Two-Stage Second Order Training in Feedforward Neural Networks

Melvin D. Robinson and **Michael T. Manry**
Department of Electrical Engineering
University of Texas at Arlington

Abstract

In this paper, we develop and demonstrate a new 2^{nd} order two-stage algorithm called OWO-Newton. First, two-stage algorithms are motivated and the Gauss Newton input weight Hessian matrix is developed. Block coordinate descent is used to apply Newton's algorithm alternately to the input and output weights. Its performance is comparable to Levenberg-Marquardt and it has the advantage of reduced computational complexity. The algorithm is shown to have a form of affine invariance.

1 Introduction

The multilayer perceptron (MLP) is the most widely used type of neural network(Shepherd 1997). Training a MLP involves solving a non-convex optimization problem to calculate the network's weights. The single hidden layer MLP is the focus of our work because any continuous function can be approximated to arbitrary precision(Cybenko 1992).

Though one-step methods such as backpropagation (Werbos 1974), conjugate gradient (Charalambous 1992) and Levenberg-Marquardt (Hagan and Menhaj 1994) are popular, a MLP can sometimes be trained effectively using two step methods in which not all weights are updated simultaneously. Example two-step methods include cascade correlation (Fahlman and Lebiere 1990), OWO-HWO (Chen, Manry, and Chandrasekaran 1999) and layer by layer (Li, Li, and Wang 2011).

Second order methods have better convergence than first order methods; however, we must use care when employing them because the full network Hessian is inherently rank deficient (Smagt and Hirzinger 1998) and this can create problems in training (LeCun et al. 1998). Two-step methods allow us to mitigate this problem.

In this work we combine two-step feedforward training with 2^{nd}-order optimization methods. This allows us to achieve fast convergence and reduced computational complexity over one-step 2^{nd}-order methods.

In Section 2 we briefly review MLP architecture and our notation. Section 3 reviews the Output Weight Optimization and Newton algorithms. In Section 4 we motivate two-step second-order training methods. After discussing problems with one-step second order methods, we introduce, in Section 5, the OWO-Newton algorithm. Section 6 develops some theoretical properties of OWO-Newton. Sections 7 and 8 give some experimental results and conclusions.

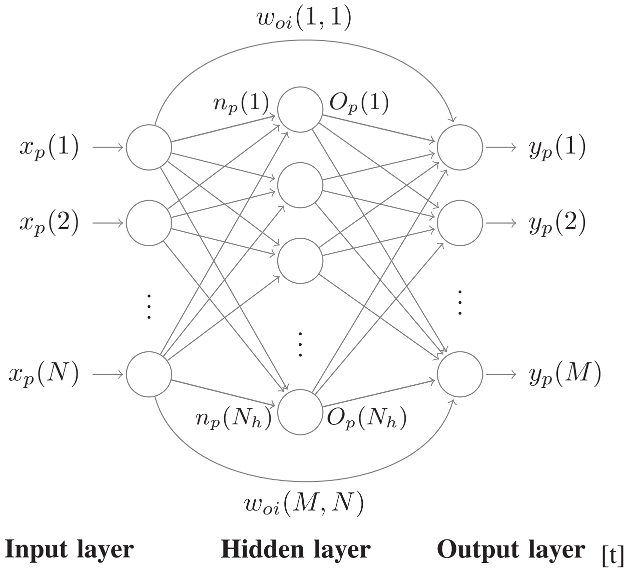

Figure 1: Illustration of a Multilayer Perceptron

2 Architecture and Notation

Figure 1 illustrates the structure of a single hidden layer MLP having an input layer, a hidden layer and an output layer. We denote the number of hidden units by N_h and the number of outputs by M. Here, the input vectors are $\mathbf{x}_p \in \mathbb{R}^N$, and the desired output vectors are $\mathbf{t}_p \in \mathbb{R}^M$. The training data $\{\mathbf{x}_p, \mathbf{t}_p\}$ has N_v pairs in the file where we denote a particular pattern with an index $p \in \{1, 2, \ldots N_v\}$. In order to handle hidden and output unit thresholds, the input vector $\mathbf{x}_p$ is augmented by an extra element $x_p(N+1)$, where $x_p(N+1) = 1$. For each training pattern p, the net function for the j^{th} hidden unit is given by:

$$n_p(j) = \sum_{k=1}^{N+1} w(j,k) x_p(k) \tag{1}$$

so the net vector is

$$\mathbf{n}_p = \mathbf{W}\mathbf{x}_p \tag{2}$$

where $\mathbf{W}$ is N_h by $N+1$ and the corresponding hidden unit activation $O_p(j) = f(n_p(j))$. Putting everything together, the i^{th} output for the p^{th} training pattern is

$$y_p(i) = \sum_{j=1}^{N+1} w_{oi}(i,j)x_p(j) + \sum_{k=1}^{N_h} w_{oh}(i,k)O_p(k) \tag{3}$$

or in matrix-vector notation

$$\mathbf{y}_p = \mathbf{W}_{oi}\mathbf{x}_p + \mathbf{W}_{oh}\mathbf{O}_p \tag{4}$$

where $\mathbf{W}_{oi} \in \mathbb{R}^{M\times(N+1)}$, contains bypass weights, $\mathbf{W}_{oh} \in \mathbb{R}^{M\times N_h}$ contains weights from hidden units to the outputs and $\mathbf{O}_p$ is a vector of length N_h. Some quantities that we define for convenience are the number of input weights $N_{iw} = N_h(N+1)$, the number of network weights $N_w = N_h(N+1) + M \cdot N_h$, and the number of basis functions $N_u = N + N_h + 1$. Sometimes we refer to $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ as $\mathbf{w}_o$ since they are both connected to the outputs, in particular $\mathbf{w}_o = vec(\mathbf{W}_{oi} : \mathbf{W}_{oh})$ where the $vec()$ operator concatenates matrices into a vector column by column.

3 Neural Network Training

In neural network training we choose network weights to minimize an objective function such as the mean square error (MSE) denoted as $E(\mathbf{w})$, which we will often abbreviate as E. The MSE over a training set, called the training error, is given by

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

where $y_p(i)$ is given in (3) and $\mathbf{w}$ is a vector of network weights, $\mathbf{w} = vec(\mathbf{W}, \mathbf{W}_{oh}, \mathbf{W}_{oi})$.

Output Weight Optimization

One method used to train and initialize neural networks is the Output Weight Optimization algorithm (Manry et al. 1992). OWO calculates the output weight matrices $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ after the input weight matrix, $\mathbf{W}$ is determined in some fashion, usually by random initialization. OWO minimizes the error function

$$E(\mathbf{w}_o) = \mathbf{w}_o^T\mathbf{R}\mathbf{w}_o - 2\mathbf{w}_o^T\mathbf{C} + E_t \tag{6}$$

In equation (6) $\mathbf{R} \in \mathbb{R}^{N_u\times N_u}$, $\mathbf{C} \in \mathbb{R}^{N_u\times M}$, and E_t is sum of average squares of the target vector elements. $\mathbf{R}$ and $\mathbf{C}$ are estimates for the auto- and crosscorrelations of the underlying random process. Equation (6) is minimized by the solution to the linear equations $\mathbf{RW} = \mathbf{C}$. These equations can be solved using any number of methods, but special care must be taken when $\mathbf{R}$ is ill-conditioned. In our work we use orthogonal least squares (Chen, Billings, and Luo 1989). Because (6) is quadratic, OWO is merely Newton's algorithm for the output weights. A modern descendant of OWO is the Extreme Learning Machine (ELM) (Huang, Zhu, and Siew 2006) training.

Newton's Algorithm

Newton's algorithm is the basis of a number of popular second order optimization algorithms including Levenberg-Marquardt (Levenberg 1944) and BFGS (Nocedal and Wright 2006). Newton's algorithm is iterative where each iteration

- Calculates the Newton direction $\mathbf{d}$
- Updates variables with direction $\mathbf{d}$

The direction vector $\mathbf{d}$ is calculated by solving the linear equations

$$\mathbf{Hd} = \mathbf{g} \tag{7}$$

where $\mathbf{H}$ is the Hessian of the objective function and $\mathbf{g}$ is the gradient of the objective function as defined earlier. The variables are then updated as

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{d} \tag{8}$$

Non-quadratic objective functions require a line search. This results in $\mathbf{w}$ being updated as $\mathbf{w} \leftarrow \mathbf{w} + z\mathbf{d}$.

For fast convergence we would like to use Newton's method to train our MLP, but the Hessian for the whole network is singular (Wille 1997). An alternative to overcome this problem is to modify the Hessian matrix as in the Levenberg-Marquardt algorithm. Another alternative is to use two-step methods such as layer by layer training (Lengellé and Denoeux 1996).

4 Motivation for Two-Step Methods

In this section we investigate the assumptions used by Newton's method. Newton's method is derived from a 2^{nd}-order Taylor series approximation to an objective function (Boyd and Vandenberghe 2004). Applying this principle to (5) gives us

$$E_H(\mathbf{w}) = E_0 + (\mathbf{w} - \tilde{\mathbf{w}})^T\mathbf{g} + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \tilde{\mathbf{w}}) \tag{9}$$

where $\tilde{\mathbf{w}}$ is $\mathbf{w}$ from the previous iteration.

When applied to the MSE given in (5), Newton's algorithm assumes that

(A1) E is approximately quadratic in $\mathbf{w}$ for small weight changes

(A2) $y_p(i)$ is well approximated as a first degree function of $\mathbf{w}$.

Note that (A2) follows immediately from (A1). We investigate whether (A2) is a valid assumption by constructing a low degree model for $y_p(i)$. A model that yields the same Hessian and gradient as E is

$$\tilde{E} = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \tilde{y}_p(i)]^2 \tag{10}$$

where $\tilde{y}_p(i)$ is

$$\tilde{y}_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)\left[O_p(k) + O'_p(k)(n_p(k) - \tilde{n}_p(k))\right] \tag{11}$$

where

$$O'_p(k) = \frac{\partial O_p(k)}{\partial n_p(k)}, \tag{12}$$

$$\tilde{n}_p(k) = \sum_{n=1}^{N+1} \tilde{w}(k,n)x_p(n) \tag{13}$$

Here, $\tilde{w}(k,n) = w(k,n)$ but $\tilde{w}(k,n)$ is fixed in the current iteration. We have used a first order Taylor series for each hidden unit for each pattern in the training file. Since we have a different model $\tilde{y}_p(i)$ for each pattern, which is first degree in $\mathbf{x}_p$, we can term $\tilde{y}_p(i)$ a piecewise linear model of $y_p(i)$.

The validity of the piecewise linear model is demonstrated by,

$$\frac{\partial E}{\partial w(u,v)} = \frac{\partial \tilde{E}}{\partial w(u,v)} \tag{14}$$

and

$$\frac{\partial^2 E}{\partial w(u,v)\partial w(m,j)} = \frac{\partial^2 \tilde{E}}{\partial w(u,v)\partial w(m,j)} \tag{15}$$

Also the corresponding errors for each model, $t_p(i) - y_p(i)$ and $t_p(i) - \tilde{y}_p(i)$ are equal for $n_p(k) = \tilde{n}_p(k)$ since

$$\begin{aligned}\frac{\partial y_p(i)}{\partial w(u,v)} &= w_{oh}(i,j)O'_p(u)x_p(v) \\ &= \frac{\partial \tilde{y}_p(i)}{\partial w(u,v)}\end{aligned} \tag{16}$$

When the vector $\mathbf{w}$ includes all the network weights contained in $\mathbf{W}, \mathbf{W}_{oh}$, and $\mathbf{W}_{oi}$, $y_p(i)$ is a not a first order function of the weights $\mathbf{w}$. To show this, we note that the exact expression for output vector $\tilde{\mathbf{y}}_p$ for our network is

$$\begin{aligned}\tilde{\mathbf{y}_p} &= \left[\mathbf{W}_{oi} + \mathbf{W}_{oh} diag(\mathbf{O}'_p)\mathbf{W}\right]\mathbf{x}_p \\ &\quad + \mathbf{W}_{oh}\left[\mathbf{O}_p - diag(\mathbf{O}'_p)\tilde{\mathbf{n}}_p\right]\end{aligned} \tag{17}$$

The model output $\tilde{y}_p(i)$ has products $w_{oh}(i,k)w(k,n)$. If all network weights vary then $\tilde{y}_p(i)$ is second degree in the unknowns and $\tilde{E}$ is a fourth degree model in $\mathbf{w}$ and assumptions (A1) and (A2) are violated. Clearly there is a discrepancy between $\tilde{E}$ in (10) and (9). Since the products $w_{oh}(i,k)w(k,n)$ cause this discrepancy, the corresponding cross terms in the network Hessian $\mathbf{H}$ are sources of error in training a MLP using Newton's method. On the other hand, if $\mathbf{w}$ contains weights from only one layer, the cross terms in (17) are first degree and the discrepancy vanishes as seen in the input weight case in (14) and (15).

This analysis justifies the use of two-step methods where in each iteration we fix $\mathbf{w}_o$ and train $\mathbf{W}$ for one half iteration and fix $\mathbf{W}$ and train $\mathbf{w}_o$ for the other. The approach above is called block coordinate descent (BCD) (Tseng 2001).

5 OWO-Newton

One BCD approach for MLP training is to use Newton's algorithm separately for input and output weights. Note that Newton's algorithm for output weights is OWO.

Calculations for Input Weights

To derive OWO-Newton we first calculate the partial derivative of E with respect to the input weights. The elements of the negative input weight gradient matrix $\mathbf{G} \in \mathbb{R}^{N_h \times (N+1)}$ are given by:

$$\begin{aligned}g(j,k) &= -\frac{\partial E}{\partial w(j,k)} \\ &= \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - y_p(i)]\frac{\partial y_p(i)}{\partial w(j,k)} \\ \frac{\partial y_p(i)}{\partial w(j,k)} &= w_{oh}(i,j)O'_p(j)x_p(k)\end{aligned} \tag{18}$$

Next we calculate the Gauss-Newton input weight Hessian denoted as $\mathbf{H}_R \in \mathbb{R}^{N_{iw} \times N_{iw}}$. Elements of $\mathbf{H}_R$ are found as

$$\frac{\partial^2 E}{\partial w(j,k)\partial w(l,m)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial w(j,k)}\frac{\partial y_p(i)}{\partial w(l,m)} \tag{19}$$

We use orthogonal least squares to solve

$$\mathbf{H}_R\mathbf{d} = \mathbf{g} \tag{20}$$

for $\mathbf{d}$ and update the input weight matrix $\mathbf{W}$ as

$$\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D} \tag{21}$$

where $\mathbf{g} = vec(\mathbf{G})$ and $\mathbf{d} = vec(\mathbf{D})$. Note that in an implementation, the calculations for $\mathbf{H}_R$ should be performed after a lexicographic ordering of $\mathbf{W}$.

In (21), z is the learning factor resulting from a line search. A line search is necessary for OWO-Newton because (5) is not a quadratic function of $\mathbf{w}$.

OWO-Newton Algorithm

```
Require: $\text{MAXITERS} > 0$
  Initialize $\mathbf{W}$
  for k=1 to MAXITERS do
    Perform OWO
    Calculate $\mathbf{G}$
    $\mathbf{g} \leftarrow vec(\mathbf{G})$
    Calculate $\mathbf{H}_R$
    Solve (20) for $\mathbf{d}$
    $\mathbf{D} = vec^{-1}(\mathbf{d})$ as
    $z \leftarrow \arg\min_z E(\mathbf{W} + z\mathbf{D})$
    $\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D}$
  end for
```

Computational Burden

When analyzing the computational complexity of one iteration of OWO-Newton we must first run OWO to train the output weights which requires

$$M_{owo} = N_v(N_u + 1)\left(M + \frac{N_u}{2}\right) \tag{22}$$

multiplies. Next, we calculate the the input weight Hessian which requires

$$M_H = N_v N_{iw}\left(N_{iw} + \frac{3}{2}\right) + \frac{M(N+1)(N+2)}{2} \quad (23)$$

multiplies. Then we must solve the linear equations to calculate the Newton direction which requires

$$M_{ols} = N_{iw}(N_{iw}+1)\left[M + \frac{N_{iw}(N_{iw}+1)}{6}\right] + \frac{3}{2} \quad (24)$$

multiplies. Putting this together, the number of multiplies required for one iteration of OWO-Newton is

$$M_{OWO-Newton} = M_{owo} + M_H + M_{ols} \quad (25)$$

6 Analysis

OWO-Newton has a number of interesting properties, but first we lay the groundwork by defining affine invariance in a neural network training algorithm.

Definition 1. *If two equivalent networks are formed whose objective functions satisfy $E(\mathbf{w}) = E(\mathbf{T}\mathbf{w}')$ with $\mathbf{w} = \mathbf{T}\mathbf{w}'$, and an iteration of an optimization method yields $\mathbf{w} = \mathbf{w}+\mathbf{d}$ and $\mathbf{w}' = \mathbf{w}'+\mathbf{d}'$ where $\mathbf{w}$ and $\mathbf{w}'$ are n-dimensional, the training method is affine invariant if $\mathbf{d} = \mathbf{T}\mathbf{d}'$ for every nonsingular matrix $\mathbf{T}$.*

An algorithm lacks affine invariance if its $\mathbf{T}$ matrix is constrained to be sparse. Affine invariance leads us to the following observation of the training error sequence E_k of equivalent networks.

Lemma 1. *Suppose two equivalent networks initially satisfy $\mathbf{w} = \mathbf{T}\mathbf{w}'$ where $\mathbf{T}$ is any nonsingular $n \times n$ matrix and $\mathbf{w}$ is $n \times 1$. If the training algorithm is affine invariant, the error sequences of the two networks, E_k and $\tilde{E}_k$, for iteration numbers $k \geq 1$ satisfy $E_k = \tilde{E}_k$.*

Proof. Affine invariant training of equivalent networks yields $\mathbf{w}' + \mathbf{d}'$ and $\mathbf{T}(\mathbf{w}' + \mathbf{d}') = \mathbf{w} + \mathbf{d}$, so the networks remain equivalent after one or more iterations. □

Lemma 2. *In OWO-Newton input and output weight training is affine invariant.*

Proof. In OWO-Newton, Newton's algorithm is used for input weight training; OWO is Newton's algorithm for output weight training. □

In multistep algorithms we train subsets of weights so we need to define an appropriate type of affine invariance for this case.

Definition 2. *If a training algorithm satisfies the conditions in Definition 1 except that $\mathbf{T}$ is always sparse, it is partially affine invariant.*

Lemma 3. *The OWO-Newton algorithm is partially affine invariant.*

Proof. OWO is Newton's method for the output weights and Newton's algorithm is used to train the input weights. Therefore by Definition 1 there exist matrices $\mathbf{T}_{OWO}$ and $\mathbf{T}_{Newton}$. As a result, we can construct a sparse $\mathbf{T}$ for the entire algorithm as

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_{OWO} & \vdots & \mathbf{0} \\ \cdots & & \cdots \\ \mathbf{0} & \vdots & \mathbf{T}_{Newton} \end{bmatrix}$$

The existence and sparsity of this $\mathbf{T}$ shows that OWO-Newton is partially affine invariant. □

Lemma 4. *Partially affine invariant algorithms satisfy Lemma 1.*

Proof. The proof is the same as that of Lemma 1. □

When two equivalent networks are trained with OWO-Newton $E_k = \tilde{E}_k$ for $k \geq 1$ because it is partially affine invariant.

7 Experimental Results

In this section we compare the performance of OWO-Newton to LM and CG using 10-fold training and validation. For each fold, all algorithms train a MLP of a given topology from the same initial set of weights.

Wine Data File

We demonstrate OWO-Newton on the wine quality UCI dataset (Frank and Asuncion 2010). The goal is to predict the quality of wine from objective measures (Cortez et al. 2009). The training file consists of 11 features, and 1 target (scored from 0-12) with 4898 training patterns. We train an MLP with 15 hidden units for 200 iterations. The 10-fold training error results are shown in Table 1. On the wine dataset, the training error of OWO-Newton is superior to that of LM and CG; however, the validation error for LM is slightly better. Figure 2 illustrates the computational burden for this training characteristic.

Remote Sensing Data File

We now demonstrate OWO-Newton on the IPNNL remote sensing dataset (Manry 1982). The goal is to predict certain measurements related to electromagnetic scattering such as surface permittivity, normalized surface rms roughness and surface correlation length (Dawson, Fung, and Manry 1993). The training file consists of 8 features and 7 targets with 1768 training patterns. We train a MLP with 10 hidden units for 250 iterations. Table 1 shows the results of the 10-fold training and validation error. The training error of LM is slightly better than OWO-Newton in this example.

Prognostics Data File

Finally, we demonstrate OWO-Newton on a prognostics dataset which can be obtained from the IPNNL at the University of Texas at Arlington(Manry 1982). The prognostics training file contains parameters that are available in a helicopter's health usage monitoring system (HUMS)(Manry,

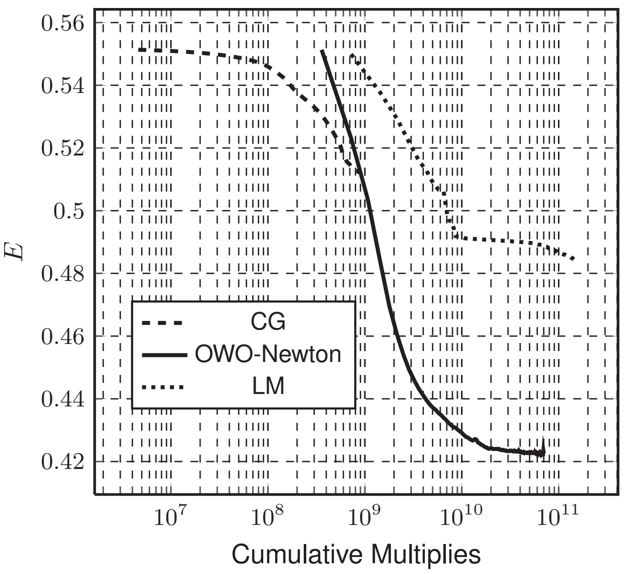

Figure 2: 10-fold training in the Wine dataset

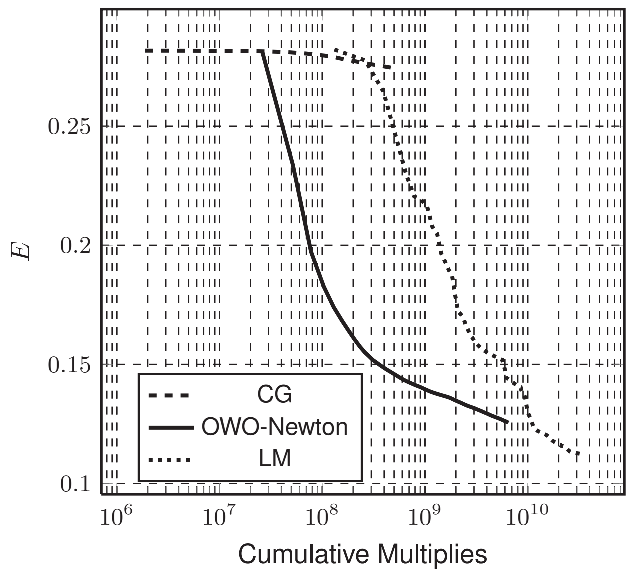

Figure 3: 10-fold training in the Remote Sensing dataset

Hsieh, and Chandrasekaran 1999). It consists of topology of 17 feature vectors and 9 target vectors. The dataset consists of 4745 training patterns. We train a MLP with 15 hidden units for 100 iterations. Table 1 shows that OWO-Newton trains and validates much better than LM and CG. Figure 4 shows that it accomplishes this at much less of a computational cost.

8 Conclusions

In this paper, we have used a piecewise linear network model, implied by Newton's algorithm, to motivate two-step second order training or Block Coordinate Descent. When Newton's algorithm is used in both BCD steps we have the OWO-Newton algorithm. Simulations show that the new algorithm is a good alternative to the Levenberg-Marquardt algorithm. We have demonstrated that OWO-Newton has fast convergence with fewer multiplies for a given error level than LM. However we find that OWO-Newton is not guaranteed to produce a smaller validation error than LM, but that the generalization properties are good.

Finally, we have defined affine invariance in neural networks and introduced a new property called partial affine invariance. Analysis of OWO-Newton shows that the algorithm partially affine invariant.

References

Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. New York, NY, USA: Cambridge University Press.

Charalambous, C. 1992. Conjugate gradient algorithm for efficient training of artificial neural networks. *Circuits, Devices and Systems, IEE Proceedings G* 139(3):301 –310.

Chen, S.; Billings, S.; and Luo, W. 1989. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control* 50(5):1873–1896.

Chen, H.-H.; Manry, M. T.; and Chandrasekaran, H. 1999. A neural network training algorithm utilizing multiple sets of linear equations. *Neurocomputing* 25(1-3):55 – 72.

Cortez, P.; Cerdeira, A.; Almeida, F.; Matos, T.; and Reis, J. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47(4):547–553.

Cybenko, G. 1992. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)* 5:455–455.

Dawson, M.; Fung, A.; and Manry, M. 1993. Surface parameter retrieval using fast learning neural networks. *Remote Sensing Reviews* 7(1):1–18.

Fahlman, S. E., and Lebiere, C. 1990. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, 524–532. Morgan Kaufmann.

Frank, A., and Asuncion, A. 2010. UCI machine learning repository. http://archive.ics.uci.edu/ml.

Hagan, M., and Menhaj, M. 1994. Training feedforward networks with the marquardt algorithm. *Neural Networks, IEEE Transactions on* 5(6):989–993.

Huang, G.-B.; Zhu, Q.-Y.; and Siew, C.-K. 2006. Extreme learning machine: theory and applications. *Neurocomputing* 70(1):489–501.

LeCun, Y.; Bottou, L.; Orr, G.; and Müller, K. 1998. Efficient backprop. In Orr, G., and Müller, K.-R., eds., *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 546–546.

Lengellé, R., and Denoeux, T. 1996. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks* 9(1):83–97.

	OWO-Newton		CG		LM	
	Training Error	Validation Error	Training Error	Validation Error	Training Error	Validation Error
Prognostics	$1.18314 \cdot 10^7$	$1.38424 \cdot 10^7$	$1.30229 \cdot 10^8$	$1.33078 \cdot 10^8$	$3.41 \cdot 10^7$	$3.67 \cdot 10^7$
Twod	0.125447	0.14454	0.27557	0.284679	0.1124	0.137521
Wine	0.422416	0.517243	0.511704	0.53738	0.484221	0.523013

Table 1: 10-fold Training and Validation Error Summary

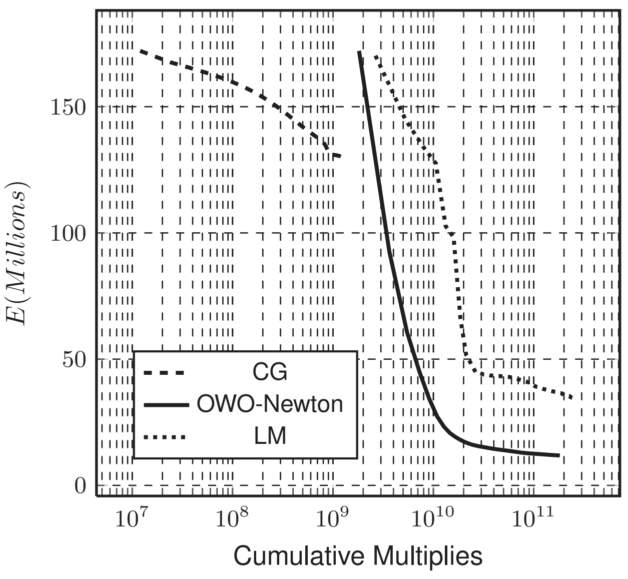

Figure 4: 10-fold training in the Prognostics dataset

Levenberg, K. 1944. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathematics* II(2):164–168.

Li, Y.; Li, T.; and Wang, K. 2011. A new layer by layer training algorithm for multilayer feedforward neural networks. In *Advanced Computer Control (ICACC), 2011 3rd International Conference on*, 600–603. IEEE.

Manry, M.; Guan, X.; Apollo, S.; Allen, L.; Lyle, W.; and Gong, W. 1992. Output weight optimization for the multilayer perceptron. In *Signals, Systems and Computers, 1992. 1992 Conference Record of The Twenty-Sixth Asilomar Conference on*, 502–506. IEEE.

Manry, M.; Hsieh, C.; and Chandrasekaran, H. 1999. Near-optimal flight load synthesis using neural nets. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop.*, 535–544. IEEE.

Manry, M. T. 1982. Image Processing and Neural Network Laboratory. http://www-ee.uta.edu/eeweb/ip/new_training.html.

Nocedal, J., and Wright, S. 2006. *Numerical Optimization*. Springer Verlag.

Shepherd, A. J. 1997. *Second-Order Methods for Neural Networks*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1st edition.

Smagt, P. P. v. d., and Hirzinger, G. 1998. Solving the ill-conditioning in neural network learning. In *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, 193–206. London, UK: Springer-Verlag.

Tseng, P. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optim. Theory Appl.* 109(3):475–494.

Werbos, P. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Dissertation, Harvard University, Cambridge, MA.

Wille, J. 1997. On the structure of the Hessian matrix in feedforward networks and second derivative methods. In *Neural Networks,1997., International Conference on*, volume 3, 1851 –1855 vol.3.