Abstract—The concept of equivalent networks is reviewed as a method for testing algorithms for affine invariance. Partial affine invariance is defined and introduced to first order training through the development of linear transforms of the hidden layer’s net function vector. The resulting two-step training algorithm has convergence properties that are comparable to Levenberg-Marquardt, but with fewer multiplications per iteration.

I. INTRODUCTION

The multilayer perceptron (MLP) is a nonlinear signal processor that has found use in a wide variety of applications. Its ability to approach the Bayes classifier [1] makes it useful for the classification problem and its universal approximation property [2] makes it well-suited for the function approximation problem.

Training a MLP can be challenging since it involves solving a non-convex optimization problem and various problems such as slow convergence, high computational complexity and rank deficiency must be overcome.

When possible, second order training is desired due to its fast convergence; however, the tradeoff is increased computational complexity. This has been partially addressed by the development of the Optimal Input Gain (OIG) algorithm [3] and the Multiple Optimal Learning Factors (MOLF) algorithm [4], both of which significantly improve input weight training.

In this work we extend MOLF to the case of dense transform matrices and analyze the kind of affine invariance present in these new algorithms. In section 2, we briefly introduce our MLP architecture and notation. We review first and second order training algorithms and their properties in section 3. In section 4, we review equivalent network concepts. In sections 5 we derive the Optimal Net Transform algorithm and use it as one half of a two-step training algorithm. Properties of this new algorithm are discussed in section 6. Finally, we provide some experimental results.

II. NOTATION AND ARCHITECTURE

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 inputs are \( x_p(n) \) where \( 1 \leq n \leq N \), and the desired outputs are \( t_p(i) \) where \( 1 \leq i \leq M \).

The training data \( \{x_p, t_p\} \) has \( N_v \) patterns where each pattern is identified by an index \( p \), so \( 1 \leq p \leq N_v \). In order to handle hidden and output unit thresholds, the input vector \( x_p \) is augmented by an extra element \( x_p(N + 1) = 1 \).

III. NEURAL NETWORK TRAINING PRELIMINARIES

The objective function to be minimized is the mean square error, often abbreviated as \( E \) and given by

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

where \( y_p(i) \) is the \( i^{th} \) element of \( y_p \).
First Order Training

Backpropagation (BP) is a first order training algorithm described by Werbos[5] which trains all network weights. These weights can be consolidated into an $N_w$-dimensional vector $w$. In BP, we use the negative gradient $g = -\frac{\partial E}{\partial w}$ to update $w$ as $w \leftarrow w + zg$ where $z$ is a learning factor.

Training performance and perhaps validation performance can be improved by finding a better update vector $d$. The weights would then be updated as $w \leftarrow w + zd$

Enhanced Direction Matrix

The Hidden Weight Optimization (HWO) algorithm [6] produces an improved weight change matrix $D$ by solving

$$DR = G$$  \hspace{1cm} (4)$$

where $R$ is the input autocorrelation matrix

$$R = \frac{1}{N_v} \sum_{p=1}^{N_v} x_p x_p^T$$  \hspace{1cm} (5)$$

and $G = -\frac{\partial E}{\partial W}$. In each iteration, after updating $W$, the output weights are calculated using the Output Weight Optimization algorithm (OWO) [7] which solves linear equations for output weights. Using HWO is equivalent to performing a whitening transform [8] on the inputs followed by BP.

The OWO-HWO algorithm has two steps per iteration, an HWO step to modify the input weights and an OWO step to modify output weights. Equation (4) suggests the possibility of improving upon the more typical negative gradient $G$ by calculating $D$ as $GR^{-1}$ where $R^{-1}$ is a dense transformation matrix.

Newton’s Algorithm

Newton’s algorithm is the basis of a number of popular second order optimization algorithms including Levenberg-Marquardt [9] and BFGS [10]. Newton’s algorithm is iterative where each iteration

- Calculates the Newton direction $d$ [10]
- Updates variables with direction $d$ as $w \leftarrow w + d$

Two essential properties of Newton’s method are

- Quadratic Convergence
- Affine Invariance

IV. EQUIVALENT NETWORKS

In this section we define equivalent networks and use them to illustrate the lack of affine invariance in first order training.

Definition 1. Two MLPs are strictly equivalent if for every input vector $x_p$, the two networks’ output vectors $y_p$ and $y'_p$ are precisely the same.

In a single hidden layer MLP, there are at least two points where we can insert a factored identity matrix in order to generate an equivalent network: between the input vector and input weight matrix and in the net vector layer.

In each iteration, the input weight matrix $W$ is to be modified using a weight change matrix $D$ from HWO.

Net Equivalence

Let the first MLP have the notation described earlier. In the second MLP, suppose that the network calculates an $N_h$-dimensional net vector $n'$ as

$$n' = W'x$$  \hspace{1cm} (6)$$

where the final net vector is found through linear transformation as

$$n = Cn'$$  \hspace{1cm} (7)$$

where $C$ is nonsingular. Then $C$ is related to $W'$ and $W$ as

$$CW' = W$$  \hspace{1cm} (8)$$

In other words, we have taken the original network and inserted the matrix $C^{-1}C$ between the net vector and an equivalent net vector. The two MLPs described here are net-equivalent, which is a form of strict equivalence.

If HWO is used to train $W$ and $W'$, their input weight change matrices, $D$ and $D'$, are related as

$$D' = C^T D$$  \hspace{1cm} (9)$$

If the weight changes in MLP 2 are mapped back to MLP 1, the weight changes for MLP 1 become

$$D'' = R_n D$$  \hspace{1cm} (10)$$

where $R_n = CC^T$ as described in [4]. If $C$ is not an orthogonal matrix then, MLPs 1 and 2 train differently.

Summary of First Order Training Problems

1) Training error decreases slowly, unlike the quadratic convergence of second order algorithms
2) First order algorithms lack affine invariance

The first problem results in long training times for satisfactory network performance. The second problem also results in a longer than necessary training times since it is unlikely that $R_n$ in (10) generates an optimal $D''$.

Lack of affine invariance can be observed in (10) if $D$ and $D''$ are respectively replaced by $G$ and $G''$, yielding $G'' = R_n G$. If $R_n$ is not an identity matrix, then $G''$ differs from $G$ and equivalent networks train differently for the two algorithms. This confirms that first order MLP training algorithms such as BP and Conjugate Gradient (CG) lack affine invariance.

V. THE OPTIMAL NET TRANSFORM ALGORITHM

In this section we discuss the Optimal Net Transform algorithm (ONT), which improves the input weight matrix $W$. ONT is combined later with OWO, yielding a two-step training algorithm. Using (10) we can improve $D$ from HWO by premultiplying it with an optimal matrix $R_n$ found as

$$R_n = \arg \min_{R} E(W + RD)$$  \hspace{1cm} (11)$$
Details

Following (10) the updated net function for ONT is
\[ n_p(k) = \sum_{n=1}^{N+1} \left[ w(k, n) + \sum_{j=1}^{N_h} r_n(k, j) d(j, n) \right] x_p(n) \] (12)

The partial derivative of \( E(W + R_n D) \) with respect to an element of the transform matrix \( R_n \) is
\[ \frac{\partial E}{\partial r_n(j, m)} = -2 \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[ t_p(i) - y_p(i) \right] \frac{\partial y_p(i)}{\partial r_n(j, m)} \] (13)

The second partial is of \( E(W + R_n D) \) with respect to \( R_n \) is
\[ \frac{\partial^2 E}{\partial r_n(j, m) \partial r_n(u, v)} = 2 \sum_{p=1}^{N_v} \sum_{i=1}^{M} \frac{\partial y_p(i)}{\partial r_n(j, m)} \frac{\partial y_p(i)}{\partial r_n(u, v)} \] (14)

\[ \frac{\partial y_p(i)}{\partial r_n(k, v)} = w_{oh}(i, k) f'(n_p(k)) \frac{\partial n_p(k)}{\partial r_n(k, v)} \] (15)

where
\[ \frac{\partial n_p(k)}{\partial r_n(k, v)} = \sum_{n=1}^{N+1} d(v, n) x_p(n) \] (16)

Mapping (13) to \( g_n \) and (14) to \( H_n \), we have the standard equations for Newton’s algorithm,
\[ H_n r_n = g_n \] (17)

which are solved for \( r_n \) using orthogonal least squares (OLS). \( R_n \) is found from \( r_n \) as \( R_n = \text{vec}^{-1}(r_n) \). Combining ONT, which improves input weights, with OWO we get the following two-step training algorithm.

**OWO-ONT Algorithm**

**Require:** MAXITERS > 0

\[ k = 0 \]

while \( k < \text{MAXITERS} \) do

- Calculate \( W_{oh} \) and \( W_{ai} \) using OWO
- Calculate a descent direction \( D \)
- Calculate \( g_n \) elements using (13)
- Calculate \( H_n \) elements using (14)
- Solve (17) for \( r_n \)
- \( R_n = \text{vec}^{-1}(r_n) \)
- Update the input weight matrix as \( W \leftarrow W + R_n D \)

if stopping criteria reached then

STOP

end if

\[ k \leftarrow k + 1 \]

end while

In an earlier version of this algorithm, known as OWO-MOLF [4], we constrained \( R_n \) to be a diagonal matrix.

VI. Partial Affine Invariance

It is well known that Newton’s algorithm has quadratic convergence and is affine invariant[11]. We can define affine invariance in neural networks as follows.

**Definition 2.** If two equivalent networks are formed whose objective functions satisfy \( E(w) = E(Tw') \) with \( w = Tw' \), and an iteration of an optimization method yields \( w = w + \Delta \) and \( w' = w' + \Delta' \) where \( w \) and \( w' \) are \( n \)-dimensional, the training method is affine invariant if \( \Delta = T \Delta' \) for every nonsingular matrix \( T \).

An algorithm lacks affine invariance if its \( T \) matrix is constrained to be sparse, but may have a different form of affine invariance:

**Definition 3.** If a training algorithm satisfies the conditions in Definition 2 except that \( T \) is always sparse it is partially affine invariant.

Partial affine invariance leads us to the following observation of the training error sequence of equivalent networks.

**Lemma 1.** Suppose two equivalent networks initially satisfy \( w = Tw' \) where \( T \) is any nonsingular \( n \times n \) matrix consistent with the training algorithm and \( w \) is \( n \times 1 \). If the training algorithm is affine invariant or partially 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.** If two networks start out equivalent, then they remain equivalent after affine invariant or partially affine invariant training.

Partial affine invariance ensures that training cannot be improved by an affine transformation of either the data or the weights. Such is not the case for all training algorithms.

VII. Properties of ONT and OWO-ONT

**Lemma 2.** If two net equivalent MLPs are trained using ONT, they remain net equivalent afterwards.

**Proof.** We apply ONT to both the original and the net-equivalent networks. The input weight matrices \( W \) and \( W' \) are modified as
\[ W \leftarrow W + R_n D \] (18)
\[ W' \leftarrow W' + R'_n D' \] (19)

Multiplying the equivalent network’s update in (19) by \( C \) and remembering that \( D' = C^T D \), we get
\[ W \leftarrow W + CR'_n C^T D \] (20)

Since ONT uses Newton’s method to find \( R_n \) and \( R'_n \), affine invariance means that the two networks remain equivalent after the iteration and
\[ R_n = CR'_n C^T \] □
At this point we see that ONT is either affine invariant or partially affine invariant. We need to determine the properties of $T$ to determine which property the algorithm has.

**Lemma 3.** ONT is affine invariant if
1) $N_h \geq N + 1$ and
2) $\text{rank}(D)$ is $N + 1$.

Otherwise ONT is partially affine invariant.

**Proof.** First, assume that we have two identical networks, one where $W$ is to be trained by ONT and the other where $W$ is to be trained using Newton’s algorithm. If the two algorithms train identically, we have

$$R_n D = E_N$$  \hspace{1cm} (21)

where $E_N$ is the $N_h \times (N + 1)$ weight change matrix for $W$ from Newton’s algorithm and $R_n D$ is the $N_h \times (N + 1)$ weight change matrix for $W$ from ONT. Transposing both sides of (21) we get

$$D^T R_n^T = E_N^T$$  \hspace{1cm} (22)

Case 1: $N_h \geq N + 1$ (underdetermined system)

Here we have $N_h$ sets of $N + 1$ equations in $N_h$ unknowns, which can be solved exactly if $\text{rank}(D) = N + 1$. Since ONT is Newton’s algorithm for $W$, ONT is affine invariant and $T_{ONT}$ the matrix $T$ for ONT, is dense.

Case 2: $N_h < N + 1$ (overdetermined system)

Here we have more equations than unknowns so (22) is not solved exactly. 

**Lemma 4.** The OWO-ONT algorithm is partially affine invariant.

**Proof.** OWO is Newton’s method for the output weights and Newton’s method is affine invariant. For the input weights, $R_n$ is calculated with Newton’s method which is affine invariant. Because we can construct a sparse $T$ for the entire OWO-ONT algorithm as

$$T = \begin{bmatrix} T_{owo} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & T_{ONT} \end{bmatrix}$$

ONT is partially affine invariant.

Because OWO-ONT is partially affine invariant, it satisfies Lemma 1.

**VIII. Computational Cost**

When analyzing the computational complexity of one iteration of OWO-ONT, we must first complete an OWO stage which requires

$$M_{owo} = N_v (N_u + 1) \left( M + \frac{N_u}{2} \right)$$  \hspace{1cm} (23)

multiplies where $N_u = N + N_h + 1$. Next, we calculate the the input weight Hessian which requires

$$M_H = N_v \left[ (N + 1) (N_h + 1) + N_h \right. \left. + MN_h^2 + \frac{N_h^2 (N_h^2 + 1)}{2} \right]$$  \hspace{1cm} (24)

Then we must solve the linear equations to calculate $r_n$ which requires

$$M_{ols} = N_h (N_h + 1) \left[ M + \frac{N_h (N_h + 1)}{6} \right] + \frac{3}{2}$$  \hspace{1cm} (25)

multiplies. The number of multiplies required for the $R_n D$ product is

$$M_{product} = N_h (N + 1) (2N_h - 1)$$  \hspace{1cm} (26)

Putting this together we have

$$M_{ONT} = M_{owo} + M_{OLS} + M_{product} + M_H$$  \hspace{1cm} (27)

By comparison MOLF, a sparse version of ONT, requires

$$M_{molf} = M_{owo-bp} + N_v \left[ N_h (N + 4) - M (N + 6N_h + 4) \right] + N_h^3$$  \hspace{1cm} (28)

multiplies where

$$M_{owo-bp} = N_v \left[ 2N_h (N + 2) + M (N_u + 1) + \frac{N_u (N_u + 1)}{2} + M (N + 6N_h + 4) \right]$$  \hspace{1cm} (29)

$$M_{ols} + N_h (N + 1)$$

Here $M_{owo-bp}$ denotes the number of multiplies of a two-step algorithm called OWO-BP [12] that alternately uses backpropagation to improve the input weights and uses OWO to solve for the output weights.

**IX. Experimental Results**

In this section we use ten-fold training and validation for each datafile. The validation takes place over several runs of randomly permuted training files. Our method is independent of CPU speed and rather concentrates on number of iterations and cumulative multiplies. This methodology allows an apples to apples comparison of training efficiency and performance.

Table I provides a summary of training and validation error.

<table>
<thead>
<tr>
<th>Datafile</th>
<th>Training Error</th>
<th>Validation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E_t</td>
<td>E_v</td>
</tr>
</tbody>
</table>

Remote Sensing Data File

We now demonstrate OWO-ONT on the IPNNL remote sensing dataset [13]. The goal is to predict certain measurements related to electromagnetic scattering such as surface permittivity, normalized surface rms roughness and surface correlation length [12]. The training file consists of 8 features and 7 targets with 1768 training patterns. We train an MLP
with 10 hidden units for 250 iterations. The results of the 10-fold training error are shown in Figures 2 and 3. The training error of LM is slightly better than OWO-ONT in this example.

Table I shows that the validation error for LM is slightly better than that of ONT.

**Oh7 Data file**

We demonstrate OWO-ONT on the Oh7 dataset [13]. It has 10453 training patterns. We train an MLP with 15 hidden units for 200 iterations. The results of the training error are shown in Figures 4 and 5. Table I shows that the validation error for ONT is slightly better than that of LM.

![Fig. 2. 10-fold training error in Remote Sensing](image)

![Fig. 4. 10-fold training error of Oh7](image)

Finally, we demonstrate OWO-ONT on a prognostics dataset which can be obtained from the IPNNL at the University of Texas at Arlington[13]. The prognostics training file contains parameters that are available in a helicopter’s health usage monitoring system (HUMS)[14]. The dataset has 17 inputs and 9 outputs and consists of 4745 training patterns. We train an MLP with 15 hidden units for 200 iterations. Figures 6 and 7 shows that OWO-ONT trains much better than LM. By adjusting the density of the transform matrix, we are able to adjust the convergence rate and computational cost of MLP training. Table I shows that the training and validation error for ONT is significantly better than that of LM.

![Fig. 3. Computational cost of 10-fold training of the Remote Sensing dataset](image)

![Fig. 5. Computational cost of 10-fold training of the Oh7 dataset](image)

**Prognostics Data File**

Finally, we demonstrate OWO-ONT on a prognostics dataset which can be obtained from the IPNNL at the University of Texas at Arlington[13]. The prognostics training file contains parameters that are available in a helicopter’s health usage monitoring system (HUMS)[14]. The dataset has 17 inputs and 9 outputs and consists of 4745 training patterns. We train an MLP with 15 hidden units for 200 iterations. Figures 6 and 7 shows that OWO-ONT trains much better than LM. By adjusting the density of the transform matrix, we are able to adjust the convergence rate and computational cost of MLP training. Table I shows that the training and validation error for ONT is significantly better than that of LM.
In this paper, we have introduced the concept of partial affine invariance to neural networks. We have developed a partially affine invariant two-step algorithm denoted as OWO-ONT that alternately improves input weights and then output weights. OWO-ONT is intermediate in complexity between BP and LM, but performs about as well as LM. By adjusting the sparsity of the transform matrix, we are able to adjust the convergence rate and computational cost of MLP training.

## X. Conclusions

In this paper, we have introduced the concept of partial affine invariance to neural networks. We have developed a partially affine invariant two-step algorithm denoted as OWO-ONT that alternately improves input weights and then output weights. OWO-ONT is intermediate in complexity between BP and LM, but performs about as well as LM. By adjusting

### Table I

<table>
<thead>
<tr>
<th></th>
<th>OWO-ONT</th>
<th>CG</th>
<th>LM</th>
<th>MOLF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$E_t$</td>
<td>$E_v$</td>
<td>$E_t$</td>
<td>$E_v$</td>
</tr>
<tr>
<td>Remote Sensing</td>
<td>0.1299</td>
<td>0.1406</td>
<td>0.288</td>
<td>0.3005</td>
</tr>
<tr>
<td>Oh7</td>
<td>1.2568</td>
<td>1.4324</td>
<td>2.3303</td>
<td>2.3566</td>
</tr>
<tr>
<td>Prognostics</td>
<td>$1.04\times10^7$</td>
<td>$1.37\times10^7$</td>
<td>$1.1618\times10^8$</td>
<td>$1.1937\times10^8$</td>
</tr>
</tbody>
</table>

The figures show the training error and computational cost for the Prognostics dataset.