Sensitivity of Nonlinear Network Training to Affine Transformed Inputs

Changhua Yu, Michael T. Manry, Pramod Lakshmi Narasimha
Department of Electrical Engineering
University of Texas at Arlington, TX 76019
changhua_yu@uta.edu, manry@uta.edu, l_pramod@uta.edu

Abstract

In this paper, the effects of nonsingular affine transforms on various nonlinear network training algorithms are analyzed. It is shown that gradient related methods, are quite sensitive to an input affine transform, while Newton related methods are invariant. These results give a connection between pre-processing techniques and weight initialization methods. They also explain the advantages of Newton related methods over other algorithms. Numerical results validate the theoretical analyses.

Introduction

Nonlinear networks, such as multi-layer perceptron (MLP) and radial basis function (RBF) networks, are popular in the signal processing and pattern recognition area. The networks' parameters are adjusted to best approximate the underlying relationship between input and output training data (Vapnik 1995). Neural net training algorithms are time consuming, in part due to ill-conditioning problems (Saarinen, Bramley, & Cybenko 1993). Pre-processing techniques, such as input feature-decorrelation , whitening transform (LeCun *et al.* 1998), (Brause & Rippl 1998) are suggested to alleviate ill-conditioning and thus accelerate the network training procedure ($\check{S}$. Raudys 2001). Other input transforms, including input re-scaling (Rigler, Irvine, & Vogl 1991), unbiasing and normalization, are also widely used to equalize the influence of the input features. In addition, some researchers think of the MLP as a nonlinear adaptive filter (Haykin 1996). Linear pre-processing techniques, such as noise cancellation, can improve the performance of this non-linear filter.

Although widely used, the effects of these linear transforms on MLP training algorithms haven't been analyzed in detail. Interesting issues, such as (1)whether these pre-processing techniques have same effects on different training algorithms, (2) whether the benefits of these pre-processing can be duplicated or cancelled out by other strategies, i.e., advanced weight initialization method, still need further research. In a previous paper (Yu, Manry, & Li 2004), the effects of an input orthogonal transform on the conjugate gradient algorithm was analyzed using the concept of equivalent states. We show that the effect of input orthogonal transform can be absorbed by proper weight initialization strategy. Because all linear transforms can be expressed as an affine transform, in this paper, we analyze the influence of the more general affine transform on typical training algorithms.

First, the conventional affine transform is described. Second, typical training algorithms are briefly reviewed. Then, the sensitivity of various training algorithms to input affine transforms are analyzed. Numerical simulations are presented to verify the theoretical results.

General Affine Transform

Suggested pre-processing techniques, such as feature decorrelation, whitening, input unbiasing and normalization can all be put into the form of nonsingular affine transform:

$$\mathbf{z}_p^T = \mathbf{A}\mathbf{x}_p^T + \mathbf{b} \tag{1}$$

where $\mathbf{x}_p$ is the original pth input feature vector, $\mathbf{z}_p$ is the affine transformed input, and $\mathbf{b} = [\ b_1 \quad b_2 \quad \cdots \quad b_N\ ]^T$.

For example, unbiasing the input features is modelled as

$$\mathbf{z}_p^T = \mathbf{x}_p^T - \mathbf{m}_x^T \tag{2}$$

An affine transform of particular interest in this paper can be expressed as a linear transform by using extended input vectors as:

$$\mathbf{Z}_p^T = \mathbf{A}_e\mathbf{X}_p^T \tag{3}$$

where $\mathbf{Z}_p = [z_{p1} \cdots z_{pN}, 1]$, $\mathbf{X}_p = [x_{p1} \cdots x_{pN}, 1]$ and

$$\mathbf{A}_e = \begin{bmatrix} a_{11} & \cdots & a_{1N} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{N1} & \cdots & a_{NN} & b_N \\ 0 & \cdots & 0 & 1 \end{bmatrix}, \tag{4}$$

Conventional Training Algorithms

Generally, training algorithms for nonlinear networks can be classified into three categories: gradient descent methods, conjugate gradient methods, and Newton related methods (Haykin 1999), (Møller 1997). In the following, we give a brief review for each method.

*This work was supported by the Advanced Technology Program of the state of Texas, under grant number 003656-0129-2001.

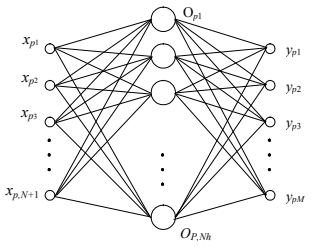


Figure 1: Topology of a three-layer MLP

Notation

In this paper, we investigate a three-layer fully connected MLP, as shown in figure 1. The training data consists of N_v training patterns $\{(\mathbf{x}_p, \mathbf{t}_p)\}$, where $\mathbf{x}_p$ and the pth desired output vector $\mathbf{t}_p$ have dimensions N and M, respectively. Thresholds in the hidden and output layers are handled by letting $x_{p,(N+1)} = 1$.

For the jth hidden unit, the net function net_{pj} and the output activation O_{pj} for the pth training pattern are

$$\begin{aligned} net_{pj} &= \sum_{n=1}^{N+1} w_{hi}(j,n) \cdot x_{pn} \\ O_{pj} &= f(net_{pj}) \qquad 1 \le j \le N_h \end{aligned} \tag{5}$$

where x_{pn} denotes the nth element of $\mathbf{x_p}$, $w_{hi}(j,n)$ denotes the weight connecting the nth input to the jth hidden unit and N_h denotes the number of hidden units. The activation function f is sigmoidal

$$f(net_{pj}) = \frac{1}{1+e^{-net_{pj}}} \tag{6}$$

The kth output y_{pk} for the pth training pattern is

$$y_{pk} = \sum_{n=1}^{N+1} w_{oi}(k,n) \cdot x_{pn} + \sum_{j=1}^{N_h} w_{oh}(k,j) \cdot O_{pj} \tag{7}$$

$k = 1, \ldots, M$, where $w_{oi}(k,n)$ denotes the weight connecting the nth input node to the kth output unit. The network's training procedure is to minimize the following global *Mean Square Error* (MSE):

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}[t_{pm}-y_{pm}]^2 = \frac{1}{N_v}\sum_{p=1}^{N_v}(\mathbf{t}_p-\mathbf{y}_p)(\mathbf{t}_p-\mathbf{y}_p)^T \tag{8}$$

where t_{pm} denotes the mth element of $\mathbf{t}_p$. Many methods have been proposed to solve this optimization problem.

Gradient Descent Method

The idea of gradient descent method is to minimize the linear approximation of MSE in (8):

$$E(\mathbf{w}+\Delta\mathbf{w}) \approx E(\mathbf{w}) + \Delta\mathbf{w}^T\frac{\partial E}{\partial \mathbf{w}} = E(\mathbf{w}) + \Delta\mathbf{w}^T\mathbf{g} \tag{9}$$

where $\mathbf{g}$ is the gradient vector of $E(\mathbf{w})$ evaluated at current $\mathbf{w}$. The weight updating strategy is:

$$\Delta\mathbf{w} = -Z \cdot \mathbf{g} \tag{10}$$

where the learning factor Z is usually set to a small positive constant or determined by an adaptive or optimal method(Magoulas, Vrahatis, & Androulakis 1999). The *back propagation* (BP) algorithm is actually a gradient descent method (Haykin 1999), (Rumelhart, Hiton, & Williams 1986). When using an optimal learning factor, the gradient descent method is also known as steepest descent (Fletcher 1987).

Conjugate Gradient Method

People have looked for some compromise between slow converging gradient methods and computationally expensive Newton's methods (Møller 1997). *Conjugate gradient* (CG) is such an intermediate method.

The basic idea of CG is to search the minimum of the error function in conjugate directions (Fletcher 1987). In CG, the directions $\mathbf{P}(i)$ for the ith searching iteration and $\mathbf{P}(j)$ for the jth iteration have to be conjugate with respect to the Hessian matrix:

$$\mathbf{P}^T(i)\mathbf{H}\mathbf{P}(j) = 0 \qquad \text{for } i \ne j \tag{11}$$

The conjugate directions can be constructed by (Fletcher 1987):

$$\mathbf{P}(k) = -\mathbf{g}(k) + B_1\mathbf{P}(k-1) \tag{12}$$

Here, $\mathbf{g}(k)$ represents the gradient of E with respect to corresponding weight and threshold for the kth searching iteration. Initial vectors are $\mathbf{g}(0) = \mathbf{0}$, $\mathbf{P}(0) = \mathbf{0}$. B_1 in Eq. (12) is the ratio of the gradient energies between the current iteration and the previous iteration. The corresponding weight changes in CG are:

$$\Delta\mathbf{w} = B_2 \cdot \mathbf{P}(k) \tag{13}$$

where B_2 is an optimal learning factor.

Newton's Method

The goal in Newton's method is to minimize the quadratic approximation of the error function $E(\mathbf{w})$ around the current weights $\mathbf{w}$:

$$E(\mathbf{w}+\Delta\mathbf{w}) \approx E(\mathbf{w}) + \Delta\mathbf{w}^T\mathbf{g} + \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} \tag{14}$$

Here, the matrix $\mathbf{H}$ is called the Hessian matrix, which is the second order partial derivatives of E with respect to $\mathbf{w}$. Eq. (14) is minimized when

$$\Delta\mathbf{w} = -\mathbf{H}^{-1}\mathbf{g} \tag{15}$$

For Newton's method to work, the Hessian matrix $\mathbf{H}$ has to be positive definite, which is not guaranteed in most situations. Many modified Newton algorithms have been developed to ensure a positive definite Hessian matrix (Battiti 1992).

Sensitivity of Training to Affine Transform

For affine transforms, the requirements to ensure equivalent states (Yu, Manry, & Li 2004) in the network with original inputs and the network with transformed inputs are:

$$\mathbf{w}_{hi} = \mathbf{u}_{hi}\mathbf{A_e} \tag{16}$$

$$\mathbf{w}_{oh} = \mathbf{u}_{oh}, \quad \mathbf{w}_{oi} = \mathbf{u}_{oi}\mathbf{A_e} \tag{17}$$

where $\mathbf{u}$ denote the corresponding weights and thresholds in the transformed network.

Sensitivity of CG to Affine Transform

In CG, for the kth training iteration, the weight updating law in the transformed network is:

$$\mathbf{u}_{hi} \leftarrow \mathbf{u}_{hi} + B_2 \cdot \mathbf{P}_{dhi}(k), \qquad \mathbf{u}_{oi} \leftarrow \mathbf{u}_{oi} + B_2 \cdot \mathbf{P}_{doi}(k)$$
$$\mathbf{u}_{oh} \leftarrow \mathbf{u}_{oh} + B_2 \cdot \mathbf{P}_{doh}(k) \tag{18}$$

The weights and thresholds in the original network are updated in the same way. If the two networks have the same learning factors B_2, the conjugate directions $\mathbf{P}$ must satisfy following conditions to ensure equivalent states after weights modification:

$$\mathbf{P}_{hi} = \mathbf{P}_{dhi}\mathbf{A_e}, \quad \mathbf{P}_{oh} = \mathbf{P}_{doh}, \quad \mathbf{P}_{oi} = \mathbf{P}_{doi}\mathbf{A_e} \tag{19}$$

The gradients in the transformed network can be found:

$$\mathbf{g}_{doi} = \frac{\partial E}{\partial \mathbf{u}_{oi}} = \begin{bmatrix} \frac{\partial E}{\partial u_{oi}(1,1)} & \cdots & \frac{\partial E}{\partial u_{oi}(1,N+1)} \\ \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial u_{oi}(M,1)} & \cdots & \frac{\partial E}{\partial u_{oi}(M,N+1)} \end{bmatrix}$$
$$= \frac{-2}{N_v}\sum_{p=1}^{N_v} \begin{bmatrix} e_{dp1} \\ \vdots \\ e_{dpM} \end{bmatrix} \mathbf{Z}_p = \frac{-2}{N_v}\left(\sum_{p=1}^{N_v} \mathbf{e}_{dp}\mathbf{X}_p\right)\mathbf{A}_e^T \tag{20}$$

where $e_{dpm} = t_{pm} - y_{dpm} \qquad m = 1, \cdots M.$

$$\mathbf{g}_{dhi} = \frac{\partial E}{\partial \mathbf{u}_{hi}} = \begin{bmatrix} \frac{\partial E}{\partial u_{hi}(1,1)} & \cdots & \frac{\partial E}{\partial u_{hi}(1,N)} \\ \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial u_{hi}(N_h,1)} & \cdots & \frac{\partial E}{\partial u_{hi}(N_h,N)} \end{bmatrix}$$
$$= \frac{-2}{N_v}\sum_{p=1}^{N_v} \begin{bmatrix} \delta_{dp1} \\ \vdots \\ \delta_{dpN_h} \end{bmatrix} \mathbf{Z}_p = \frac{-2}{N_v}\left(\sum_{p=1}^{N_v} \boldsymbol{\delta}_{dp}\mathbf{X}_p\right)\mathbf{A}_e^T \tag{21}$$

and $\mathbf{g}_{doh}$ is:

$$\mathbf{g}_{doh} = \frac{-2}{N_v}\sum_{p=1}^{N_v} \mathbf{e}_{dp}\mathbf{O}_{dp} \tag{22}$$

In the original network, the matrices are:

$$\mathbf{g}_{oi} = \frac{-2}{N_v}\sum_{p=1}^{N_v} \mathbf{e}_p\mathbf{X}_p, \qquad \mathbf{g}_{hi} = \frac{-2}{N_v}\sum_{p=1}^{N_v} \boldsymbol{\delta}_p\mathbf{X}_p$$
$$\mathbf{g}_{oh} = \frac{-2}{N_v}\sum_{p=1}^{N_v} \mathbf{e}_p\mathbf{O}_p \tag{23}$$

From Eq. (12), after the first training iteration in CG, the conjugate directions are:

$$\mathbf{P}(1) = -\mathbf{g}(1) \tag{24}$$

Assuming the two networks start from same initial states, for the first training iteration, we have from (20) that:

$$\mathbf{P}_{doi} = -\mathbf{g}_{doi} = -\mathbf{g}_{oi}\mathbf{A}_e^T = \mathbf{P}_{oi}\mathbf{A}_e^T \tag{25}$$

As $\mathbf{A}_e$ is not orthogonal, condition (19) can't be satisfied. So the training curves in the two networks diverge from the very beginning. B_1 and B_2 in the two networks won't be the same either. Therefore the CG algorithm is sensitive to input affine transforms.

Affine Transforms and OWO-BP

In output weight optimization (OWO), which is a component of several training algorithms (Chen, Manry, & Chandrasekaran 1999), we solve linear equations for MLP output weights. In the following, we will use $\mathbf{W_o} = [\mathbf{w_{oi}} \quad \mathbf{w_{oh}}]$ and an augmented input vector:

$$\hat{\mathbf{O}}_p = [\ x_{p1} \quad \cdots \quad x_{pN} \quad 1 \quad O_{p1} \quad \cdots \quad O_{pN_h}\ ] \tag{26}$$

For the three-layer network in figure 1, the output weights can be found by solving linear equations, which result when gradients of E with respect to the output weights are set to zero. Using equation (7) and (8), we find the gradients of E with respect to the output weights as

$$\begin{aligned} \hat{g}_o(m,j) &= -2 \cdot \tfrac{1}{N_v}\sum_{p=1}^{N_v}\left[t_{pm} - \sum_{i=1}^{L} w_o(m,i)\hat{O}_{pi}\right]\cdot \hat{O}_{pj} \\ &= -2\left[C(m,j) - \sum_{i=1}^{L} w_o(m,i)R(i,j)\right] \end{aligned} \tag{27}$$

where the autocorrelation matrix $\mathbf{R}$ has the elements:

$$R(i,j) = \frac{1}{N_v}\sum_p \hat{O}_{pi}\hat{O}_{pj} \tag{28}$$

the cross-correlation matrix $\mathbf{C}$ has the elements:

$$C(m,j) = \frac{1}{N_v}\sum_p t_{pm}\hat{O}_{pj} \tag{29}$$

Setting $\hat{g}_o(m,j)$ to zero, the jth equation in the mth set of equations is

$$\sum_{i=1}^{L} w_o(m,i)R(i,j) = C(m,j) \tag{30}$$

So the OWO procedure in the two networks means to solve

$$\mathbf{W}_o\mathbf{R}_x = \mathbf{C}_x \quad \mathbf{U}_o\mathbf{R}_z = \mathbf{C}_z \tag{31}$$

If before OWO the two networks have equivalent states, we have

$$\hat{\mathbf{O}}_{dp}^T = \begin{bmatrix} \mathbf{A}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{N_h \times N_h} \end{bmatrix} \hat{\mathbf{O}}_{\mathbf{p}}^T \tag{32}$$

And in the original network,

$$\mathbf{R}_x = \frac{1}{N_v}\sum_{p=1}^{N_v} \hat{\mathbf{O}}_p^T\hat{\mathbf{O}}_p \quad \mathbf{C}_x = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{t}_p^T\hat{\mathbf{O}}_p \tag{33}$$

For the network with transformed inputs, these matrices are:

$$\begin{aligned} \mathbf{R}_z &= \tfrac{1}{N_v}\sum_{p=1}^{N_v} \hat{\mathbf{O}}_{dp}^T\hat{\mathbf{O}}_{dp} \\ &= \begin{bmatrix} \mathbf{A}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{R}_x \begin{bmatrix} \mathbf{A}_e^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{aligned} \tag{34}$$

$$\mathbf{C}_z = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{t}_p^T\hat{\mathbf{O}}_{dp} = \mathbf{C}_x \begin{bmatrix} \mathbf{A}_e^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{35}$$

Invariance of OWO to Nonsingular Affine Transform
In the following lemma, we describe the effects of affine transforms on the OWO procedure.

Lemma 1 *For any nonsingular matrix $\mathbf{A}_e$, if output weights satisfy the conditions of Eq. (17), then after the OWO procedure it is still valid.*

Proof. As the two networks start from equivalent states, $\mathbf{U}_o\mathbf{R}_z = \mathbf{C}_z$ in Eq. (31) can be rewritten as:

$$\mathbf{U}_o \begin{bmatrix} \mathbf{A}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{R}_x \begin{bmatrix} \mathbf{A}_e^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \mathbf{C}_x \begin{bmatrix} \mathbf{A}_e^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (36)$$

it can be simplified further as:

$$\mathbf{U}_o \begin{bmatrix} \mathbf{A}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{R}_x = \mathbf{C}_x \quad (37)$$

Comparing it with $\mathbf{W}_o\mathbf{R}_x = \mathbf{C}_x$, so we have

$$\mathbf{W}_o = \mathbf{U}_o \begin{bmatrix} \mathbf{A}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (38)$$

which means condition (17) is still valid.

The Sensitivity of BP to Affine Transform
For BP procedure, in order to keep equivalent states in the two networks, the gradients must satisfy:

$$\mathbf{g}_{hi} = \mathbf{g}_{dhi}\mathbf{A}_e \quad (39)$$

As we know, when B_1 in (12) is set to zero, CG degrades into BP algorithm. Thus the analyses for CG in is also valid for BP. Condition (39) won't be satisfied after BP. So BP is also sensitive to the input affine transforms.

In OWO-BP training (Chen, Manry, & Chandrasekaran 1999), we alternately use OWO to find output weights and use BP to modify hidden weights. Because of the sensitivity of BP to affine transform, OWO-BP has a similar sensitivity.

Effects on Newton Related Method

Generally, it is much more difficult to find the hidden weights in neural networks. Due to its high convergence rate, Newton's method is preferred for small scale problems (Battiti 1992). The proposed hidden weight optimization (HWO) in (Chen, Manry, & Chandrasekaran 1999) can be considered as a Newton related method.

To ensure equivalent states in the two networks, the hidden weights and thresholds must satisfy (16) and (17), and correspondingly, the hidden weight changes in the two networks must satisfy:

$$\Delta\mathbf{w}_{hi} = \Delta\mathbf{u}_{hi}\mathbf{A_e} \quad (40)$$

Rearrange $\Delta\mathbf{w}_{hi}$ in the form of a column vector:

$$\Delta\mathbf{W} = [\ \Delta w_{hi}(1,1) \quad \cdots \quad \Delta w_{hi}(N_h, N+1)\ ]^T \quad (41)$$

and the corresponding column vector in the transformed network is denoted as $\Delta\mathbf{U}$. So condition (40) is equivalent to:

$$\Delta\mathbf{W} = \tilde{\mathbf{A}}^T\Delta\mathbf{U} \quad (42)$$

where

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_e & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_e & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A}_e \end{bmatrix}_{q\times q} \quad (43)$$

with $q = N_h(N+1)$. The hidden weights are updated by the law of (15). The Hessian matrix of the original network is:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_{hi}^2(1,1)} & \cdots & \frac{\partial^2 E}{\partial w_{hi}(1,1)\partial w_{hi}(N_h,N+1)} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_{hi}(N_h,N+1)\partial w_{hi}(1,1)} & \cdots & \frac{\partial^2 E}{\partial w_{hi}^2(N_h,N+1)} \end{bmatrix} \quad (44)$$

The gradient vector used in the Newton method is:

$$\begin{aligned} \mathbf{g} &= \left[\ \frac{\partial E}{\partial w_{hi}(1,1)} \quad \cdots \quad \frac{\partial E}{\partial w_{hi}(1,N+1)} \quad \cdots \quad \frac{\partial E}{\partial w_{hi}(N_h,N+1)}\ \right]^T \\ &= \frac{-2}{N_v}\sum_{p=1}^{N_v}\left[\ \delta_{p1}x_{p1} \quad \cdots \quad \delta_{p1}x_{p(N+1)} \quad \cdots \quad \delta_{pN_h}x_{p(N+1)}\ \right]^T \\ &= \frac{-2}{N_v}\sum_{p=1}^{N_v}\left[\ \mathbf{X}_p\cdot\delta_{p1} \quad \cdots \quad \mathbf{X}_p\cdot\delta_{pN_h}\ \right]^T = \frac{-2}{N_v}\sum_p \mathbf{\Lambda}_{xp}\tilde{\boldsymbol{\delta}}_p \end{aligned} \quad (45)$$

where $\mathbf{\Lambda}_{xp} = diag\{x_{p1}\cdots x_{p(N+1)}\cdots x_{p1}\cdots x_{p(N+1)}\}$ is a $q \times q$ matrix, and

$$\tilde{\boldsymbol{\delta}}_p = [\ \delta_{p1} \quad \cdots \quad \delta_{p1} \quad \cdots \quad \delta_{pN_h} \quad \cdots \quad \delta_{pN_h}\ ]^T \quad (46)$$

is a $q \times 1$ vector. Similarly, in the transformed network,

$$\begin{aligned} \mathbf{g}_d &= \left[\ \frac{\partial E}{\partial u_{hi}(1,1)} \quad \cdots \quad \frac{\partial E}{\partial u_{hi}(N_h,N+1)}\ \right]^T \\ &= \frac{-2}{N_v}\sum_{p=1}^{N_v}\left[\ \delta_{dp1}z_{p1} \quad \cdots \quad \delta_{dpN_h}z_{p(N+1)}\ \right]^T \\ &= \frac{-2}{N_v}\sum_{p=1}^{N_v}\left[\ \mathbf{Z}_p\cdot\delta_{dp1} \quad \cdots \quad \mathbf{Z}_p\cdot\delta_{dpN_h}\ \right]^T \\ &= \frac{-2}{N_v}\sum_p \mathbf{\Lambda}_{xp}\tilde{\boldsymbol{\delta}}_{dp} = \frac{-2}{N_v}\tilde{\mathbf{A}}\sum_p \mathbf{\Lambda}_{xp}\tilde{\boldsymbol{\delta}}_{dp} \end{aligned} \quad (47)$$

with

$$\tilde{\boldsymbol{\delta}}_{dp} = [\ \delta_{dp1} \quad \cdots \quad \delta_{dp1} \quad \cdots \quad \delta_{dpN_h} \quad \cdots \quad \delta_{dpN_h}\ ]^T \quad (48)$$

From Eq. (44) and (45), the $q \times q$ Hessian matrix in the original network is

$$\mathbf{H} = \frac{-2}{N_v}\sum_p \mathbf{\Lambda}_{xp} \begin{bmatrix} \frac{\partial\delta_{p1}}{\partial w_{hi}(1,1)} & \cdots & \frac{\partial\delta_{p1}}{\partial w_{hi}(N_h,N+1)} \\ \vdots & \vdots & \vdots \\ \frac{\partial\delta_{p1}}{\partial w_{hi}(1,1)} & \cdots & \frac{\partial\delta_{p1}}{\partial w_{hi}(N_h,N+1)} \\ \vdots & \cdots & \vdots \\ \frac{\partial\delta_{pN_h}}{\partial w_{hi}(1,1)} & \cdots & \frac{\partial\delta_{pN_h}}{\partial w_{hi}(N_h,N+1)} \\ \vdots & \vdots & \vdots \\ \frac{\partial\delta_{pN_h}}{\partial w_{hi}(1,1)} & \cdots & \frac{\partial\delta_{pN_h}}{\partial w_{hi}(N_h,N+1)} \end{bmatrix} \quad (49)$$

In the rightmost matrix of (49), every $(N+1)$ rows from the $[(i-1)(N+1)+1]$th row to the $i\cdot(N+1)$th row are exactly same, with $i=1,\ldots,N_h$. The elements

$$\frac{\partial\delta_{pi}}{\partial w_{hi}(j,n)} = -\gamma_{pij}x_{pn} \tag{50}$$

where $i,j=1,\ldots,N_h$, $i\neq j$, $n=1,\ldots,(N+1)$, and

$$\gamma_{pij} = \left[\sum_{m=1}^{M} w_{oh}(m,j)w_{oh}(m,i)f'_{pj}f'_{pi}\right] \tag{51}$$

It is easy to verify that $\gamma_{pij}=\gamma_{pji}$. For the case $i=j$,

$$\frac{\partial\delta_{pi}}{\partial w_{hi}(i,n)} = -\gamma_{pii}x_{pn} \tag{52}$$

with

$$\gamma_{pii} = \left[\sum_{m=1}^{M} w_{oh}^2(m,i)(f'_{pi})^2 - [t_{pm}-y_{pm}]\,w_{oh}(m,i)f''_{pi}\right] \tag{53}$$

Here, f'_{pi}, f''_{pi} are shorthand for the first order and second order derivatives of the sigmoid function $f(net_{pi})$. So, we have

$$\mathbf{H} = \frac{2}{N_v}\sum_p \mathbf{\Lambda}_{xp}\mathbf{\Psi}_p \tag{54}$$

where

$$\mathbf{\Psi}_p = \begin{bmatrix} \gamma_{p11}\mathbf{X} & \cdots & \gamma_{p1N_h}\mathbf{X} \\ \vdots & \cdots & \vdots \\ \gamma_{pN_h1}\mathbf{X} & \cdots & \gamma_{pN_hN_h}\mathbf{X} \end{bmatrix} \tag{55}$$

where every row in the $q\times q$ matrix $\mathbf{X}$ is just the vector $\mathbf{X}_p$. Similarly, for the transformed network, the corresponding Hessian matrix is:

$$\mathbf{H}_d = \frac{2}{N_v}\tilde{\mathbf{A}}\sum_p \mathbf{\Lambda}_{xp}\begin{bmatrix} \gamma_{dp11}\mathbf{Z} & \cdots & \gamma_{dp1N_h}\mathbf{Z} \\ \vdots & \cdots & \vdots \\ \gamma_{dpN_h1}\mathbf{Z} & \cdots & \gamma_{dpN_hN_h}\mathbf{Z} \end{bmatrix} \tag{56}$$

where every row in the $q\times q$ matrix $\mathbf{Z}$ is just the vector $\mathbf{Z}_p$. Using the fact that

$$\mathbf{Z} = \mathbf{X}\mathbf{A}_e^T \tag{57}$$

Eq. (56) becomes

$$\mathbf{H}_d = \frac{2}{N_v}\tilde{\mathbf{A}}\left\{\sum_p \mathbf{\Lambda}_{xp}\mathbf{\Psi}_{dp}\right\}\tilde{\mathbf{A}}^T \tag{58}$$

with

$$\mathbf{\Psi}_{dp} = \begin{bmatrix} \gamma_{dp11}\mathbf{Z} & \cdots & \gamma_{dp1N_h}\mathbf{Z} \\ \vdots & \cdots & \vdots \\ \gamma_{dpN_h1}\mathbf{Z} & \cdots & \gamma_{dpN_hN_h}\mathbf{Z} \end{bmatrix} \tag{59}$$

Theorem 1 *If two networks are initially equivalent, and their input vectors are related via a nonsingular affine transform, the networks are still equivalent after Newton training.*

Proof. When the two networks are in equivalent states, conditions (16) is satisfied and we have:

$$\tilde{\boldsymbol{\delta}}_p = \tilde{\boldsymbol{\delta}}_{dp}, \qquad \mathbf{\Psi}_p = \mathbf{\Psi}_{dp} \tag{60}$$

so the change of the hidden weights and thresholds in the transformed network are

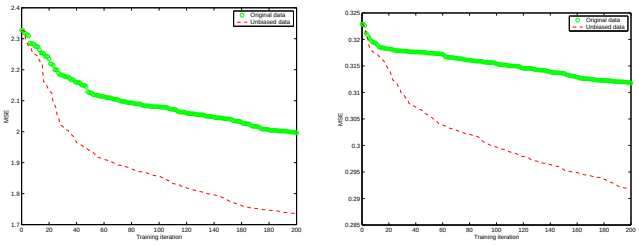
$$\begin{aligned} \Delta\mathbf{U} &= -\mathbf{H}_d^{-1}\mathbf{g}_d = \left(\tilde{\mathbf{A}}^T\right)^{-1}\left\{\sum_p \mathbf{\Lambda}_{xp}\mathbf{\Psi}_{dp}\right\}^{-1} \\ &\quad \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{A}}\left\{\sum_p \mathbf{\Lambda}_{xp}\tilde{\boldsymbol{\delta}}_{dp}\right\} \\ &= \left(\tilde{\mathbf{A}}^T\right)^{-1}\left\{\sum_p \mathbf{\Lambda}_{xp}\mathbf{\Psi}_p\right\}^{-1}\left\{\sum_p \mathbf{\Lambda}_{xp}\tilde{\boldsymbol{\delta}}_p\right\} \\ &= -\left(\tilde{\mathbf{A}}^T\right)^{-1}\mathbf{H}^{-1}\mathbf{g} = \left(\tilde{\mathbf{A}}^T\right)^{-1}\Delta\mathbf{W} \end{aligned} \tag{61}$$

So the required condition (42) would be satisfied after the Newton training procedure, which means that the conditions (16) would be valid again.

Numerical Results

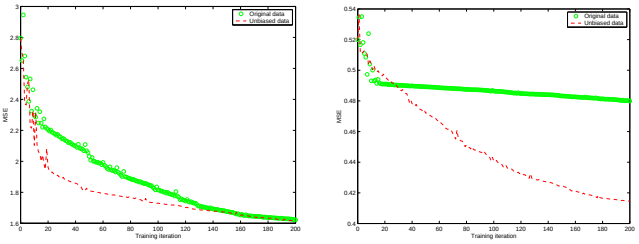
In the experiments, we use the unbiasing procedure as an example of the affine transform, and verify the theoretical results for two remote sensing data sets.

Inputs for training data set *Oh7.tra* are VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg (Oh, Sarabandi, & Ulaby 1992). The corresponding desired outputs are $\Theta=\{\sigma,l,m_v\}^T$, where σ is the rms surface height; l is the surface correlation length; m_v is the volumetric soil moisture content in g/cm^3. There are 20 inputs, 3 outputs, 10453 training patterns. We use 20 hidden units. Training data set *Twod.tra* contains simulated data based on models from backscattering measurements (Dawson, Fung, & Manry 1993). This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an in-homogeneous irregular layer above a homogeneous half space from back scattering measurements. The data set has 8 inputs, 7 outputs, and 1768 patterns. There are 10 hidden units.



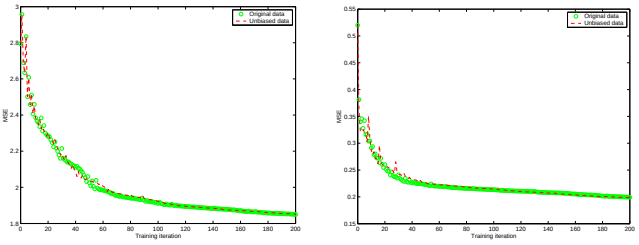
(a) oh7.dat (b) twod.tra

Figure 2: The sensitivity of CG to affine transforms



(a) oh7.dat (b) twod.tra

Figure 3: The effect of affine transform on OWO-BP



(a) oh7.dat (b) twod.tra

Figure 4: The invariance of OWO-HWO to affine transforms

For both data sets, the original and transformed networks start from equivalent states and are trained for 200 iterations. From fig. 2 and fig. 3, we can see that CG and BP are quite sensitive to the affine transform. Even though the two networks start from same initial states, the difference of their training curves are quite large. For the Newton related OWO-HWO method, the training curves for the two networks in fig. 4 are almost the same. The slight differences are caused by the finite numerical precision problem.

Summary and Discussion

In this paper, we analyze the effects of the affine transform on different training algorithms. Using input unbiasing as an example, we show that CG and BP are quite sensitive to input affine transforms. As a result, the input unbiasing strategy helps to accelerate the training of these gradient algorithms.

As for Newton methods, the network with original inputs and the network with transformed inputs have the same states during the training procedure as long as they start from equivalent states. This implies that OWO-HWO is more powerful than non-Newton algorithms when dealing with correlated/biased data. That is, in Newton methods, the benefits of feature de-correlation or unbiasing have already been realized by proper initial weights. This gives a connection between the pre-processing and the weight initialization procedure. However, this also indicates a trade-off. Sometimes the effects of pre-processing may be cancelled out by the initial weights. Naturally, further investigation can be done on whether there is a similar connection between the pre-processing and other techniques, for example, the learning factor setting. With theoretical analyses on these techniques, better neural network training strategy can be developed.

References

Battiti, R. 1992. First- and second-order methods for learning: between steepest descent and newton's method. *Neural Computation* 4(2):141–166.

Brause, R. W., and Rippl, M. 1998. Noise suppressing sensor encoding and neural signal orthonormalization. *IEEE Trans. Neural Networks* 9(4):613–628.

$\check{S}$. Raudys. 2001. *Statistical and Neural Classifiers: An Integrated Approach to Design*. Berlin, Germany: Springer-Verlag.

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.

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

Fletcher, R. 1987. *Practical Methods of Optimization*. Chichester, NY: John Wiley & Sons, second edition.

Haykin, S. 1996. *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice Hall, third edition.

Haykin, S. 1999. *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ: Prentice Hall, second edition.

LeCun, Y.; Bottou, L.; Orr, G. B.; and Muller, K. R. 1998. Efficient backprop. In Orr, G. B., and Muller, K. R., eds., *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer-Verlag.

Magoulas, G. D.; Vrahatis, M. N.; and Androulakis, G. S. 1999. Improving the convergence of the backpropagation algorithm using learning adaptation methods. *Neural Computation* 11(8):1769–1796.

Møller, M. 1997. *Efficient Training of Feed-forward Neural Networks*. Ph.D. Dissertation, Aarhus University, Denmark.

Oh, Y.; Sarabandi, K.; and Ulaby, F. T. 1992. An empirical model and an inversion technique for radar scattering from bare soil surfaces. *IEEE Trans. Geosci. Remote Sensing* 30:370–381.

Rigler, A. K.; Irvine, J. M.; and Vogl, T. P. 1991. Rescaling of variables in back propagation learning. *Neural Networks* 4:225–229.

Rumelhart, D. E.; Hiton, G. E.; and Williams, R. J. 1986. *Learning internal representation by error propagation*, volume 1 of *Parallel Distributed Processing*. Cambridge, MA: MIT Press.

Saarinen, S.; Bramley, R.; and Cybenko, G. 1993. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing* 14:693–714.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.

Yu, C.; Manry, M. T.; and Li, J. 2004. Invariance of mlp training to input feature de-correlation. In *Proceedings of the 17th International Conference of the Florida AI Research Society*, 682–687.