Neural Process Lett (2017) 45:841–854
DOI 10.1007/s11063-016-9553-7

Properties of a Batch Training Algorithm for Feedforward Networks

Melvin D. Robinson[1] · **Michael T. Manry**[2] · **Sanjeev S. Malalur**[3] · **Changhua Yu**[4]

Published online: 14 September 2016

Abstract We examine properties for a batch training algorithm known as the output weight optimization–hidden weight optimization (OWO–HWO). Using the concept of equivalent networks, we analyze the effect of input transformation on BP. We introduce new theory of affine invariance and partial affine invariance for neural networks and prove this property for OWO–HWO. Finally, we relate HWO to BP and show that every iteration of HWO is equivalent to BP applied to whitening transformed data. Experimental results validate the connection between OWO–HWO and OWO–BP.

Keywords Whitening transform · Backpropagation · Output weight optimization · Hidden weight optimization · Equivalent networks · Affine invariance

1 Introduction

The multilayer perceptron (MLP) has several desirable properties including universal approximation [11], the ability to mimic Bayes discriminant [23] and the ability to perform approximate maximum a-posteriori (MAP) estimation [33]. Multilayer perceptrons are used

✉ Melvin D. Robinson
mrobinson@uttyler.edu

Michael T. Manry
manry@uta.edu

Sanjeev S. Malalur
sanjeev.malalur@gmail.com

Changhua Yu
ychmailuta@yahoo.com

1 Department of Electrical Engineering, University of Texas at Tyler, Tyler, TX, USA

2 Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX, USA

3 O-Geo Well LLC, Fort Worth, TX, USA

4 Abbott Labs, Chicago, IL, USA

in many fields including medical image processing [9], face detection [5], and data mining [30].

Training algorithms for the MLP can be classified according to the number of weight change steps per iteration. One-step training algorithms such as backpropagation (BP), conjugate gradient [8,13] and Newton-related algorithms such as quick prop [7] and Levenberg–Marquardt (LM) [10,28] change all of the MLP's weights in one step. Multi-step algorithms [6,21,29] by contrast, change only a portion of the weights at one time. One multi-step algorithm, known as output weight optimization–backpropagation (OWO–BP), alternatively updates input weights using BP and solves linear equations for output weights. [3]

Multi-step algorithms do not guarantee better performance. OWO–BP, like many first order algorithms, is prone to slow convergence. Lacking affine invariance, it is sensitive to input means and the choice of initial weights.

Chen et al. [2] constructed a batch mode layer-by-layer training algorithm called output weight optimization-hidden weight optimization (OWO–HWO), building upon the online training algorithm of Scalero and Tepedelenlioglu [26]. In OWO–HWO, a multi-step algorithm, output weights and hidden weights are alternately modified to reduce the training error. The algorithm modifies the hidden weights based on the minimization of the MSE between a desired net function [2] and the actual net function. The HWO method has been further improved by utilizing a hidden layer Hessian matrix [32].

Although, OWO–HWO performs well and has been used in many applications [4,15,16, 19,20,27], to this point numerous properties are largely unexplored. In this paper, we prove the convergence of OWO–HWO and explore the algorithm's properties. Section 2 introduces MLP notation and describes the OWO–BP algorithm. In this section we also give a definition of partial affine invariance for Neural Networks and prove that OWO–HWO has this property. Using the concept of equivalent networks, the effect of OWO–BP on transformed inputs is established. In Sect. 4, the OWO–HWO method is introduced and its convergence is proved. In Sect. 6 we discuss some of the variations of OWO–HWO. In Sect. 6.2, numerical results are presented. Finally, conclusions are given in Sect. 7.

2 A First Order MLP Training Algorithm

In this section, we first describe the structure and notation of a three layer fully connected MLP with linear output activation functions. Then we review the two-step OWO–BP algorithm and provide a simple proof of its convergence.

2.1 Notation

The network structure is shown in Fig. 1. For clarity, the bypass weights from input layer to output layer are not shown.

Training patterns $\{\mathbf{x}_p, \mathbf{t}_p\}$ for the MLP consist of input vectors $\mathbf{x}_p \in \mathbb{R}^N$, and the machine's output vectors $\mathbf{y}_p \in \mathbb{R}^M$, where the pattern number p varies from 1 to N_v, the number of patterns in the training file. 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 $x_p(N+1) = 1$. For the p^{th} training pattern, the net function vector $\mathbf{n}_p$ is given by

$$\mathbf{n}_p = \mathbf{W}\mathbf{x}_p \tag{1}$$

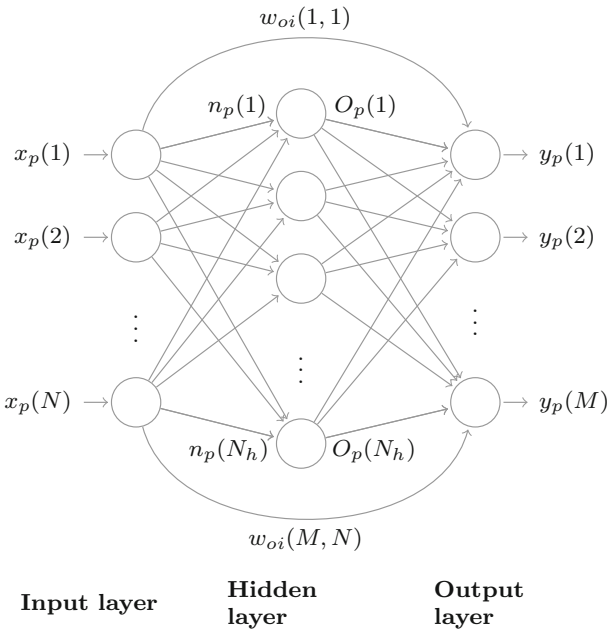

Fig. 1 Multilayer perceptron

and the corresponding jth hidden unit activation $o_p(j)$ is

$$O_p(j) = f(n_p(j)) \tag{2}$$

where $\mathbf{W}$ is an N_h by $N+1$ matrix of hidden weight and thresholds, N_h is the number of hidden units and the $(N+1)^{th}$ column of $\mathbf{W}$ corresponds to the weights associated with the hidden unit thresholds. For the sigmoid activation case, the output of the jth hidden unit is given by

$$f(n_p(j)) = \frac{1}{1+e^{-n_p(j)}} \tag{3}$$

The network's output vector $\mathbf{y}_p$ for the pth training pattern is

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

$$= \left[\mathbf{W}_{oi} \vdots \mathbf{W}_{oh}\right] \begin{bmatrix} \mathbf{x}_p \\ \cdots \\ \mathbf{O}_p \end{bmatrix} \tag{5}$$

where $\mathbf{W}_{oi}$ is an M by $N+1$ matrix, containing bypass weights from inputs to the outputs along with output thresholds. $\mathbf{W}_{oh}$ is of size M by N_h and contains weights from hidden units to the outputs. A typical error function used in batch mode MLP training is the mean-squared error (MSE) described as

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - y_p(i)\right]^2 \tag{6}$$

where $t_p(i)$ denotes the ith element of the pth desired output vector. We also denote the number of network weights as N_w where $N_w = MN_h + (M+N_h)(N+1)$.

2.2 OWO–BP Training Algorithm

Each epoch of OWO–BP training consists of a BP component and a OWO component. During the first half of a training epoch, the BP component updates the input weights and thresholds contained in $\mathbf{W}$ as

$$\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{\Delta W} \tag{7}$$

where z is the optimal learning factor and $\mathbf{\Delta W}$ is the weight update matrix. $\mathbf{\Delta W}$ here is simply equal to $\mathbf{G}$, the negative gradient of E, defined as

$$\mathbf{G} = -\frac{\partial E}{\partial \mathbf{W}} = \frac{1}{N_v}\sum_{p=1}^{N_v} \boldsymbol{\delta}_p \mathbf{x}_p^T \tag{8}$$

For the p^{th} pattern, output and hidden layer delta functions $\delta_{po}(i)$, $\delta_p(k)$ [24] are respectively found as

$$\delta_{po}(i) = -\frac{\partial E}{\partial y_p(i)} = 2\left[t_p(i) - y_p(i)\right] \tag{9}$$

$$\delta_p(k) = -\frac{\partial E}{\partial n_p(k)} = O'(k)\sum_{i=1}^{M} \delta_{po}(k) w_{oh}(i,k) \tag{10}$$

where $O'_p(k)$ is the first derivative of hidden unit activation and $\boldsymbol{\delta} = \left[\delta_p(1), \delta_p(2), \ldots, \delta_p(N_h)\right]^T$. The optimal learning factor z is derived using a Taylor series expansion of the mean square error E, expressed in terms of z, as [14]

$$z = -\frac{\frac{\partial E}{\partial z}}{\frac{\partial^2 E}{\partial z^2}} \tag{11}$$

In the second half of a training epoch, the OWO component [1,18,25] updates the output weights in arrays $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ by solving a set of linear equations $\dfrac{\partial E}{\partial \mathbf{W}_o} = 0$ where $\mathbf{W}_o = [\mathbf{W}_{oi} : \mathbf{W}_{oh}]$. These sets of equations can be written as

$$\begin{bmatrix} \mathbf{R}_i & \vdots & \mathbf{R}_{ih} \\ \cdots & & \cdots \\ \mathbf{R}_{ih}^T & \vdots & \mathbf{R}_h \end{bmatrix} \begin{bmatrix} \mathbf{W}_{oi}^T \\ \cdots \\ \mathbf{W}_{oh}^T \end{bmatrix} = \begin{bmatrix} \mathbf{C}_i \\ \cdots \\ \mathbf{C}_h \end{bmatrix} \tag{12}$$

since the outputs have linear activations. Here the correlation matrices are calculated as $\mathbf{R}_i =< \mathbf{x}_p \mathbf{x}_p^T >$, $\mathbf{R}_{ih} =< \mathbf{x}_p \mathbf{O}_p^T >$, $\mathbf{R}_h =< \mathbf{O}_p \mathbf{O}_p^T >$, $\mathbf{C}_i =< \mathbf{x}_p \mathbf{t}_p^T >$, and $\mathbf{C}_h =< \mathbf{O}_p \mathbf{t}_p^T >$ where the outer product averaging operator is defined as

$$< \mathbf{a}_p \mathbf{b}_p^T >= \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{a}_p \mathbf{b}_p^T \tag{13}$$

If N_u is the number of units (basis functions) connected to every output, then $N_u = N + N_h + 1$. Equation (12) can be condensed as

$$\mathbf{R}_b \mathbf{W}_o^T = \mathbf{C} \tag{14}$$

where, $\mathbf{R}_b$ is the autocorrelation matrix of dimension N_u by N_u for the basis function vector $\left[\mathbf{x}_p^T, \mathbf{O}_p^T\right]^T$ and $\mathbf{C}$ is the cross correlation matrix of dimension N_u by M between the desired

output vector and the basis function vector. The system of equations in (14) can be solved using orthogonal least squares [12].

A description of OWO–BP is given below.

Algorithm 1 OWO–BP Algorithm

```
Initialize W,W_oi,W_oh
Initialize E_0 as the training error for the initial weights
k ← 0
while stopping criterion not reached do
  if k is even then
    Solve the linear equations of (14) and update the output weights
    Calculate E_k
    k ← k + 1
  else
    Find G_k as in (8)
    Line Search: Compute the optimal learning factor z using (11)
    Update the hidden weights using (7)
    k ← k + 1
  end if
end while
```

BP is equivalent to steepest descent and for an optimal learning factor, is guaranteed to converge to a global or local minimum. The OWO–BP technique is attractive for two reasons. First, the training is faster than BP alone, since training weights connected to the outputs is equivalent to solving linear equations. Second, it helps us avoid some local minima.

3 Equivalent Network Theory

In this section, we review equivalent network theory [32] which is a tool for testing training algorithms for affine invariance.

3.1 Basics

Consider the affine transformation

$$\mathbf{x}' = \mathbf{A}\mathbf{x} \tag{15}$$

where $\mathbf{A}$ is a non-singular transformation matrix of size $N+1$ by $N+1$. To analyze the effect of transformed data on BP training, we define equivalent networks.

Definition 1 Consider a nonlinear network with inputs, $\mathbf{x} \in \mathbb{R}^{N+1}$, with the restriction $x(N+1) = 1$ and outputs, $\mathbf{y} \in \mathbb{R}^M$. A second network with inputs $\mathbf{x}'$ has outputs $\mathbf{y}' \in \mathbb{R}^M$. The two networks are strongly equivalent if $\forall \mathbf{x} \in \mathbb{R}^{N+1}$, $\mathbf{y}' = \mathbf{y}$.

As an example, suppose that the equivalent network has the same net vector $\mathbf{n}'_p$ and activation vector $\mathbf{O}_p$ as the original network, so

$$\begin{aligned} \mathbf{n}'_p &= \mathbf{W}'\mathbf{x}'_p \\ &= \mathbf{W}'\mathbf{A}\mathbf{x}_p \\ &= \mathbf{W}\mathbf{x}_p \end{aligned} \tag{16}$$

The original network can be related to the equivalent network as

$$\mathbf{W} = \mathbf{W}'\mathbf{A} \tag{17}$$

$$\mathbf{W}_{oh} = \mathbf{W}'_{oh} \tag{18}$$

$$\mathbf{W}_{oi} = \mathbf{W}'_{oi}\mathbf{A} \tag{19}$$

Lemma 1 *For two MLP networks related through Eqs.* (17)–(19) *to be strongly equivalent it is sufficient that* $rank(\mathbf{A}) = N + 1$.

Proof In (17) and (19), $\mathbf{A}$ is invertible if $rank(\mathbf{A}) = N + 1$, so the strongly equivalent network's weight matrices can be found as

$$\mathbf{W}' = \mathbf{W}\mathbf{A}^{-1} \tag{20}$$

$$\mathbf{W}'_{oh} = \mathbf{W}_{oh} \tag{21}$$

$$\mathbf{W}'_{oi} = \mathbf{W}_{oi}\mathbf{A}^{-1} \tag{22}$$

□

3.2 Learning in Equivalent Networks

Assume that two networks are strongly equivalent as in (17)–(19). If BP is used to train the equivalent network's hidden weights, we have

$$\Delta\mathbf{W}' = z\mathbf{G}' \tag{23}$$

$$\mathbf{G}' = \frac{1}{N_v}\sum_{p=1}^{N_v} \boldsymbol{\delta}'_p \mathbf{x}'^T_p \tag{24}$$

Because the two networks are equivalent, we have $\boldsymbol{\delta}_p = \boldsymbol{\delta}'_p$. Equations (8),(15) and (24) lead to

$$\mathbf{G}' = \mathbf{G}\mathbf{A}^T \tag{25}$$

If the negative gradient matrix $\mathbf{G}'$ is mapped back to the original network by postmultiplying both sides of (25) by $\mathbf{A}$, the input weight change $\mathbf{G}''$ is found as

$$\mathbf{G}'' = \mathbf{G}'\mathbf{A} = \mathbf{G}\mathbf{A}^T\mathbf{A} \tag{26}$$

In other words, unless $\mathbf{A}$ is orthogonal, BP in the equivalent transformed network is not equivalent to BP in the original network. Training in the transformed network is not guaranteed to converge to a local minimum equivalent to that in the original network.

3.3 Affine Invariance

We begin with the following definition of affine invariance in a neural network training algorithm.

Definition 2 If two equivalent networks with weights $\mathbf{w} = vec(\mathbf{W}, \mathbf{W}_{oh}, \mathbf{W}_{oi})$ and $\mathbf{w}' = vec(\mathbf{W}', \mathbf{W}'_{oh}, \mathbf{W}'_{oi})$ 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 $\mathbf{T}$.

Lemma 2 *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 3 *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* E'_k, *for iteration numbers* $k \geq 1$ *satisfy* $E_k = E'_k$.

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. □

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

Definition 3 Assume that two equivalent networks are formed whose objective functions satisfy $E(\mathbf{W}) = E(\mathbf{W}'\mathbf{A})$ with $\mathbf{W} = \mathbf{W}'\mathbf{A}$, where $\mathbf{W}$ is an $i \times m$ coefficient matrix rather than a vector. If an iteration of an optimization method for the two networks yields $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{D}$ and $\mathbf{W}' \leftarrow \mathbf{W}' + \mathbf{D}'$, the method is partially affine invariant if $\mathbf{D} = \mathbf{D}'\mathbf{A}$ for every nonsingular $m \times m$ matrix $\mathbf{A}$.

Lemma 4 *When two equivalent networks undergo partially affine invariant training* $E_k = E'_k$ *for* $k \geq 1$.

This follows from the fact that the two networks remain equivalent after training, as in Lemma 3.

Lemma 5 *BP, CG , and OWO–BP are not affine invariant or partially affine invariant.*

Proof This follows from (26) □

One implication of Lemma 5 is that the three first order algorithms are sensitive to the means of the input vector. This follows because a non-othogonal matrix $\mathbf{A}$ exists that changes the input means.

4 OWO–HWO Algorithm

In this section, we first review the output weight optimization-hidden weight optimization (OWO–HWO) algorithm and then prove its convergence.

4.1 OWO–HWO Description

During the first half of the training epoch, OWO updates the output weights using (14). During the second half of the training epoch, the hidden weights are updated using HWO as

$$\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D} \tag{27}$$

where $\mathbf{D}$ is a matrix of hidden weight changes. Thus OWO–HWO is similar to OWO–BP with the BP component replaced by HWO. For the pth pattern, the desired net function $\mathbf{n}_{pd}$ is constructed as [26]

$$\mathbf{n}_{pd} \cong \mathbf{n}_p + z\boldsymbol{\delta}_p \tag{28}$$

z is the learning rate and $\boldsymbol{\delta}_p$ is the delta function defined in Eq. (10). In OWO–HWO, the weight changes are derived using

$$\mathbf{n}_p + z\boldsymbol{\delta}_p \cong \mathbf{n}_p + z\left(\mathbf{D}\mathbf{x}_p\right) \tag{29}$$

Therefore,

$$\boldsymbol{\delta}_p \cong \mathbf{D}\mathbf{x}_p \tag{30}$$

The error of (30) for the jth hidden unit is measured as

$$E_\delta(j) = \frac{1}{N_v}\sum_{p=1}^{N_v}\left[\delta_{pj} - \sum_{n=1}^{N+1} d(j,n)x_{pn}\right]^2 \tag{31}$$

Setting to zero the derivative of $E_\delta(j)$ with respect to $d(j,n)$, we get

$$\mathbf{D}\mathbf{R}_i = \mathbf{G} \tag{32}$$

where $\mathbf{R}_i$ is the same input autocorrelation matrix used in §2.2. Instead of directly using $\mathbf{G}$ to update the hidden weights, HWO minimizes a separate error function, given by (31), solves the system of linear equations in (32) and updates the hidden weights using (27). Equation (32) can be rewritten as

$$\mathbf{D} = \mathbf{G}\mathbf{R}_i^{-1} \tag{33}$$

OWO–HWO is summarized below. For every training epoch:

Algorithm 2 OWO–HWO Algorithm

Find the negative gradient matrix $\mathbf{G}$ described in equation (8)
Solve the linear equations in (32) to find the HWO update $\mathbf{D}$
Compute the optimal learning factor z using equation (11)
Update the hidden weights using (27)
Solve the linear equations in (32) and update the output weights

5 Properties of HWO and OWO–HWO

Having derived a relationship between OWO–HWO and OWO–BP, we use equivalent network theory to prove several properties.

5.1 Relationship of OWO–HWO to Whitening

Lemma 6 *HWO training with data* $\{\mathbf{x}_p, \mathbf{t}_p\}$ *is equivalent to BP training with data* $\{\mathbf{x}'_p, \mathbf{t}_p\}$ *where* $\mathbf{x}'_p = \mathbf{A}\mathbf{x}_p$ *and* $\mathbf{A}$ *denotes the whitening transform matrix* [22].

Proof Expressing the singular value decomposition (SVD) of the autocorrelation matrix as

$$\mathbf{R}_i = \mathbf{U}\Sigma\mathbf{U}^T \tag{34}$$

its inverse is

$$\mathbf{R}_i^{-1} = \mathbf{U}\Sigma^{-1}\mathbf{U}^T \tag{35}$$

Equating $\mathbf{G}''$ in (26) to $\mathbf{D}$ in (33) we get

$$\mathbf{D} = \mathbf{G}\mathbf{A}^T\mathbf{A} \tag{36}$$

where $\mathbf{A}$ is the whitening transform matrix [22],

$$\mathbf{A} = \Sigma^{-\frac{1}{2}}\mathbf{U}^T \tag{37}$$

Comparing (26) and (36), it is clear that training hidden weights with HWO for original data is equivalent to training hidden weights with BP for data transformed by matrix $\mathbf{A}$. □

Raudys [22] suggests that applying BP after a whitening transform can lead to faster convergence. HWO is equivalent to that approach.

Lemma 7 *There are an uncountably infinite number of whitening transform matrices* $\mathbf{A}$ *that have identical performance.*

Proof Let $\mathbf{A} = \mathbf{Z}\Sigma^{-\frac{1}{2}}\mathbf{U}$ where $\mathbf{Z}$ is any orthogonal matrix. Because $\mathbf{Z}$ is orthogonal, substituting this $\mathbf{A}$ matrix into (36) results in $\mathbf{Z}$ cancelling out. Because the number of orthogonal matrices is uncountably infinite, there is an uncountably infinite number of whitening transform matrices that have identical performance. □

5.2 Partial Affine Invariance

Lemma 8 *The HWO algorithm is partially affine invariant.*

Proof For a given MLP and an arbitrary nonsingular matrix $\mathbf{A}$, construct an equivalent MLP using (20)–(22). From (32) the update matrix, $\mathbf{D}$ for the equivalent network satisfies

$$\mathbf{D}'\mathbf{R}_i' = \mathbf{G}' \tag{38}$$

Using (25)

$$\mathbf{D}'\mathbf{A}\mathbf{R}_i\mathbf{A}^T = \mathbf{G}\mathbf{A}^T \tag{39}$$

postmultiplying by $\left(\mathbf{A}^T\right)^{-1}$ we get,

$$\mathbf{D}'\mathbf{A}\mathbf{R}_i = \mathbf{G} \tag{40}$$

Equating (32) and (40),

$$\mathbf{D}'\mathbf{A}\mathbf{R}_i = \mathbf{D}\mathbf{R}_i \tag{41}$$

Postmultiplying by $\mathbf{R}_i^{-1}$,

$$\mathbf{D}'\mathbf{A} = \mathbf{D} \tag{42}$$

From (20) and (42) the weight update operations of the two networks are

$$\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D} \tag{43}$$

$$\mathbf{W}\mathbf{A}^{-1} \leftarrow \mathbf{W}\mathbf{A}^{-1} + z\mathbf{D}\mathbf{A}^{-1} \tag{44}$$

From (42) and Definition 3, we see that HWO is partially affine invariant. □

Lemma 9 *The OWO–HWO algorithm is partially affine invariant.*

Proof OWO is Newton's method for the output weights. We can construct $\mathbf{T}$ as $\mathbf{T} = \begin{bmatrix} \mathbf{T}_{HWO} & \vdots & \mathbf{0} \\ \cdots & & \cdots \\ \mathbf{0} & \vdots & \mathbf{T}_{OWO} \end{bmatrix}$.

Since $\mathbf{T}$ for the entire algorithm is sparse, OWO–HWO is partially affine invariant via Lemma 2 and Definition 3. □

One of the implications of partial affine invariance in HWO and OWO–HWO is that they are immune from the effects of input biases. We will illustrate this further in §6.2.

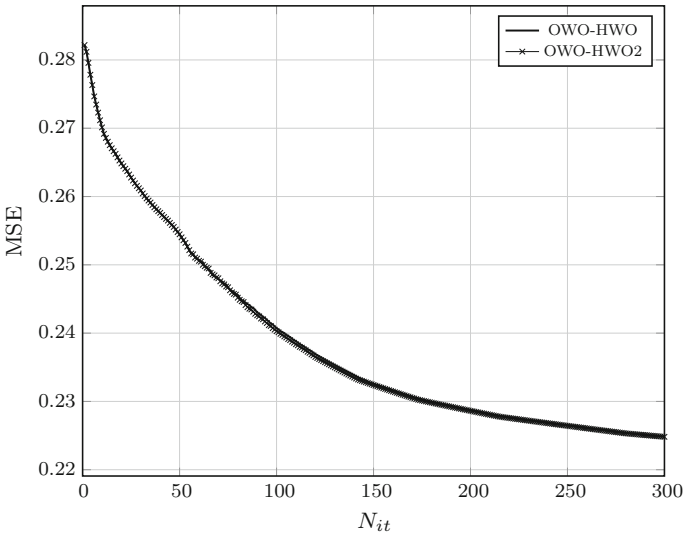

Fig. 2 Comparison of OWO–HWO and OWO–HWO2

6 Variations of OWO–HWO

There are two variations of the HWO component of OWO–HWO that differ in the way **D** is calculated. HWO uses orthogonal least squares to solve the system of linear equations in (32). The second version, HWO2, first transforms the inputs according to Eq. (15) with **A** defined in (37) and then uses BP on transformed data to calculate **D**. It should be mentioned that the OWO stage is the same in all variants. Depending on the variant of HWO used, there will be some additional computational overhead.

6.1 Computational Burden

We can now comment on the advantages of each variant. Let M_{hwo} and M_{hwo2} denote the number of multiplies for the three variants described above.

M_{bp} and M_{olf} respectively denote the numbers of multiplies for computing the BP negative gradient matrix **G** and the optimal learning factor z and are given by

$$M_{owo-bp} = N_{it}\{N_v[2N_h\,(N+2) + M\,(N_u+1) + \frac{N_u\,(N_u+1)}{2} + M\,(N+6N_h+4)] + M_{ols} + N_h\,(N+1)\} \quad (45)$$

$$M_{bp} = N_{it}\{N_v[MN_u + 2N_h(N+1) + M(N+6N_h+4)] + N_w\} \quad (46)$$

then we calculate the computation burden as

$$M_{hwo} = N_{it}(M_{owo-bp} + M_{bp} + M_{olf}) + N_{it}[N_h(N+1)(N+2)] \quad (47)$$

$$M_{hwo2} = N_{it}\left(M_{owo-bp} + M_{bp} + M_{olf}\right) + N_{it}\left[N_h(N+1)^2\right] + (N+1)^3 \quad (48)$$

where N_{it} is the number of training iterations or epochs.

$$M_{owo} = N_v(N_u+1)\left(M + \frac{N_u}{2}\right) \quad (49)$$

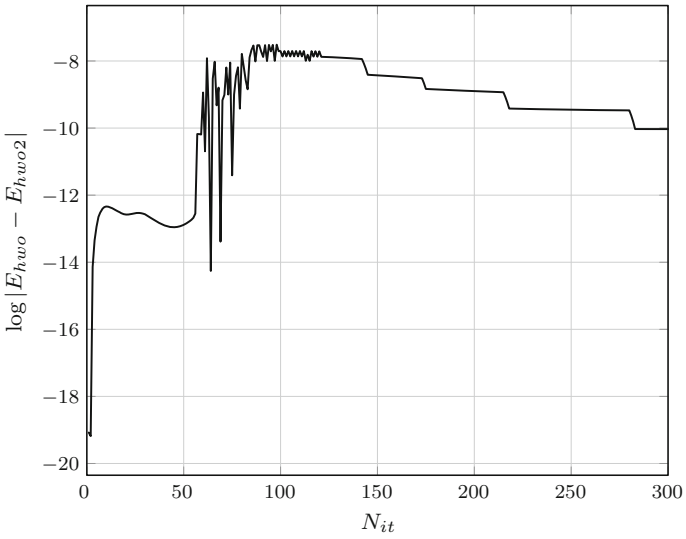

Fig. 3 Numerical difference between OWO–HWO and OWO–HWO2

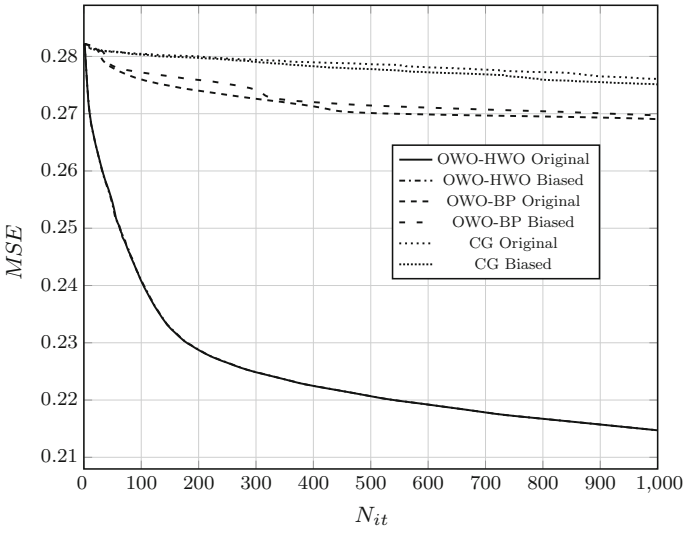

Fig. 4 Illustration of Affine invariance in OWO–HWO

6.2 Examples

In this section, we illustrate some of the properties in the previous sections with a remote sensing data file [17] having 1768 patterns of 8 inputs and 7 outputs.

In our first experiment we test the claims in Lemma 6 by training original data with OWO–HWO. We then train whitening transformed data as described in §5 with OWO–BP. We label this OWO–HWO2. Each of the MLPs are trained with 10 hidden units. In Fig. 2, the plots for OWO–HWO and OWO–HWO2 have significant overlap as predicted by Lemma 6. This

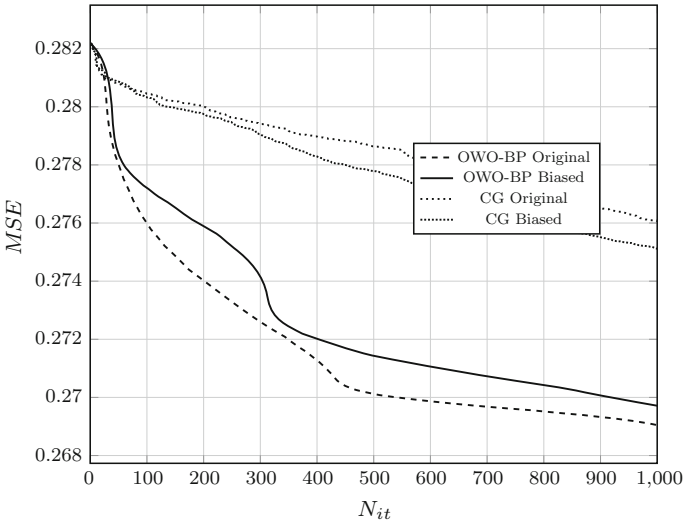

Fig. 5 Effect of input biases on OWO–BP and CG

similarity is spelled out more clearly in Fig. 3 by plotting the log of the absolute difference. We can conclude that OWO–HWO and OWO–HWO2 are identical within numerical precision and roundoff error of the machine.

Finally, we examine the responses of first order training algorithms to biased and unbiased versions of the remote sensing data file. We observe significant overlap between original and biased data for OWO–HWO in Fig. 4, as we would expect from Lemmas 4 and 9. The figure also confirms that OWO–BP and CG are not affine invariant because there is a small, but discernible difference between original and biased training.

Figure 5 more clearly shows this difference due to the influence of input biases on OWO–BP and CG.

7 Conclusions

Using the concept of equivalent networks, we have shown that OWO–HWO, is equivalent to applying OWO–BP to whitening transformed [22] input data. The convergence of OWO–BP training therefore proves the convergence of OWO–HWO. In addition, we have shown that the whitening transform matrix is not unique.

We have analyzed two methods for implementing OWO–HWO and considered the relative efficiencies of these implementations. Finally, we have defined partial affine invariance and proven that the HWO and OWO–HWO training algorithms have this property. One benefit of partial affine invariance is immunity to input biases which we have demonstrated in an example.

References

1. Barton S (1991) A matrix method for optimizing a neural network. Neural Comput 3(3):450–459

2. Chen HH, Manry MT, Chandrasekaran H (1999) A neural network training algorithm utilizing multiple sets of linear equations. Neurocomputing 25(1–3):55–72. doi:10.1016/S0925-2312(98)00109-X
3. Dawson M, Fung A, Manry M (1993) Surface parameter retrieval using fast learning neural networks. Remote Sens Rev 7(1):1–18
4. De Bernardez L, Buitrago R, García N (2011) Photovoltaic generated energy and module optimum tilt angle from weather data. Int J Sustain Energy 30(5):311–320
5. Ebrahimpour R, Kabir E, Yousefi M (2007) Face detection using mixture of MLP experts. Neural Process Lett 26:69–82. doi:10.1007/s11063-007-9043-z
6. Ergezinger S, Thomsen E (1995) An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer. IEEE Trans Neural Netw 6(1):31–42
7. Fahlman SE (1988) Faster-learning variations on back-propagation: an empirical study. In: Proceedings of the 1988 connectionist models summer school. Morgan Kaufmann, San Mateo, pp 38–51
8. Fletcher R (1987) Practical methods of optimization, 2nd edn. Wiley, New York
9. Güler E, Sankur B, Kahya Y, Raudys S (1998) Visual classification of medical data using MLP mapping. Comput Biol Med 28(3):275–287
10. Hagan M, Menhaj M (1994) Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993
11. Hornik K, Stinchcombe M, White H (1990) Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Netw 3(5):551–560
12. Kaminski W, Strumillo P (1997) Kernel orthonormalization in radial basis function neural networks. IEEE Trans Neural Netw 8(5):1177–1183
13. Kim T, Manry M, Maldonado J (2003) New learning factor and testing methods for conjugate gradient training algorithm. In: Proceedings of the international joint conference on neural networks, 2003. IEEE, vol 3, pp 2011–2016
14. LeCun Y, Bottou L, Orr G, Müller K (1998) Efficient backprop. In: Orr G, Müller KR (eds) Neural Networks: tricks of the trade, vol 1524. Lecture Notes in Computer Science. Springer, Berlin, pp 546–546
15. Li J, Manry M, Narasimha P, Yu C (2006) Feature selection using a piecewise linear network. IEEE Trans Neural Netw 17(5):1101–1115
16. Li J, Yao J, Petrick N, Summers R, Hara A (2006) Hybrid committee classifier for a computerized colonic polyp detection system. In: Proceedings of the SPIE medical imaging. Citeseer, San Diego
17. Manry MT (1982) Image processing and neural network laboratory. http://www-ee.uta.edu/eeweb/ip/new_training.html
18. Manry MT, Apollo SJ, Allen LS, Lyle WD, Gong W, Dawson M, Fung AK (1994) Fast training of neural networks for remote sensing. Remote Sens Rev 9:77–96. doi:10.1016/0893-6080(89)90020-8
19. Mantena V, Jiang W, Li J, McKenzie R (2009) Prostate cancer biomarker identification using maldi-ms data: initial results. In: Life science systems and applications workshop, 2009. LiSSA 2009. IEEE/NIH. IEEE, pp 116–119
20. Ondimu S, Murase H (2007) Reservoir level forecasting using neural networks: Lake Naivasha. Biosyst Eng 96(1):135–138
21. Parisi R, Di Claudio E, Orlandi G, Rao B (1996) A generalized learning paradigm exploiting the structure of feedforward neural networks. IEEE Trans Neural Netw 7(6):1450–1460
22. Raudys S (2001) Statistical and neural classifiers: an integrated approach to design. Springer, Berlin
23. Ruck D, Rogers S, Kabrisky M, Oxley M, Suter B (1990) The multilayer perceptron as an approximation to a Bayes optimal discriminant function. IEEE Trans Neural Netw 1(4):296–298. doi:10.1109/72.80266
24. Rumelhart D, Hinton G, Williams R (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
25. Sartori M, Antsaklis P (1991) A simple method to derive bounds on the size and to train multilayer neural networks. IEEE Trans Neural Netw 2(4):467–471
26. Scalero R, Tepedelenlioglu N (1992) A fast new algorithm for training feedforward neural networks. IEEE Trans Signal Process 40(1):202–210
27. Sheikhan M, Shabani AA (2013) Pso-optimized modular neural network trained by OWO–HWO algorithm for fault location in analog circuits. Neural Comput Appl 23(2):519–530
28. Shepherd AJ (1997) Second-order methods for neural networks, 1st edn. Springer, New York
29. Wang G, Chen C (1996) A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures. IEEE Trans Neural Netw 7(3):768–775
30. Wang L (2009) Data mining with computational intelligence. Springer, Berlin
31. Yeh J (2006) Real analysis: theory of measure and integration. World Scientific Publishing Company Incorporated, Singapore

32. Yu C, Manry MT, Jiang L (2005) Effects of nonsingular preprocessing on feedforward network training. Int J Pattern Recognit Artif Intell 19(02):217–247. doi:10.1142/S0218001405004022
33. Yu Q, Apollo S, Manry M (1993) Map estimation and the multilayer perceptron. In: Proceedings of the 1993 IEEE-SP workshop on neural networks for processing [1993] III. IEEE, pp 30–39