution to the equation,

$$\mathbf{H}_{og} \cdot \mathbf{r} = \mathbf{g} \tag{3.26}$$

The above equation is solved using OLS. Note that elements of **r** can be positive or negative.

3.2 OOG Steps

In this section we will describe a series a steps taken in each iteration of OOG Algorithm.

In each iteration of the training Algorithm the following steps are followed.

1. Solve linear equations for all output weights.
2. Save all the Weights information
3. Calculate Error E and compare it with the error in previous iteration.
4. If error in any iteration is greater than the previous, we read back the weights from the previous iteration, calculate the optimal learning factor and update the input weights using OLF as in equation (3.13). OLF can be calculated from OOG algorithm using the equation (4.7)
5. Calculate $g_i(k,n)$ using equation (3.11)
6. Calculate the vector **g** using the equation (3.17).
7. Calculate Hessian $\mathbf{H}_{og}$ using equation (3.21)
8. Calculate output gain co-efficient vector **r** using the equation (3.26). This can be solved by OLS algorithm.
9. Update the input weights as

$$\mathbf{W} \longleftarrow \mathbf{W} + \sum_{i=1}^{M} r(i) \cdot \mathbf{G}_i \tag{3.27}$$

10. Go back to step 1 until all iterations are complete.

3.3 OOG-HWO Algorithm

In this section, we introduce the hidden weight optimization technique.

Output weight optimization-hidden weight optimization (OWO-HWO) [25] training is very similar to OWO-BP. The weights connected to the outputs are adapted using OWO mentioned in II-B. However, unlike BP, HWO minimizes the objective function [33]

$$E_{\delta}(j) = \sum_{p=1}^{Nv} \left[\delta_p(j) - \sum_{n=1}^{N+1} g_{hwo}(k,n) x_p(n) \right]^2 \tag{3.28}$$

for $0 \le I \le Nh$, by solving for linear equations of the form

$$\sum_{n=1}^{N+1} g_{hwo}(k,n) r(n,m) = -\frac{\partial E}{\partial w(j,m)} \tag{3.29}$$

In matrix notation,

$$\mathbf{G}_{hwo} \cdot \mathbf{R} = \mathbf{G} \tag{3.30}$$

where R is the input auto-correlation matrix and $\mathbf{G}_{hwo}$ is the HWO weight change matrix. In other words, $\mathbf{G}_{hwo}$ replaces the matrix G in (3.16), (3.19). The linear equations in (3.29) can be solved for $\mathbf{G}_{hwo}$ using orthogonal least squares (OLS) or matrix inversion using the singular value decomposition (SVD). It can be shown that HWO is equivalent to applying a whitening transform [32] to the training data and then performing BP.

3.3.1 Convergence Proof for HWO Algorithm

The equation 32 can be rewritten as

$$\mathbf{G}_{hwo} = \mathbf{G} \cdot \mathbf{R}^{-1} \tag{3.31}$$

Expressing the singular value decomposition (SVD) of the auto-correlation matrix as

$$\mathbf{R} = \mathrm{U} \sum \mathrm{U}^{\mathrm{T}} \tag{3.32}$$

$\mathbf{R}^{-1}$ becomes,

$$\mathbf{R}^{-1} = \mathrm{U} \sum^{-1} \mathrm{U}^{\mathrm{T}} \tag{3.33}$$

Equation 32 now becomes,

$$\mathrm{G}_{\mathrm{hwo}} = \mathrm{G} \cdot \mathrm{A}^{\mathrm{T}} \cdot \mathrm{A} \tag{3.34}$$

where,

$$\mathrm{A} = \sum^{-1/2} \mathrm{U}^{\mathrm{T}} \tag{3.35}$$

Comparing (3.34) with equation (2.25) it is clear that performing OWO-HWO is equivalent to performing OWO-BP on transformed data. Since BP with optimal learning factor (OLF) converges, it is clear that HWO with an OLF converges as well.

CHAPTER 4

ANALYSES

In this chapter we will discuss the relationship between OOG and OLF, the effects of linearly dependent outputs on gradient **G** and Hessian **H**

4.1 Relationship of OOG to OWO-BP with OLF

The optimal learning factor can be calculated from gradient and Hessian from OOG algorithm. This is clearly explained in this section.

In order to calculate the OLF, we need the first derivative of E with respect to z that is $\partial E/\partial z$ and also the second derivative of E with respect to z that is $\partial^2 E/\partial z^2$

The first derivative can be calculated as,

$$\frac{\partial E}{\partial z} = \frac{-2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] \frac{\partial y_p(i)}{\partial z}, \tag{4.1}$$

This gives,

$$\frac{\partial y_p(i)}{\partial z} = \sum_{m=1}^{M} \frac{\partial y_p(i)}{\partial r(m)}, \tag{4.2}$$

(4.2) can be calculated easily using the equation (3.19). After calculating (4.2) we can plug in the values in (4.1) to get $\frac{\partial E}{\partial z}$

The second derivative of E with respect to z can be written as,

$$\frac{\partial^2 E}{\partial z^2} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial z}\frac{\partial y_p(i)}{\partial z} \tag{4.3}$$

This can be further written as below,

$$\frac{\partial^2 E}{\partial z^2} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\sum_{n=1}^{M}\sum_{m=1}^{M}\frac{\partial y_p(i)}{\partial r(n)}\frac{\partial y_p(i)}{\partial r(m)} \tag{4.4}$$

$$\frac{\partial^2 E}{\partial z^2} = \sum_{n=1}^{M}\sum_{m=1}^{M}\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial r(n)}\frac{\partial y_p(i)}{\partial r(m)} \tag{4.5}$$

This leads to,

$$\frac{\partial^2 E}{\partial z^2} = \sum_{n=1}^{M}\sum_{m=1}^{M} h_{og}(m,n) \tag{4.6}$$

The second derivative of E with respect to z is nothing but the sum of all the elements of the Hessian H. So now the OLF can be easily calculated as

$$OLF = -\frac{(\frac{\partial E}{\partial z})}{(\frac{\partial^2 E}{\partial z^2})} \tag{4.7}$$

4.2 Effects of linearly dependent outputs on G′

The negative gradient of E' is given by

$$g'(k,n) = \sum_{i=1}^{M} r(i)\cdot g_i(k,n) \tag{4.8}$$

Where,

$$g_i(k,n) = \frac{1}{N_v}\sum_{p=1}^{N_v}\delta_p(i,k)x_p(n) \tag{4.9}$$

and,

$$\delta_p(i,k) = f'(n_p(k)) \cdot W_{oh}(i,k) \cdot \delta_{po}(i) \tag{4.10}$$

The gradient can now be expressed as,

$$g'(k,n) = \frac{1}{N_v} \sum_{i=1}^{M} r(i) \cdot W_{oh}(i,k) \sum_{p=1}^{N_v} f'(n_p(k)) \cdot \delta_{po}(i) x_p(n) \tag{4.11}$$

Case 1: <u>With one extra linearly dependent desired output</u>

Let us consider a case with one extra output (output M+1) which is linearly dependent on the other M outputs.

$$\mathbf{t}_p(M+1) = \sum_{i=1}^{M} r(i) \mathbf{t}_p(i) \tag{4.12}$$

Then the gradient matrix is given by

$$g''(k,n) = \frac{1}{N_v} \sum_{i=1}^{M+1} r(i) \cdot W_{oh}(i,k) \sum_{p=1}^{N_v} f'(n_p(k)) \cdot \delta_{po}(i) x_p(n) \tag{4.13}$$

$$= g'(k,n) + \{\frac{1}{N_v} r(M+1) \cdot W_{oh}(M+1,k) \sum_{p=1}^{N_v} f'(n_p(k)) \cdot \delta_{po}(M+1) x_p(n)\} \tag{4.14}$$

We can clearly see that the new gradient is linear sum of previous gradient $g'(k,n)$ and some other terms. Ideally we want r(M+1) to be zero so that the gradients are unchanged even when there is an extra dependent output.

Case 2: <u>With L extra linearly dependent outputs</u>

Let us consider a case with L extra output (output M+L) which are linearly dependent on the other M outputs. The gradient can be expressed as,

$$g''(k,n) = g'(k,n) + \frac{1}{N_v}\sum_{i=M+1}^{M+L} r(i)\cdot W_{oh}(i,k)\sum_{p=1}^{N_v} f'(n_p(k))\cdot\delta_{po}(i)x_p(n) \tag{4.15}$$

We can clearly see that the new gradient is sum of previous gradient $g'(k,n)$ and some other distortion terms.

4.3 Effects of linearly dependent outputs on Hessian $\mathbf{H}_{og}$

The Hessian $\mathbf{H}_{og}$ is given by,

$$h_{og}(m,u) \equiv \frac{\partial^2 E}{\partial r(m)\partial r(u)} \tag{4.16}$$

$$= \frac{\partial^2 E}{\partial r(m)\partial r(u)} \tag{4.17}$$

$$= \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial r(m)}\frac{\partial y_p(i)}{\partial r(u)} \tag{4.18}$$

$$= \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M} v_p(i,m)\cdot v_p(i,u) \tag{4.19}$$

Where $v_p(i,m)$ is defined as,

$$v_p(i,m) = \frac{\partial y_p(i)}{\partial r(m)} = \sum_{k=1}^{N_h} w_{oh}(i,k)\, f'(n_p(k))\sum_{n=1}^{N+1} g_m(k,n)x_p(n) \tag{4.20}$$

Let us consider a case where we have one extra dependent output.

Case 1: With one extra linearly dependent output

The number of outputs will now be M+1. Hence the index i should be from 1 to M+1.

The new Hessian $\mathbf{H}'_{og}$ can be written as,

$$h'_{og}(m,u) \equiv \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M+1} v_p(i,m)\cdot v_p(i,u) \tag{4.21}$$

We can now separate the term containing only the last output ,

$$h'_{og}(m,u) \equiv \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M} v_p(i,m)\cdot v_p(i,u) + \{\frac{2}{N_v}\sum_{p=1}^{N_v} v_p(M+1,m)\cdot v_p(M+1,u)\} \tag{4.22}$$

This leads to,

$$h'_{og}(m,u) \equiv h_{og}(m,u) + \{\frac{2}{N_v}\sum_{p=1}^{N_v} v_p(M+1,m)\cdot v_p(M+1,u)\} \tag{4.23}$$

Where ,

$$v_p(M+1,m) = \sum_{k=1}^{N_h} w_{oh}(M+1,k)\, f'(n_p(k)) \sum_{n=1}^{N+1} g_m(k,n) x_p(n) \tag{4.24}$$

We can clearly see that the new Hessian is the sum of the previous Hessian $h_{og}(m,u)$ and a sum of products of the v functions.

Case 2: <u>With L extra linearly dependent outputs</u>

The number of outputs will now be M+L. Hence the equation for the new Hessian will be as shown below,

$$h'_{og}(m,u) \equiv h_{og}(m,u) + \{\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=M+1}^{M+L} v(i,m)\cdot v(i,u)\} \tag{4.25}$$

We can clearly see that the new Hessian is the sum of the previous Hessian $h_{og}(m,u)$ and additional noise/distortion terms.

4.4 Computational Burden

In this section, we describe the computational burden for using the training algorithms compared in this paper. All the algorithms were implemented in Microsoft Visual C++ compiler version 6.0.

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 M_{ols}, which is given by

$$M_{ols} = N_u(N_u + 1)\left[M + \frac{1}{6}N_u(2N_u + 1) + \frac{3}{2}\right] \tag{4.26}$$

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

$$M_{bp} = N_v\left[MN_u + 2N_h(N + 1) + M(N + 6N_h + 4)\right] + N_w \tag{4.27}$$

$$M_{owo-bp} = N_v\left[2N_h(N + 2) + M(N_u + 1) + \frac{Nu(N + 1)}{2} + M(N + 6N_h + 4)\right] + M_{ols} + N_h(N + 1) \tag{4.28}$$

$$M_{oog} = M_{owo-bp} + N_v\left[M\left((N+1)(3N_h+1) + N_h(N+1)(1+2M) + M^2 + 2M + 1\right)\right] + M_{ols} \tag{4.27}$$

$$M_{lm} = M_{bp} + N_v\left[MN_u(N_u + 3N_h(N+1)) + 4N_h{}^2(N+1)^2\right] + N_w{}^2 + N_w{}^3 \tag{4.28}$$

Note that M_{oog} consists of M_{owo-bp} plus the required multiplies for calculating optimal output gains. Similarly, M_{lm} consists of M_{bp} plus the required multiplies for calculating and inverting the Hessian matrix..

CHAPTER 5

NUMERICAL RESULTS

Here we compare the performance of OOG and OOG-HWO to those of BP-OLF, LM, and conjugate gradient (CG), where the OLF was used in the latter three algorithms. In CG and LM, all weights are varied in each iteration. In OOG, OOG-HWO, we first solve linear equations for the output weights and subsequently update the input weights.

For a given network, we obtain the training error and the number of multiplies required for each training iteration. We also obtain the validation error for a fully trained network. This information is used to subsequently generate the plots and compare performances.

Table 5.1 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
Oh7.tra	20	3	15000
Concrete Data Set	8	1	1030

Table I lists the data sets used for comparison and generating the plots. We use the *k-fold* validation procedure to obtain the average training and validation errors. Given a data set, we split the set into k non-overlapping parts of equal size, and use (k − 1) parts for training and the remaining one part for validation. The procedure is repeated till we have exhausted all k combinations (k = 10 for our simulations). In all our simulations we have 4000 iterations for the first order algorithms BP-OLF and CG, 4000 iterations for OOG and OOG-HWO algorithms and for LM we have 300 iterations. All the data sets used for simulation are publicly available. In all data sets, the inputs have been normalized to be zero-mean and unit variance.

We have chosen different number of hidden units for different data files for our simulations. Given a data file, number of inputs, number of outputs we first determine the MLP network sizing using the NU-MAP 7.1 software that is available in Image Processing and Neural Networks Lab repository [19]. MLP Sizing utility gives the number of hidden units for which the MLP can perform the best. The description and the simulation results for each of the data files and are explained in this section.

5.1 Twod.tra Data Set

This data file is available on the Image Processing and Neural Networks Lab repository [19]. 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.

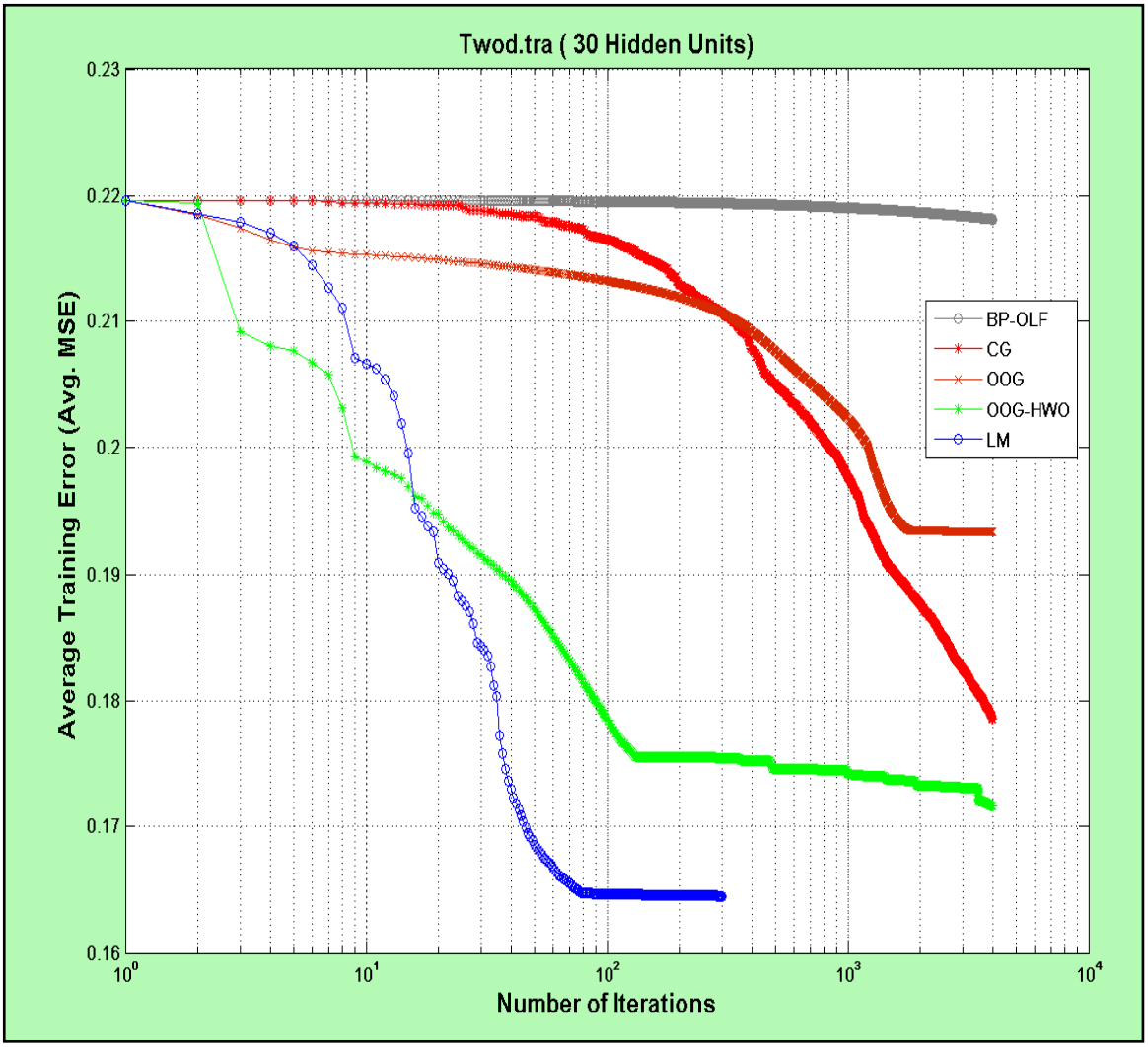

Fig 5.1 Twod.tra data set: Average error vs. iterations

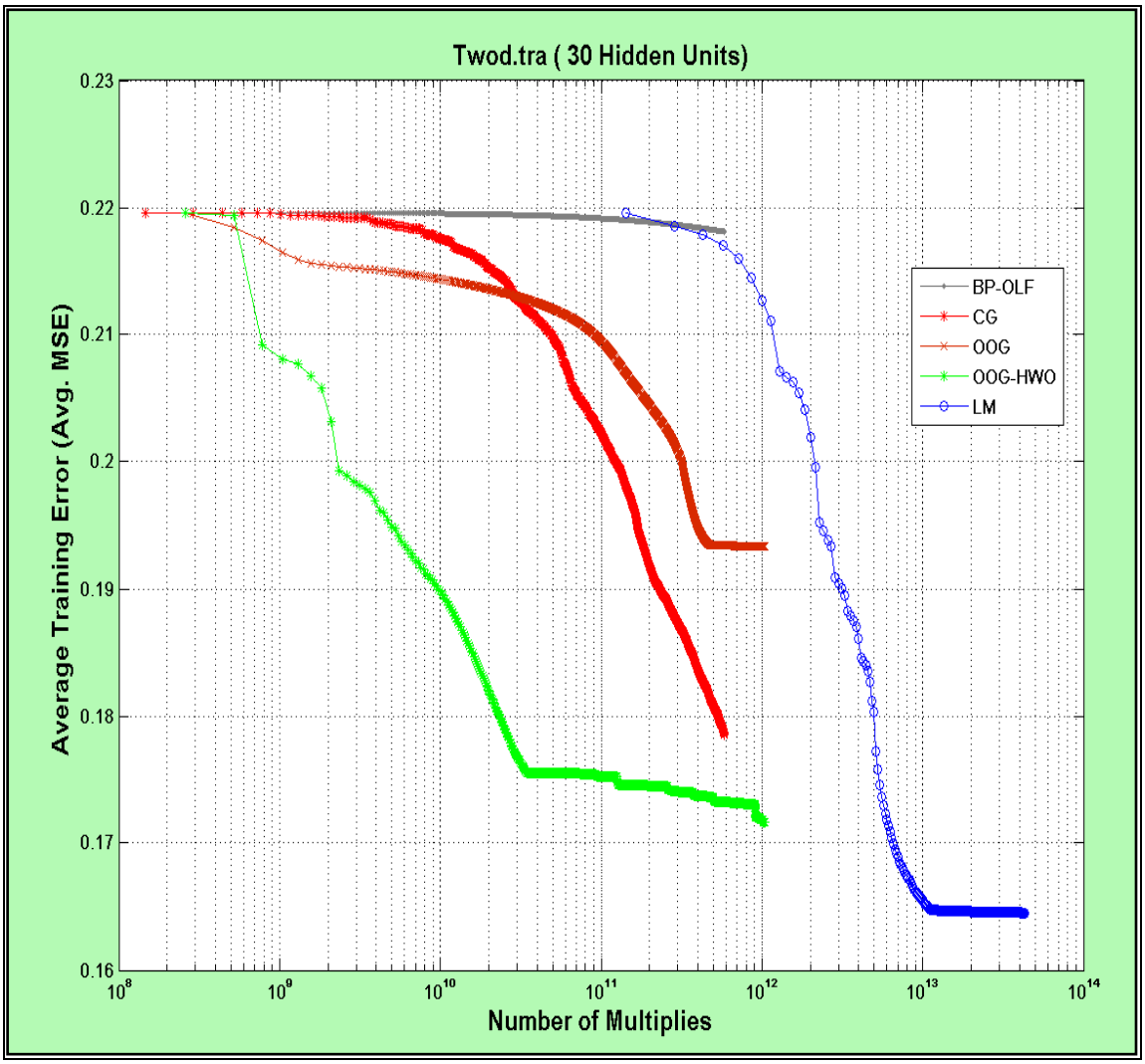

Fig 5.2 Twod.tra data set: Average error vs. Multiplies per iteration

For this data file, we trained an MLP having 30 hidden units. In Fig. 5.1, 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 Fig. 5.2, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a log10 scale).

From Fig. 5.1, OOG and OOG-HWO are better than the previous version of BP-OLF and CG in terms of the overall training error. LM has better overall training error, however, the performance comes with a significantly higher computational demand, as shown in Fig 5.2

5.2 Single2.tra Data Set

This data file is available on the Image Processing and Neural Networks Lab repository [19]. 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.

For this data file, we trained an MLP having 20 hidden units. From Fig. 5.3, the average training MSE from 10-fold validation for both OOG and OOG-HWO are better than all other algorithms being compared except for LM.

Fig. 5.4 shows the computational cost of achieving this performance. The proposed algorithms consume slightly more computation compared to BP-OLF and CG. However, all algorithms utilize about two orders of magnitude fewer computations than LM.

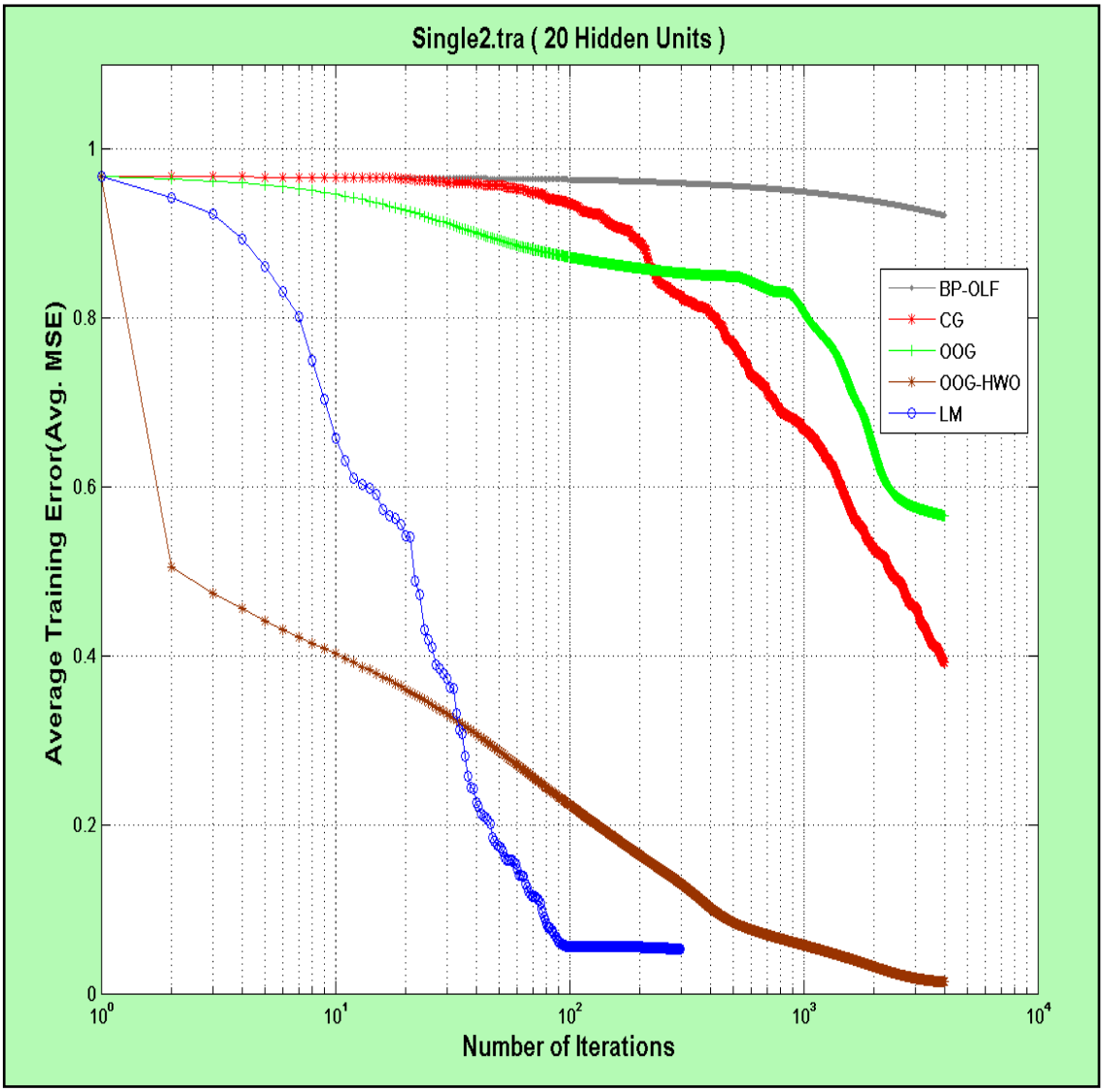

Fig 5.3 Single2.tra data set: Average error vs. iterations

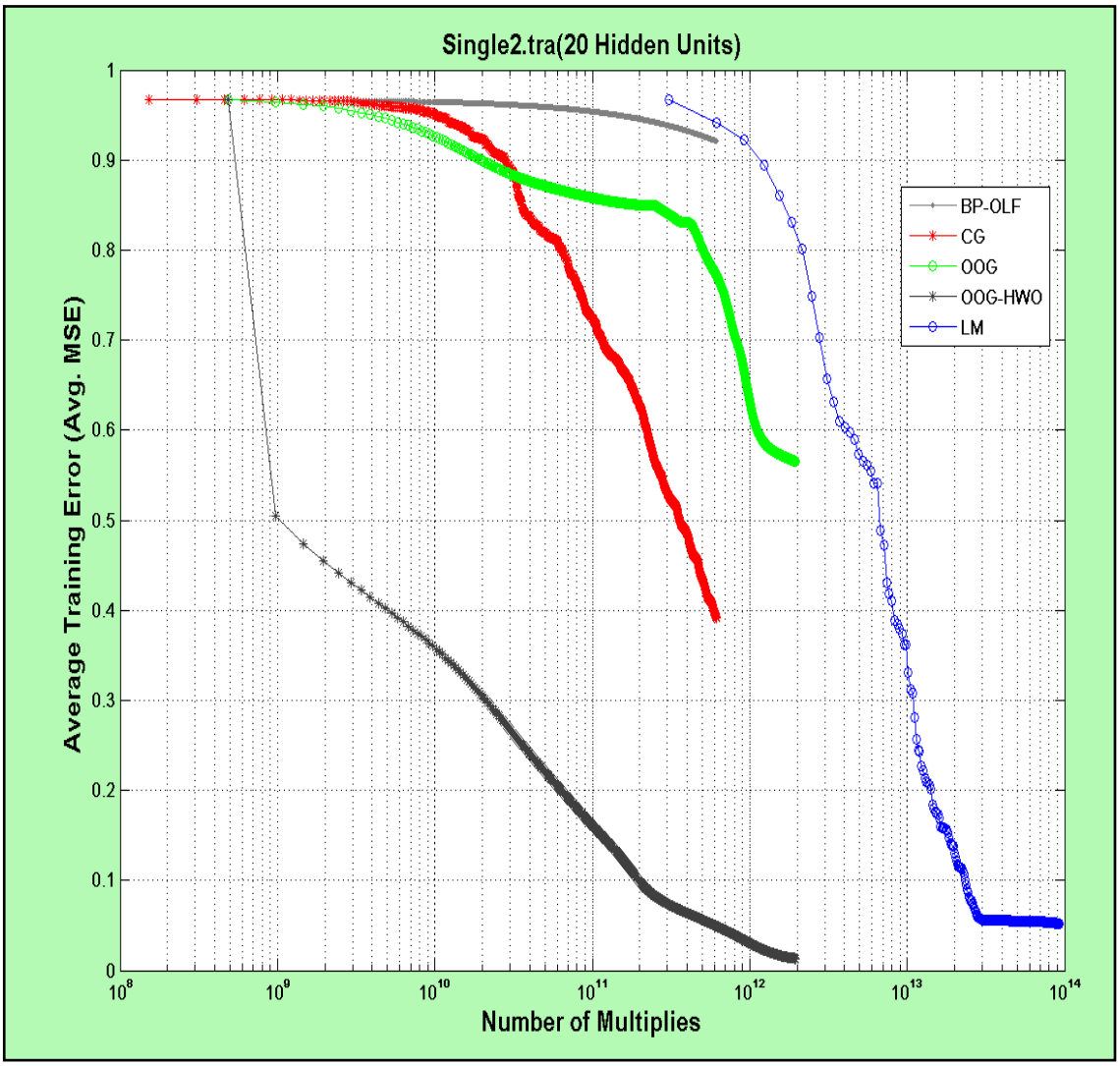

Fig 5.4 Single2.tra data set: Average error vs. Multiplies per iterations

5.3 OH7.tra Data Set

This data file is available on the Image Processing and Neural Networks Lab repository [19]. This data set is given in Oh, Y., K. Sarabandi, and F.T. Ulaby, "An Empirical Model and an Inversion Technique for Radar Scattering from Bare Soil Surfaces," in IEEE Trans. on Geoscience and Remote Sensing, pp. 370-381, 1992. The

training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g / cubic cm.

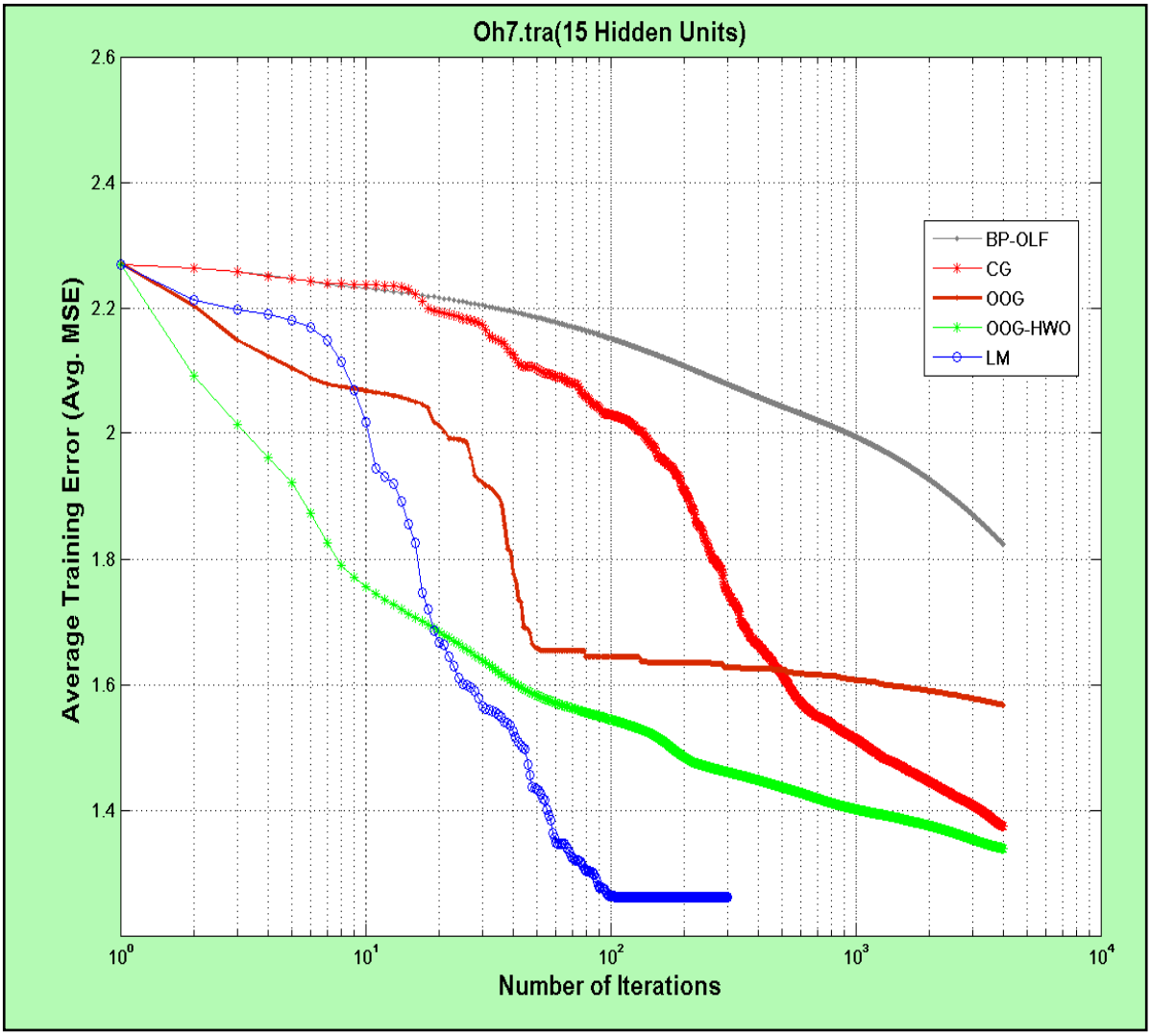

Fig. 5.5 OH7.tra data set: average error vs. iterations

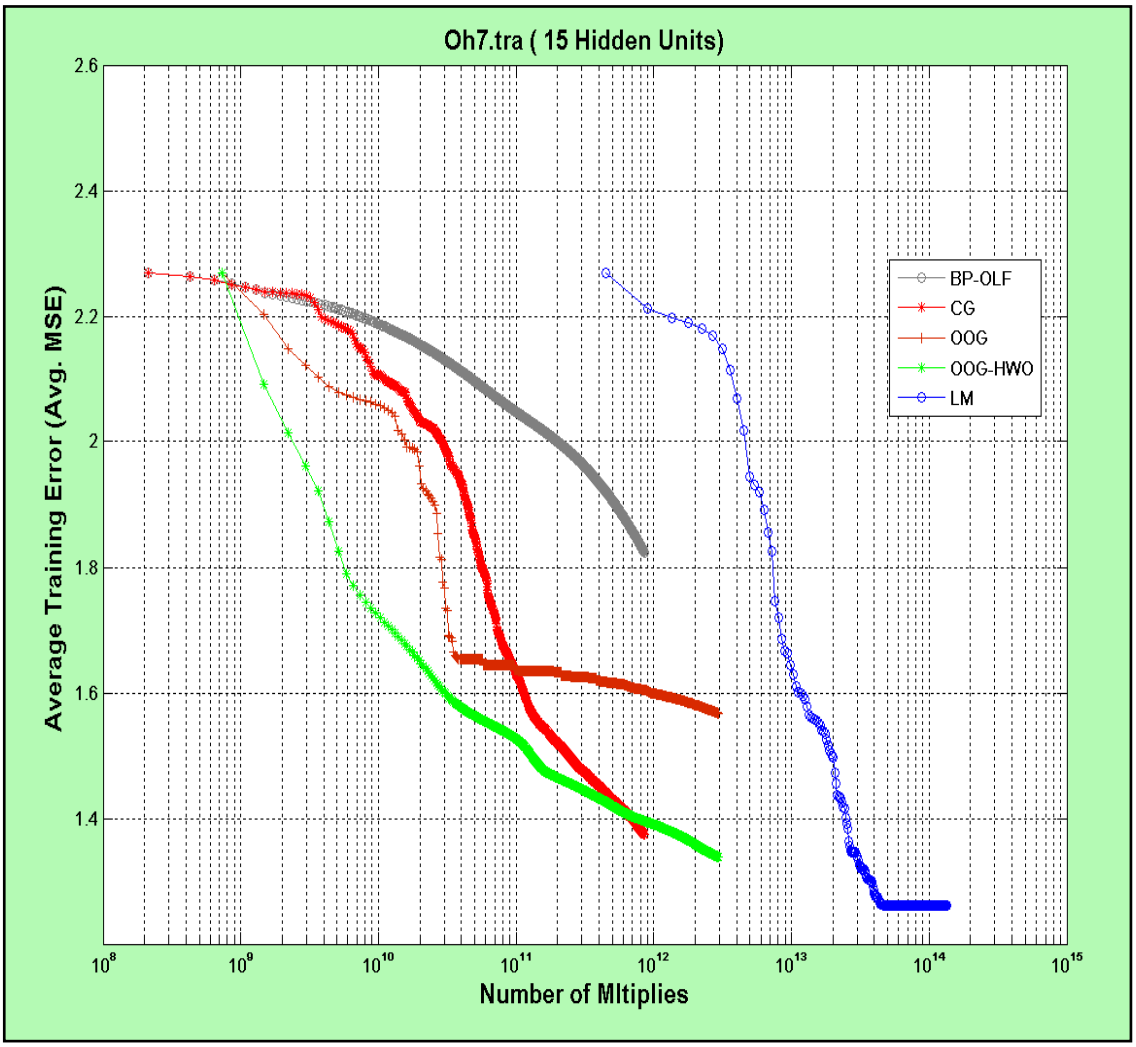

Fig. 5.6 OH7.tra data set: average error vs. Multiplies per iterations

For this data file, we trained an MLP having 15 hidden units. From Fig. 5.5, we see a similar trend. The proposed OOG-HWO, perform better than all other algorithms considered for comparison except for LM. The required multiplies for the OOG and OOG-HWO are not significantly more compared with CG, BP-OLF. Again, OOG and OOG-HWO perform better. This is evident from Fig. 5.6

5.4 Concrete Data Set

This data file is available on the UCI Machine Learning Repository [21]. 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. For this data file, we trained an MLP having 15 hidden units. For this data set, the LM algorithm has a better overall training error, however the proposed algorithms OOG and OOG-HWO, present a good balance between performance and computational cost. These are evident from Fig 5.7 and Fig 5.8

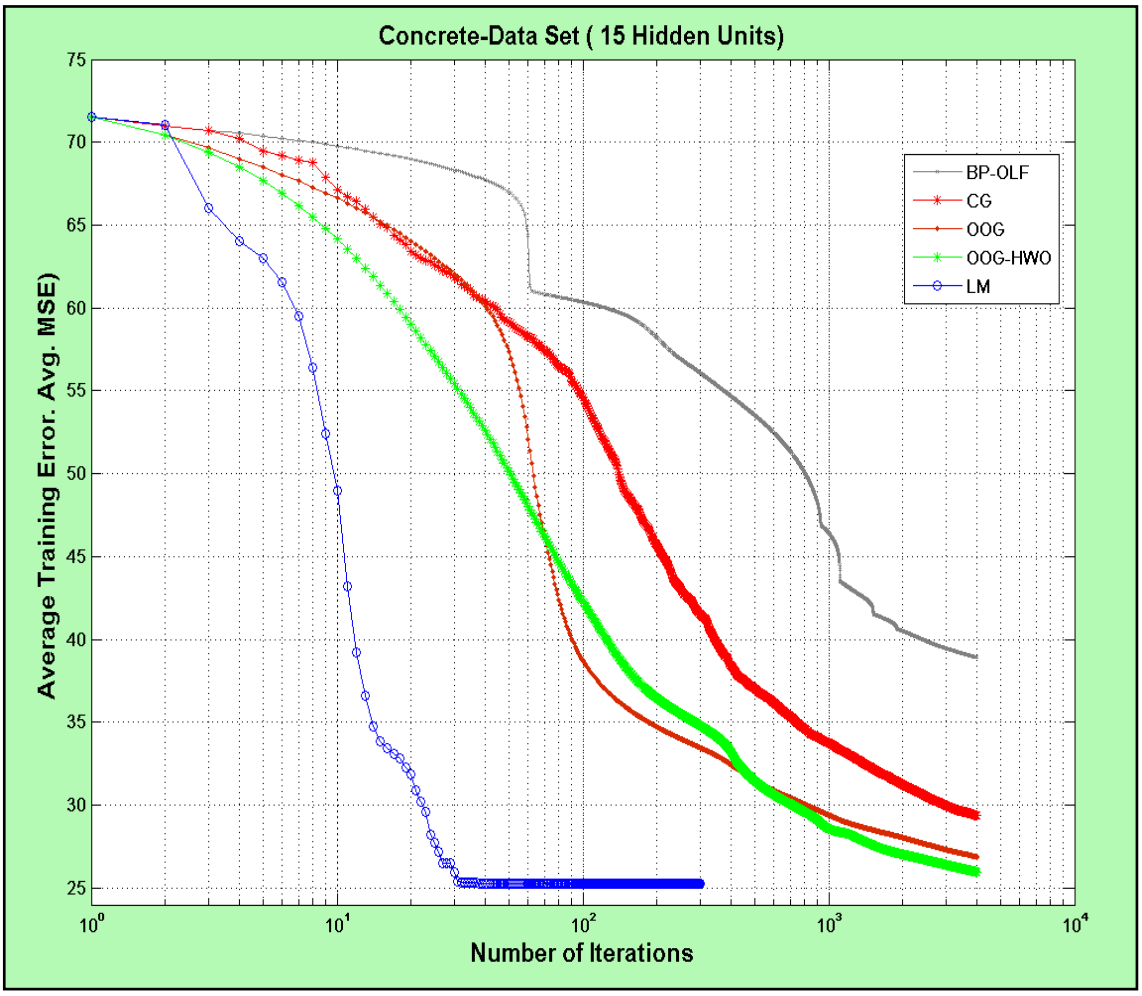

Fig. 5.7 . Concrete data set: average error vs. iterations

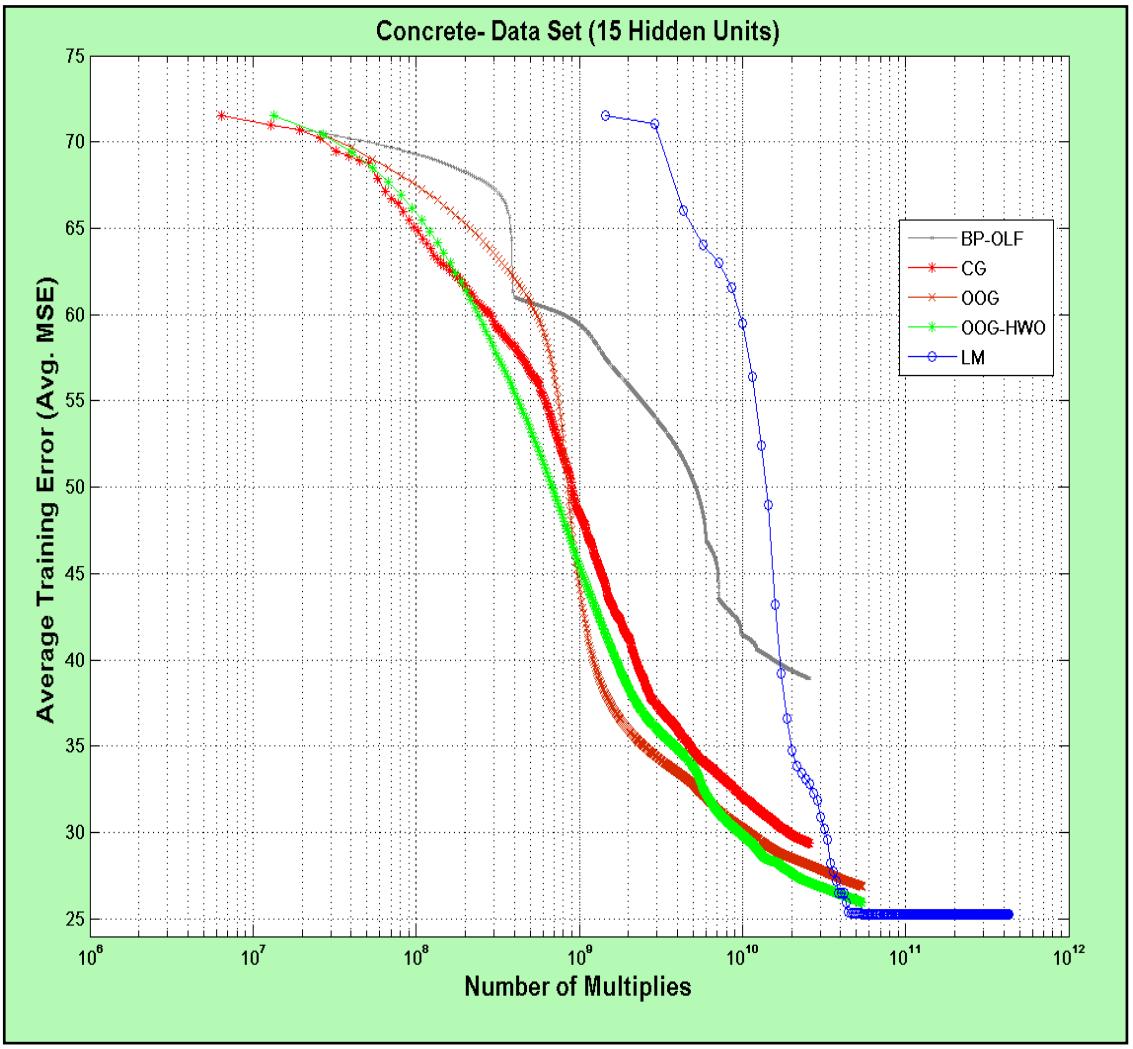

Fig. 5.8. Concrete data set: average error vs. Multiplies per iteration

Table II compares the average training and validation errors of the proposed OOG algorithms with CG, BP-OLF and LM on different data sets. For each data set, the training and validation errors again come from 10-fold validation.

Table 5.2 Average 10-Fold Training and Validation Error

Data Set		BP-OLF	CG	OOG	OOG-HWO	LM
Twod.tra	E_{trn}	0.218699	0.190905	0.1977993	0.1847215	0.1689533
	E_{val}	0.253583	0.214977	0.248837	0.2033308	0.172326
Single2.tra	E_{trn}	0.939597	0.5724019	0.6859521	0.0504069	0.1313133
	E_{val}	1.11143	0.974455	1.045655	0.0630476	0.145549
OH7.tra	E_{trn}	1.940934	1.49723053	1.5983069	1.3888627	1.3480560
	E_{val}	2.295675	2.074047	2.069648	1.985654	1.571530
Concrete	E_{trn}	43.9238	33.28654328	29.361767	28.694411	26.779379
	E_{val}	68.634015	67.542675	43.782042	42.735953	40.832865

From the plots and the table, we see that the two algorithms OOG and OOG-HWO are effective in further reducing the training and validation errors. OOG-HWO seems to show consistently better performance than the other algorithms.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

We have derived second order methods for simultaneously optimizing output gains and the learning factor. These methods have been successfully demonstrated on four data sets. Results show that these algorithms perform much better than two common first order algorithms with comparable complexity, namely CG and BP-OLF. They come close to LM in terms of the training error, but with orders of magnitude less computation. This is evident in all of the plots of training error versus the required number of multiplies and also from the expressions for the numbers of multiplies. Although LM works very well in practice, it has a high computational burden and is sub-optimal in the way it handles the 'scaling' factor, OOG and OOG-HWO on the other hand use a Newton type update combined with the optimal learning factor, leaving little room for heuristics.

6.2 Future Work

Much work remains to be done. We hope to extend our approach to additional network parameters, yielding fast second order methods that rival the performance of LM, but with greatly reduced complexity. OOG can also be extended for forecasting applications where the desired outputs are highly correlated. OOG can be used to extract more efficient input change matrices.

REFERENCES

[1] 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.

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

[3] G. Cybenko, "Approximations by Superpositions of a Sigmoidal Function," *Math. Contrl., Signals, Syst.*, Vol. 2, pp. 303-314, 1989.

[4] Dennis W. Ruck et al., "The Mulitlayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," IEEE Trans. on Neural Networks, Vol. 1, No. 4, 1990.

[5] M.T. Manry, S.J. Apollo, and Q. Yu, "Minimum Mean Square Estimation and Neural Networks," *Neurocomputing*, vol. 13, September 1996, pp. 59-74.

[6] Matan C J., Burges C., Le Cun Y. and Denker J S. Multi-digit recognition using a space displacement neural network. In: *Advances in Neural Information Processing Systems*, vol. 4, pp. 488-495, 1991

[7] S-B. Cho, “Neural-Network Classifiers for Recognizing Totally Unconstrained Handwritten Numerals”, *IEEE Trans. on Neural Networks*, vol. 8, pp. 43-53, 1997.

[8] K. Liu, S. Subbarayan, R.R.Shoults, M.T.Manry, C.Kwan, F.L.Lewis, and J.Naccarino, "Comparison of Very Short-Term Load Forecasting Techniques," *IEEE Transactions on Power Systems*, vol.11, no.2, May 1996, pp. 877-882.

[9] M.T. Manry, H. Chandrasekaran, and C-H Hsieh, "Signal Processing Applications of the Multilayer Perceptron," book chapter in *Handbook on Neural Network Signal Processing*, edited by Yu Hen Hu and Jenq- Nenq Hwang, CRC Press, 2001.

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

[11] Lipo Wang and Xiuju Fu, *Data Mining With Computational Intelligence*, Springer-Verlag, 2005.

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

[13] F. J. Maldonado and M.T. Manry, "Optimal Pruning of Feed Forward Neural Networks Using the Schmidt Procedure", *Conference Record of the Thirty Sixth Annual Asilomar Conference on Signals, Systems, and Computers*., November 2002, pp. 1024-1028

[14]QR_decomposition",*Wikipedia*, http://en.wikipedia.org/wiki/QR_decomposition

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

[16] Y. LeCun, "Efficient Learning and Second-Order Methods, " A Tutorial at NIPS 93, Denver 1993.

[17] M.T. Manry, H. Chandrasekaran, and C-H Hsieh, "Signal Processing Applications of the Multilayer Perceptron," book chapter in *Handbook on Neural Network Signal Processing*, edited by Yu Hen Hu and Jenq- Nenq Hwang, CRC Press, 2001.

[18] A. K. Fung, Z. Li, and K. S. Chen, "Back scattering from a Randomly Rough Dielectric Surface," *IEEE Trans. Geo. and Remote Sensing*, Vol. 30, No. 2, March 1992.

[19] A. Mennon, K. Mehrotra, C. K. Mohan, and S. Ranka, “Characterization of a class of sigmoid functions with applications to neural networks,” Neural Networks, vol. 9, pp. 819{835, 1996.

[20] US Census Bureau [http://www.census.gov] (under Lookup Access [http://www.census.gov/cdrom/lookup]: Summary Tape File 1)

[21] Source: http://www.stls.frb.org/fred/index.html

[22] Source: http://www-ee.uta.edu/eeweb/ip/training_data_files.htm

[23] Source: http://www.cs.toronto.edu/~delve/data/datasets.html

[24] Source: http://archive.ics.uci.edu/ml/datasets.html

[25] F. J. Maldonado, M. T. Manry, Tae-Hoon Kim, "Finding optimal neural network basis function subsets using the Schmidt procedure", *Proceedings of the International Joint Conference on Neural Networks*, 20-24 July 2003, vol. 1, pp. 444 - 449.

[26] 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.

[27] S. S. Malalur, M. T. Manry, "Feed-forward Network Training Using Optimal Input Gains,"*Proceedings of the International Joint Conference on Neural Networks*, Georgia, Atlanta, USA, pp. 1953-1960, June 14-19, 2009

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

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

[30] T. Vogl, J. Mangis, J. Rigler, W. Zink, D. Alkon, "Accelerating the convergence of the back-propagation method," *Biological Cybernetics* 59, pp.257263, 1988.

[31] R. Battiti, "Accelerated backpropagation learning: Two optimization methods,", *Complex Systems* 3, pp. 331342, 1989.

[32] S. Raudys, *Statistical and Neural Classifiers: An Integrated Approach to Design*, Springer-Verlag, 2001.

[33] R.S. Scalero, N. Tepedelenlioglu, “A fast new algorithm for training feedforward neural networks,” *IEEE Transactions on Signal Processing*, Vol. 40, Issue 1 pp. 202-210, 1997.

[34] “Levenberg-Marquardt Algorithm”, *Wikipedia*, http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

[35] “Newton’s Method in Optimization”, *Wikipedia*, http://en.wikipedia.org/wiki/Newton%27s_method_in_optimization

[36] R. Battiti, “First- and Second-Order Methods for Learning: Between Steepest Descent and Newton’s Method,” *Neural Computation* vol. 4, pp. 141-166, 1992.

[37]] M.T. Manry, S.J. Apollo, L.S. Allen, W.D. Lyle, W. Gong, M.S. Dawson, and A.K. Fung, "Fast Training of Neural Networks for Remote Sensing," *Remote Sensing Reviews*, vol. 9, pp. 77-96, 1994

[38] W.H. Delashmit and M.T. Manry, “A Neural Network Growing Algorithm that Ensures Monotonically Non Increasing Error”, *Advances in Neural Networks*, vol.14, August 2007, pp.280-284.

[39] Fahlman, S. E. and C. Lebiere (1990) "The Cascade-Correlation Learning Architecture" in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky (ed.), Morgan-Kaufmann, 1990

[40] P. L. Narasimha, W.H. Delashmit, M.T. Manry, Jiang Li, and F. Maldonado, "An Integrated Growing-Pruning Method for Feedforward Network Training," *NeuroComp.*, vol. 71, Spring 2008, pp. 2831-2847

BIOGRAPHICAL INFORMATION

Babu Hemanth Kumar was born in India in 1984. He did his Bachelor of Technology in Electronics and Communication Engineering from M.S.Ramaiah Institute of Technology Bangalore in May 2006. He obtained his Master of Science degree from the University of Texas at Arlington in December 2010. He worked for Intel as an intern in the spring and summer of 2010, where he worked in developing test application for media drivers. He will be joining Intel as a full-time employee in January 2011. His current research interests includes neural networks, image processing and multimedia processing. He has served as a Graduate Teaching Assistant for the courses Digital signal Processing / Statistical Signal Processing in the Electrical Engineering department of University of Texas at Arlington (2010)