MULTISTEP SECOND ORDER TRAINING FOR THE MULTILAYER PERCEPTRON

by

MELVIN DELOYD ROBINSON

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2013

To my mother Evelyn, my wife Pamela, my late father Melvin, my grandmothers Elizabeth and Katherine, my grandfather Cornelius, all of my brothers, sisters, uncles and aunts, the remainder of my family and all of my friends who have supported me throughout this long hard slog.

ACKNOWLEDGEMENTS

I would like to thank my supervising professor, Dr. Michael Manry, for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. He made me a better student, researcher and man than I could have ever been had he not come into my life.

I would like to thank Dr. Ioannis Schizas, Dr. Wei-Jen Lee, Dr. Dan Popa and Dr. Daniel Levine for serving on my comprehensive and dissertation committees.

I would like to thank my wife, Pamela Robinson, for putting up with the countless chores that I did not do (mostly by design) in pursuit of my studies. She has been steadfast in her support and provided the love that I needed to pursue this degree.

I would like to thank two of my best friends Travis Askew and Michael Gilmore. Life gives one just a few true friends and I am very fortunate to have those two. You have endured my uncountably infinite number of aphorisms especially as I was "beginning to believe".

I would like to extend a special acknowledgement to Carla Thomas who put me on the right path in high school.

I would also like to thank the countless folks who had a small, but important hand in my life through the hard times. You know who you are: those that gave me a roof over my head when I could have been homeless, gave me food to eat when I was hungry, a job lead when I needed to work, advice when I was struggling through an issue and much much more. I appreciate you all and want to acknowledge your role in my life and my success.

I would like thank Donya Henry for her hard work in helping me through my coursework. I definitely could not have succeeded without her.

Finally, I would like to thank my mother, Evelyn Harris, who fostered that strong will and fighting spirit in me. This allowed me to never give up in pursuit of my education.

April 12, 2013

ABSTRACT

MULTISTEP SECOND ORDER TRAINING FOR THE MULTILAYER PERCEPTRON

MELVIN DELOYD ROBINSON, Ph.D.

The University of Texas at Arlington, 2013

Supervising Professor: Michael T. Manry

Training a feedforward multilayer perceptron (MLP) requires obtaining training data and solving a non-convex optimization problem to calculate the network's weights. Various problems can arise during training that ultimately can limit a MLP's usefulness such as slow convergence and high computational complexity. Additionally, when training one needs to have confidence that the chosen algorithm is working optimally for the chosen coordinate system.

We introduce novel second order training algorithms to overcome these difficulties. In the process, a piecewise affine model of the multilayer perceptron is introduced which shows that objective functions for training are poorly modeled by quadratic functions of network weights. One step and multistep second order training algorithms are derived which avoid the problems implied by the model.

The new second order algorithms are shown to have a form of affine invariance which ensures that they are optimal in the sense that they cannot be improved by affine transformation.

In simulation, their training and validation performance is comparable to Levenberg-Marquardt, yet they have the advantage of reduced computational complexity.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Foundational Material

Artificial Neural Networks were initially conceived as an attempt to determine a computational model of the human brain. McCulloch and Pitts[1] felt that the "all or none" characteristic of the human nervous sytem could be modeled by propositional logic. In 1949, Hebb[2], a neuropsychologist, proposed the concept of associative learning. Later, work by Rosenblatt[3] provided the scientific community with the perceptron and its adjustable weights. Later still, Werbos[4, 5] revolutionized neural network research with his invention of the backpropagation algorithm. Neural networks, in fact, have become state of the art tools in the vast and growing field of machine learning. As a result of their good performance and flexibility, neural networks have found application to a wide variety of fields.

Neural networks are usually employed to solve two particular types of engineering problems: regression and classification. In both problems, our goal is to approximate an unknown function which characterizes the relationship between the input and the output which can be expressed as $f\colon \mathbb{R}^N \to \mathbb{R}^M$. A neural network calculates a function $g\colon \mathbb{R}^N \to \mathbb{R}^M$ that is as "close" as possible in some sense to f. The function g is calculated through experimental data, which is mathematically modeled as $\{\mathbf{x}, \mathbf{t}\}$ pairs where $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{t} \in \mathbb{R}^M$.. The process of determining g or "learning" is called training. In general learning is the process of using a set of observations to uncover an underlying random process. When training neural networks this typically results in the need to solve an unconstrained optimization problem. The objective function in this problem is a measure of error between the desired output and a function of the network's weights. While the classical methods for solution of these types of problems have existed for centuries, numerical optimization as a field took off in the 1940's[6]. Applying these optimization techniques to a neural network presents its own challenges:

1. the problems are highly nonlinear
2. the objective function is non-convex
3. large numbers of features can result in large networks which render a problem computationally taxing
4. training algorithms can result in singular or ill-conditioned matrices

In batch learning the network's weights are updated after a pass through the entire training set[7]. A pass through the training set is often called an epoch. This is in contrast to stochastic learning, sometimes called online learning, where adjustments to the weights are made on a pattern by pattern basis. Each offers its own advantages and disadvantages, but the work done in this proposal will consider only batch supervised learning.

1.1.1 First Order Methods

First order training methods use first derivative information in minimizing the error function. Though first order methods are almost always guaranteed to converge, they can be very slow and depending on the application, these types of methods can be impractical. Backpropagation (BP), the most fundamental first order algorithm, is an application of classic steepest descent to train neural networks. The Conjugate Gradient algorithm (CG) is another classic first order method that has been used effectively to train a neural network[8] and should scale very well to large networks. When compared to BP, the CG produces is a vastly improved search direction and as a result, converges faster. It is widely known that while the gradient is the direction that the error function descends at the greatest rate, we are not guaranteed that it is the best direction towards the function's minimum. Complicating this problem is the fact that a neural network's error function is non-convex and iterative methods can get stuck in a local minimum.

1.1.2 Classic Second Order Methods

Second order methods hold the promise of faster convergence at a mild increase of computational cost. There have been many second order optimization techniques developed through the years the most prominent being the Levenberg-Marquardt method, Newton's method, and BFGS. All take different approaches to calculating the Hessian and all solve linear equations to determine a new descent direction. Convergence of classic second order optimization methods is strongly dependent on the distribution of the Hessian's eigenvalues[9].

1.2 Why use Neural Networks?

There are a myriad of applications that are using neural networks. As the field of machine learning grows so will the applications of neural networks.

1.2.1 Approximation properties

Neural networks offer several advantages over other learning machines. Their strength lies in their universal approximation properties. Hornik[10] and White[11] explicitly prove this characteristic. Neural networks are even able to approximate discontinuous functions [12]. Bounds for their approximation properties have even been calculated[13].

1.2.2 Bayes classification

Not only are neural networks' properties useful for universal approximation, but they are equally applicable in the classification problem. Ruck[14] proves that a well-designed neural network's output approaches the Bayes optimal discriminant function:

$$\underset{i}{\text{maximum}} \quad p(i|\mathbf{x}) \tag{1.1}$$

the class label, i, that maximizes the probability given the feature vector. Wan provides further material on this relationship[15]. Nonlinear processing improves a learning machine's ability to discern input/output relationships.

1.2.3 Applications

Neural networks work well for each of the major engineering efforts: function approximation and classification. The applications of neural networks are very broad. Examples are in the area of pattern recognition[16, 17, 18, 19], medical[20, 21, 22, 23, 24], remote sensing[25, 26, 27], image processing[28, 29, 30], power load forecasting[31, 32, 33] and even psychiatry [34]. There are numerous others.

Neural networks naturally lend themselves engineering applications that require nonlinear estimation[35, 36, 37, 38]. They are ubiquitious in classification and recognition applications[39, 40]. Neural networks are attractive in these applications because the exact relationship between the features that an engineer may deem important and the resulting output of interest is not exactly known and neural networks can be employed to learn this relationship through training.

1.3 Architecture and Notation

Our model of an MLP will contain N inputs, N_h hidden units and M outputs. The MLP topology used in this proposal is illustrated in Figure 1.1.

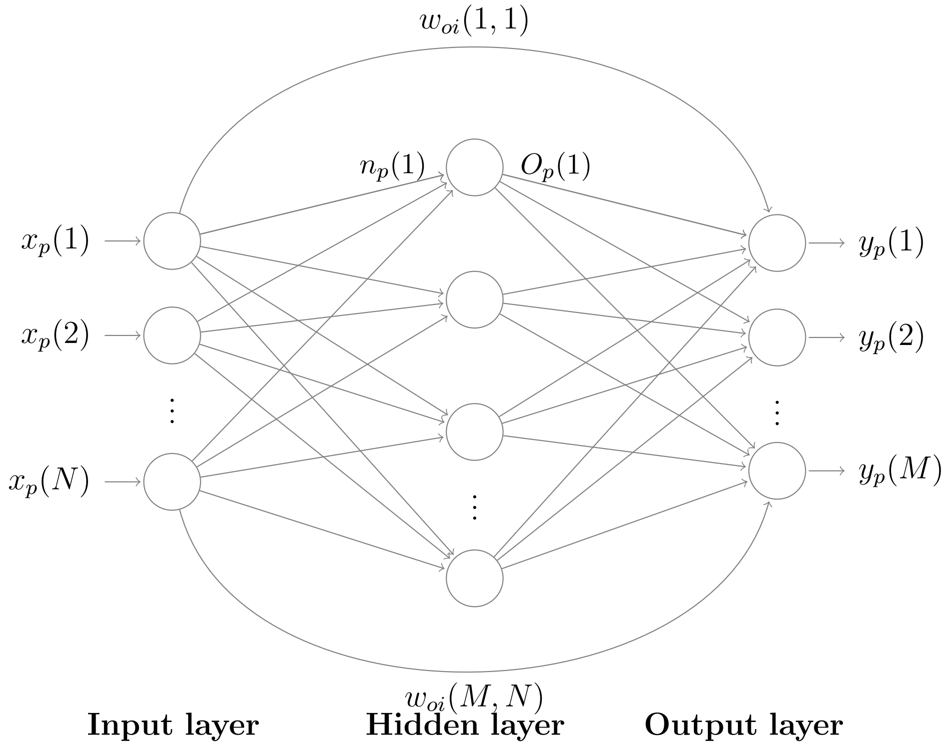

Figure 1.1. Illustration of a Multilayer Perceptron. Bypass weights exist from every node in the input layer to every node in the output layer..

As mentioned in Chapter 1, each line of a training file is processed as a pair of vectors $\{\mathbf{x}_p, \mathbf{t}_p\}$. Here, the input vectors are $\mathbf{x}_p \in \mathbb{R}^N$,and the desired output vectors are $\mathbf{t}_p \in \mathbb{R}^M$. Values of p range from 1 to N_v, where N_v is the total number of patterns in the training file. The values in $\mathbf{t}_p$ represent real data in a regression problem or contain class labels in a classification problem. 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 $x_p(N+1) = 1$. This covers the threshold and allows for the modeling of constant functions. For each training pattern, the net function vector $\mathbf{n}_p$ is given by

$$n_p(j) = \sum_{k=1}^{N+1} w(j,k)x_p(k) \tag{1.2}$$

or in matrix-vector notation

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

and the corresponding j^{th} hidden unit activation $O_p(j)$ is

$$O_p(j) = f(n_p(j)) \tag{1.3}$$

where $\mathbf{W}$ is an N_h by $N+1$ matrix of hidden unit weights and thresholds. N_h is the number of units in the hidden layer. The $(N+1)^{th}$ column of $\mathbf{W}$ accounts for the hidden unit threshold. For the logistic sigmoidal activation the output of the j^{th} hidden unit is given by

$$f(n_p(j)) = \frac{1}{1+e^{-n_p(j)}} \tag{1.4}$$

The predicted value of the i^{th} output for the p^{th} training pattern $y_p(i)$ is

$$y_p(i) = \sum_{j=1}^{N+1} w_{oi}(i,j)x_p(j) + \sum_{k=1}^{N_h} w_{oh}(i,k)O_p(k) \tag{1.5}$$

or in matrix-vector notation

$$\mathbf{y}_p = \mathbf{W}_{oi}\mathbf{x}_p + \mathbf{W}_{oh}\mathbf{O}_p \tag{1.6}$$

where $\mathbf{W}_{oi} \in \mathbb{R}^{M\times(N+1)}$, contains bypass weights in the first N columns and the output thresholds in the last or $(N+1)^{th}$ column. $\mathbf{W}_{oh} \in \mathbb{R}^{M\times N_h}$ contains weights from hidden units to the outputs. As defined by (1.3), $\mathbf{O}_p$ is a vector of length N_h. The error function used for our work is the MSE objective function

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - y_p(i)\right]^2 \tag{1.7}$$

where $t_p(i)$ denotes the i^{th} element of the p^{th} desired output vector and is obtained from the training data. When rewritten in vector notation the above becomes:

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} (\mathbf{t}_p - \mathbf{y}_p)^T (\mathbf{t}_p - \mathbf{y}_p)$$

Generally, the goal of training a neural network is to determine the weights so that the values of $\mathbf{y}_p$ are as close as possible to $\mathbf{t}_p$.

1.4 Dissertation Roadmap

In this dissertation, we outline a number of MLP training problems to overcome and design and implement training algorithms to overcome these problems. In Chapter 2, we review some popular one-step methods. Chapter 3 reviews some of the fundamentals of equivalent network theory and properties of affine invariance. Chapter 4 outlines some neural network training difficulties and proposes novel research to overcome these difficulties. Chapter 5 discusses the OWO-ONT algorithm. Chapter 6 discusses the assumptions inherent to Newton training and motivates multistep methods. Chapter 7 introduces OWO-Newton. Chapter 8 introduces the HOST algorithm. Finally, Chapter 9 shows some simulations using the new methods against the performance of reference methods. Finally, in Chapter 10 we present conclusions, summarize contributions and propose future work.

CHAPTER 2

REVIEW OF ONE-STEP TRAINING

This proposal focuses on the three layer fully connected multi-layer perceptron, or MLP, with linear activation functions in the output layer and sigmoidal activation functions in the hidden layer. These MLPs are usually referred to as single hidden layer MLPs. A fully connected MLP has weights between all nodes of the network. While there are many different possible topologies of the MLP, Cybenko and others show that a single hidden layer MLP's generality is sufficient[41, 42]. This chapter presents a review of the basic notation, processing and training methodologies of a MLP.

2.1 Output Weight Optimization

One method used to train and initialize neural networks is the Output Weight Optimization algorithm[43, 44, 45]. 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 diagonal elements of the error function

$$E = trace(\mathbf{W}_o\mathbf{R}\mathbf{W}_o^T - 2\mathbf{C}^T\mathbf{W}_o + \mathbf{E}_t) \tag{2.1}$$

$\mathbf{R}$ and $\mathbf{C}$ are matrices calculated as

$$\mathbf{R} = \frac{1}{N_v}\sum_{p=1}^{N_v}\mathbf{x}_{ap}\mathbf{x}_{ap}^T, \tag{2.2}$$

$$\mathbf{C} = \frac{1}{N_v}\sum_{p=1}^{N_v}\mathbf{x}_{ap}\mathbf{t}_p^T \tag{2.3}$$

$$\mathbf{E}_t = \frac{1}{N_v}\sum_{p=1}^{N_v}\mathbf{t}_p\mathbf{t}_p^T,$$

and

$$\mathbf{x}_{ap} = \begin{bmatrix} \mathbf{x}_p \\ \dots \\ \mathbf{O}_p \end{bmatrix}$$

Equation (2.1) is minimized by the solution to the linear equations

$$\mathbf{R}\mathbf{W}_o^T = \mathbf{C} \tag{2.4}$$

These M sets of equations in N_u unknowns 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 [46]. Because (2.1) is quadratic, OWO is merely Newton's algorithm for the output weights. A modern descendant of OWO is the Extreme Learning Machine (ELM) [47] training.

An interesting fact of OWO is its relationship to Newton's algorithm

Lemma 2.1. *OWO is Newton's algorithm for the output weights.*

Proof. We update the output weights as $\mathbf{W}_o \leftarrow \mathbf{W}_o + \mathbf{D}_o$, where $\mathbf{D}_o$ is $\begin{bmatrix}\mathbf{D}_{oi} & \vdots & \mathbf{D}_{oh}\end{bmatrix}$. We use Newton's algorithm to calculate $\mathbf{D}_o$. The Hessian and gradient of (2.1) are $2\mathbf{R}$ and $2\mathbf{R}\mathbf{W}_o^T - 2\mathbf{C}$ respectively. The weight update becomes

$$\begin{aligned}\mathbf{W}_o &\leftarrow \mathbf{W}_o - \mathbf{H}^{-1}\mathbf{g} \\ &= \mathbf{W}_o + (2\mathbf{R})^{-1}2(\mathbf{C} - \mathbf{R}\mathbf{W}_o^T) \\ &= \mathbf{W}_o + \mathbf{R}^{-1}\mathbf{C} - \mathbf{R}^{-1}\mathbf{R}\mathbf{W}_o^T \\ &= \mathbf{R}^{-1}\mathbf{C}\end{aligned} \tag{2.5}$$

This is the same solution to OWO given in (2.4). □

2.2 Backpropagation

As mentioned in Chapter 1, a major improvement in neural network research occurred when Rumelhart, Werbos and others discovered that the chain rule could be used to train a network's input weights. Here we briefly cover these concepts. Let N_w be the total number of network weights and $\mathbf{w}$ be a vector containing these weights.

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \frac{\partial E}{\partial \mathbf{w}_k} \tag{2.6}$$

For practical purposes (2.6) should include a line search and thus we have

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - z\frac{\partial E}{\partial \mathbf{w}_k} \tag{2.7}$$

where z is the result of solving the line search subproblem is discussed in the Appendix §A. BP generally converges very slowly, but despite this slowness, it remains one of the most popular neural network training algorithms. Hecht-Nielsen[48] and Werbos[49] provide a very good review of the algorithm.

```
Initialize $\mathbf{W}$,$\mathbf{W}_{oi}$,$\mathbf{W}_{oh}$
while stopping criterion not reached do
    Calculate network gradients
    Line Search: Choose $z$ to minimize $\frac{\partial E(\mathbf{w}_k + z\mathbf{g}_k)}{\partial \mathbf{w}_k}$
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + z\mathbf{g}_k$
    $k \leftarrow k + 1$
end while
```

Algorithm 2.1: Backpropagation Algorithm

2.3 Conjugate Gradient

The information in this section is taken from [50]. Conjugate Gradient or CG builds on the principles introduced in BP, but modifies $\mathbf{g} = -\frac{\partial E}{\partial \mathbf{w}}$ in a way that produces a superior descent direction. Like BP, CG is an iterative algorithm. During each iteration we update a direction vector $\mathbf{p} \in \mathbb{R}^{N_w}$ as

$$\mathbf{p}_{k+1} \leftarrow -\mathbf{g}_k + B_1\mathbf{p}_k \tag{2.8}$$

$$B_1 = \frac{E_g(k)}{E_g(k-1)} \tag{2.9}$$

where $E_g(k)$ is the sum of squares for the gradient in the k^{th} iteration and are calculated as $\mathbf{g}^T\mathbf{g}$.. For the first iteration $\mathbf{p}$ is taken to be $-\mathbf{g}$. Finally, the weights are updated as

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + z\mathbf{p}_k \tag{2.10}$$

There are several forms of CG. The form illustrated below does not require Hessian information and is thus a first order optimization algorithm.

```
Require: $\epsilon > 0$
  $i \leftarrow 1$
  $\text{XD} \leftarrow 1$
  $\mathbf{p}_0 \leftarrow 0$
  Initialize network
  while $i < N + 1$ do
    Calculate the gradient $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}}$
    $\text{XN} \leftarrow ||\mathbf{g}||_2^2$
    $B_1 \leftarrow \frac{\text{XN}}{\text{XD}}$
    $\text{XD} = \text{XN}$
    $\mathbf{p}_{k+1} \leftarrow -\mathbf{g} + B_1\mathbf{p}_k$
    Minimize $E(\mathbf{w}_k + B_2\mathbf{p}_k)$ with respect to $B_2$ such that
        $\frac{\partial E(\mathbf{w}_k + B_2\mathbf{p}_k)}{\partial B_2} = 0$
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + B_2\mathbf{p}_k$
    $i \leftarrow i + 1$
  end while
```

Algorithm 2.2: Conjugate Gradient Algorithm

2.4 Newton-type algorithms

Newton's algorithm is the basis of a number of popular second order optimization algorithms including Levenberg-Marquardt [51] and BFGS [52]. Newton's algorithm is iterative where each iteration

- Calculates the Hessian and gradient, $\mathbf{H}$ and $\mathbf{g}$ of the MSE error function where the elements of $\mathbf{H}$ are given by

$$h(m,n) = \frac{\partial^2 E}{\partial w(m)\partial w(n)}$$

  and the elements of $\mathbf{g}$ are given by

$$g(n) = \frac{\partial E}{\partial w(n)}$$

- Calculates the Newton direction, $\mathbf{d}$, by solving the set of linear equations

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

- Updates variables with direction $\mathbf{d}$ as

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

Non-quadratic objective functions will require a line search which will calculate a scalar step length z. The network weights, $\mathbf{w}$, are 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 [53]. 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 [54].

Newton's method is derived from a 2^{nd}-order Taylor series approximation to an objective function [55]. Applying this principle to (1.7) gives us

$$E_H(\mathbf{w}) = E_0 + (\mathbf{w} - \tilde{\mathbf{w}})^T\mathbf{g} + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \tilde{\mathbf{w}}) \tag{2.11}$$

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

CHAPTER 3

EQUIVALENT NETWORK THEORY

Equivalent network theory can provide insights into multilayer perceptron training. As we shall see in this section equivalent network theory also affords us an opportunity to improve MLP training algorithms.

We start off by defining strict equivalence.

Definition 3.1. *Two networks are strictly equivalent if* $y(\mathbf{w}) = y(\mathbf{A}\mathbf{w}')$ *where* $\mathbf{w} = \mathbf{A}\mathbf{w}'$ *and* $\mathbf{A}$ *is nonsingular.*

Armed with this definition, we can analyze necessary conditions to produce equivalent networks. First we analyze input equivalence and net equivalence.

3.1 Input Equivalence

Let MLP A be trained using training data with input vectors $\mathbf{x}_p$ as described in Chapter 2. In a second network called MLP B we can have input vectors $\mathbf{x}'_p$, input weights $\mathbf{W}'$ and output weight matrices $\mathbf{W'}_{oh}$ and $\mathbf{W'}_{oi}$. Assume that the input vectors to be used to train MLP B can be defined as $\mathbf{A}\mathbf{x}_p$. In other words, the input vectors for MLP A are linearly transformed by a matrix $\mathbf{A} \in \mathbb{R}^{N' \times (N+1)}$ to serve as input vectors for MLP B.

As with MLP A, the processing equations for MLP B are:

$$\mathbf{n'}_p = \mathbf{W}'\mathbf{x'}_p$$

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

Under these circumstances, we would like to determine the conditions that MLP B becomes an equivalent network to MLP A.

We now relate the weights in the MLP A to those of the transformed network, MLP B. Substituting we obtain:

$$\mathbf{n}'_p = \mathbf{W}'\mathbf{A}\mathbf{x}'_p$$
$$\mathbf{y}'_p = \mathbf{W}'_{oi}\mathbf{A}\mathbf{x}_p + \mathbf{W}'_{oh}\mathbf{O}'_p$$

For strong equivalence $\mathbf{y}'_p = \mathbf{y}_p$ and equating quantities we have

$$\mathbf{W} = \mathbf{W}'\mathbf{A} \tag{3.1}$$
$$\mathbf{W}_{oi} = \mathbf{W}'_{oi}\mathbf{A} \tag{3.2}$$
$$\mathbf{W}_{oh} = \mathbf{W}'_{oh}\mathbf{A} \tag{3.3}$$

The following equations are the delta equations for backpropagation:

$$\delta_{po}(i) = -\frac{\partial E}{\partial y_p(i)} = 2\left[t_p(i) - y_p(i)\right]$$
$$\delta_p(k) = -\frac{\partial E}{\partial n_p(k)} = O'_p(k)\sum_{i=1}^{M}\delta_{po}(i)w_{oh}(i,k)$$

where $O'_p(k)$ is the first derivative of hidden unit activation and

$$\boldsymbol{\delta}_p = \Big[\delta_p(1), \delta_p(2), ..., \delta_p(N_w)\Big]^T$$

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

The weight updates are as follows

$$\mathbf{W}' \leftarrow \mathbf{W}' + \mathbf{G}' \tag{3.4}$$

$$\mathbf{W}\mathbf{A}^{-1} \leftarrow \mathbf{W}\mathbf{A}^{-1} + \mathbf{G}\mathbf{A}^{T} \tag{3.5}$$

For the purposes of improving training, (3.1) implies that we can map this back to the original network by post-multiplying by $\mathbf{A}$. Performing this on (3.4) gives us

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{G}\mathbf{A}^{T}\mathbf{A}$$

In conclusion, it is possible that we can improve training of a network if we can derive equivalent networks that transform a weight modification matrix. This should be a symmetric matrix, non-orthogonal and nonsingular. To this point, we can see an orthogonal transform will not improve training because we do not alter the original descent direction.

3.2 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 $\mathbf{n}'$ as

$$\mathbf{n}' = \mathbf{W}'\mathbf{x}$$

where the final net vector is found through linear transformation as

$$\mathbf{n} = \mathbf{C}\mathbf{n}'$$

where $\mathbf{C}$ is nonsingular. Then $\mathbf{C}$ is related to $\mathbf{W}'$ and $\mathbf{W}$ as

$$\mathbf{C}\mathbf{W}' = \mathbf{W}$$

In other words, we have taken the original network and inserted the matrix $\mathbf{C}^{-1}\mathbf{C}$ 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 $\mathbf{W}$ and $\mathbf{W}'$, their input weight change matrices, $\mathbf{D}$ and $\mathbf{D}'$, are related as

$$\mathbf{D}' = \mathbf{C}^T\mathbf{D}$$

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

$$\mathbf{D}'' = \mathbf{R}_n\mathbf{D} \tag{3.6}$$

where $\mathbf{R}_n = \mathbf{C}\mathbf{C}^T$ as described in [56]. If $\mathbf{C}$ is not an orthogonal matrix then, MLPs 1 and 2 train differently. Were we training the equivalent network, we would simply update the weights as

$$\mathbf{W}' = \mathbf{W}' + z\mathbf{G}' \tag{3.7}$$

However, we are wanting to improve training on the original network. We see that we can accomplish this by premultiplying (3.7) by $\mathbf{C}$ which leads to the following relationship:

$$\mathbf{W} = \mathbf{W} + \mathbf{C}\mathbf{C}^T\mathbf{G} \tag{3.8}$$

3.3 The Multiple Optimal Learning Factors Algorithm: An application of Equivalent Network Theory

In this section we review the Multiple Optimal Learning Factors (MOLF) algorithm. Malalur and Manry[56] provide a comprehensive discussion of the algorithm. In MOLF we calculate a vector $\mathbf{z}$ of optimal learning factors. As in other iterative algorithms, the input weight matrix $\mathbf{W}$ is updated using using these learning factors.

$$\mathbf{W} = \mathbf{W} + \mathbf{R}_{MOLF}\mathbf{D}$$

Let $\mathbf{z}$ be the vector containing the optimal learning factors and z_k represent the OLF used to update each of the hidden unit input weights, $w(k,n)$. The error function to be minimized is given in equation (1.7). In this algorithm the predicted output, $y_p(i)$, becomes

$$y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k)f(\sum_{j=1}^{N+1}[w(k,j) + z_k d(k,j)x_p(j)])$$

where $d(k,j)$ is an element of the descent direction $\mathbf{D}$ and the function $f\colon \mathbb{R}^{N_h} \to \mathbb{R}$ denotes the hidden layer activation function. The negative first partial of E with respect to an OLF z_l is,

$$g_{molf}(m) = -\frac{\partial E}{\partial z_m} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - y_p(i)]\frac{\partial y_p(i)}{\partial z_m} \tag{3.9}$$

$$\frac{\partial y_p(i)}{\partial z_m} = w_{oh}(i,m)f'(n_p(m))v_p(m); \tag{3.10}$$

where,

$$n_p(m) = \sum_{n=1}^{N+1} w(m,n)x_p(n)$$
$$v_p(m) = \sum_{n=1}^{N+1} d(m,n)x_p(n)$$

Using the Gauss-Newton approximation of the Hessian, the second partial derivative elements of the Hessian $\mathbf{H}_{molf}$ are derived as,

$$h_{molf}(m,j) = \frac{\partial^2 E}{\partial z_m \partial z_j} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p(i)}{\partial z_m}\frac{\partial y_p(i)}{\partial z_j} \tag{3.11}$$

Given the Hessian $\mathbf{H}_{molf}$, the error E can be minimized with respect to the vector $\mathbf{z}$ using Newton's method. This means solving:

$$\mathbf{H}_{molf}\mathbf{z} = \mathbf{g}_{molf} \tag{3.12}$$

The information above is summarized in the following MOLF algorithm:

```
Initialize $\mathbf{W}$,$\mathbf{W}_{oi}$,$\mathbf{W}_{oh}$
**Require:** MAXITERS $> 0$
  $k \leftarrow 0$
  **while** $k <$ MAXITERS **do**
    Minimize output weights using OWO
    Calculate descent direction $\mathbf{D}$
    Calculate $z_k$ using Newton's method in (3.12)
    $w(k,n) \leftarrow w(k,n) + z_k d(k,n)$
    Solve linear equations for all output weights.
    **if** stopping criterion reached **then**
      STOP
    **end if**
    $k \leftarrow k+1$
  **end while**
  **if** $k =$ MAXITERS **then**
    **print** Program reached maximum number of iterations
  **end if**
```

Algorithm 3.1: MOLF Algorithm

Thus the MOLF procedure has been successfully applied to OWO-BP, and the resulting algorithm is denoted as MOLF-BP. Similarly the MOLF procedure can also be used to improve other training algorithms.

3.4 Neural Network Training Invariance Properties

It is well known that Newton's algorithm has quadratic convergence and is affine invariant[55]. We can define affine invariance in neural networks as follows

Definition 3.2. *If two equivalent networks are formed whose objective functions satisfy* $E(\mathbf{w}') = E(\mathbf{Tw})$ *with* $\mathbf{w}' = \mathbf{Tw}$*, and an iteration of an optimization method yields* $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{d}$ *and* $\mathbf{w}' \leftarrow \mathbf{w}' + \mathbf{d}'$ *where* $\mathbf{w}$ *and* $\mathbf{w}'$ *are* n*-dimensional, the training method is affine invariant if* $\mathbf{d}' = \mathbf{Td}$ *for every nonsingular matrix* $\mathbf{T}$*.*

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

Definition 3.3. *If a training algorithm satisfies the conditions in Definition 3.2 except that* $\mathbf{T}$ *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 3.1. *Suppose two equivalent networks initially satisfy* $\mathbf{w} = \mathbf{Tw}'$ *where* $\mathbf{T}$ *is any nonsingular* $n \times n$ *matrix consistent with the training algorithm and* $\mathbf{w}$ *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. □

Lemma 3.2. *Partially affine invariant algorithms satisfy Lemma 3.1.*

Proof. The proof is the same as that of Lemma 3.1. □

CHAPTER 4

PROBLEMS AND PROPOSED TASKS

In this chapter we outline a few second order MLP training problems and propose research tasks to solve them.

4.1 Problems

4.1.1 Incomplete Analysis for Newton Training

It is known that the full network Hessian is singular or numerically singular[57]. Smagt and Hirzinger [58] provide a variety of reasons why this is the case, while Wille [53] provides a rigorous proof of this fact. Singular or numerically singular Hessians will break 2^{nd} order algorithms if care is not taken. Indefinite Hessians can cause training algorithms to reach a saddle point instead of a true minimum.

On the other hand, positive definiteness guarantees that the nonlinear least squares problem is well-posed[59]. We believe that we have found another cause for the numerical singularity of the MLP Hessian: Newton training with all of the weights leads to a discrepancy between the assumed and actual models of the MSE error function.

Newton training, the most desirable second order training, assumes a first order piecewise affine model. When using a second order model, we encounter problems in MLP training.

4.1.2 Computational complexity of Levenberg-Marquardt

Levenberg-Marquardt (LM) trains an MLP well, but is not scalable due to its computational complexity. This limits the number and variety of applications for which it would be a suitable training method. It is desirable to use algorithms that perform at least as well as LM, but with less computational complexity.

4.1.3 Undeveloped MLP invariance properties

A training algorithm may reach a point of convergence, but a simple transformation of either the weights or the data could yield a lower training error or faster convergence. Invariance properties of a training algorithm must be analyzed to ensure optimal performance for a given coordinate system. Invariance properties of a multilayer perceptron are undeveloped to date.

We must provide a suitable framework that provides concrete basis for our analysis.

4.2 Proposed Tasks

We propose strategies to solve the problems outlined above by developing new algorithms and conducting novel analyses. The effectiveness of the solutions will be shown in simulations.

4.2.1 Analyze problems with Newton training

We will expound on the problems with Newton training by presenting and analyzing the full network Hessian. We will determine implicit and explicit assumptions of Newton's method and show implications of these assumptions on MLP training. This analysis will provide a roadmap for solving second order training problems.

4.2.2 Develop MLP invariance property theory

We will develop notions of affine invariance for multilayer perceptrons. In doing so, we find that some training algorithms have different invariance properties which we will call partial affine invariance. We develop a framework that builds on the notions of MLP invariance properties and test new algorithms for these properties.

4.2.3 Develop multistep affine invariant 2^{nd} order training algorithms

Affine-invariant training algorithms ensure us that our training methods are effective regardless of coordinate system. Creating second order algorithms ensures fast training. We can avoid some of of the second order training problems by using multistep methods. We will develop these algorithms and observe the performance.

4.2.4 Develop affine invariant one-step training algorithms

Once the problems of Newton MLP training have been analyzed we can also develop algorithms that use second order methods to train all weights simultaneously. We develop and demonstrate novel one step methods to accomplish just this task.

4.2.5 Evaluate new algorithms against reference algorithms

After we develop the new 2^{nd} order algorithms we will evaluate their performance against reference algorithms. The metrics for this evaluation are training error and cumulative multiplies. We also demonstrate their effectiveness by calculating the validation error.

CHAPTER 5

THE OWO-ONT ALGORITHM

In this chapter we discuss OWO-ONT, the Output Weight Optimization-Optimal Net Transform algorithm, which improves the input weight matrix $\mathbf{W}$. ONT is combined later with OWO, yielding a two-step training algorithm. Using (3.6) we can improve $\mathbf{D}$ from HWO by premultiplying it with an optimal matrix $\mathbf{R}_n$ found as

$$\mathbf{R}_n = \arg\min_{\mathbf{R}} \; E(\mathbf{W} + \mathbf{R}\mathbf{D}) \tag{5.1}$$

5.1 Details

Following (3.6) the updated net function for ONT is

$$n_p(k) = \sum_{n=1}^{N+1} \left[w(k,n) + \sum_{j=1}^{N_h} r_n(k,j) d(j,n) \right] x_p(n) \tag{5.2}$$

The partial derivative of $E(\mathbf{W} + \mathbf{R}_n\mathbf{D})$ with respect to an element of the transform matrix $\mathbf{R}_n$ is

$$\frac{\partial E}{\partial r_n(j,m)} = -\frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] \frac{\partial y_p(i)}{\partial r_n(j,m)} \tag{5.3}$$

The second partial is of $E(\mathbf{W} + \mathbf{R}_n\mathbf{D})$ with respect to $\mathbf{R}_n$ is

$$\begin{aligned} &\frac{\partial^2 E}{\partial r_n(j,m) \partial r_n(u,v)} \\ &= \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \frac{\partial y_p(i)}{\partial r_n(j,m)} \frac{\partial y_p(i)}{\partial r_n(u,v)} \end{aligned} \tag{5.4}$$

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

where

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

Mapping (5.3) to $\mathbf{g}_n$ and (5.4) to $\mathbf{H}_n$, we have the standard equations for Newton's algorithm,

$$\mathbf{H}_n \mathbf{r}_n = \mathbf{g}_n \tag{5.5}$$

which are solved for $\mathbf{r}_n$ using orthogonal least squares (OLS). $\mathbf{R}_n$ is found from $\mathbf{r}_n$ as $\mathbf{R}_n = vec^{-1}(\mathbf{r}_n)$. In an earlier version of this algorithm, known as OWO-MOLF [56], we constrained $\mathbf{R}_n$ to be a diagonal matrix.

5.2 Properties of ONT and OWO-ONT

We now investigate some of the properties of the entire OWO-ONT algorithm and the Optimal Net Transform component.

Lemma 5.1. *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 $\mathbf{W}$ and $\mathbf{W}'$ are modified as

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{R}_n \mathbf{D} \tag{5.6}$$

$$\mathbf{W}' \leftarrow \mathbf{W}' + \mathbf{R}'_n \mathbf{D}' \tag{5.7}$$

Multiplying the equivalent network's update in (5.7) by $\mathbf{C}$ and remembering that $\mathbf{D}' = \mathbf{C}^T \mathbf{D}$, we get

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{C}\mathbf{R}'_n \mathbf{C}^T \mathbf{D}$$

Since ONT uses Newton's method to find $\mathbf{R}'_n$ and $\mathbf{R}_n$, affine invariance means that the two networks remain equivalent after the iteration and

$$\mathbf{R}_n = \mathbf{C}\mathbf{R}'_n \mathbf{C}^T$$

□

At this point we see that ONT is either affine invariant or partially affine invariant. We need to determine the properties of $\mathbf{T}$ to determine which property the algorititthm has.

Lemma 5.2. *ONT is affine invariant if*

1. $N_h \geq N+1$ *and*
2. $rank(\mathbf{D})$ *is* $N+1$.

Otherwise ONT is partially affine invariant.

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

$$\mathbf{R}_n\mathbf{D} = \mathbf{E}_N \tag{5.8}$$

where $\mathbf{E}_N$ is the $N_h \times (N+1)$ weight change matrix for $\mathbf{W}$ from Newton's algorithm and $\mathbf{R}_n\mathbf{D}$ is the $N_h \times (N+1)$ weight change matrix for $\mathbf{W}$ from ONT . Transposing both sides of (5.8) we get

$$\mathbf{D}^T\mathbf{R}_n^T = \mathbf{E}_N^T \tag{5.9}$$

Case 1: $N_h > N+1$ (underdetermined system)

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

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

Here we have more equations than unknowns so (5.9) is not solved exactly.

Case 3: $N_h = N+1$

In this case, equation (5.9) can be solved exactly and we have identical weight change matices for ONT and OWO-Newton.

□

Lemma 5.3. *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, $\mathbf{R}_n$ is calculated with Newton's method which is affine invariant. Because we can construct a sparse $\mathbf{T}$ for the entire OWO-ONT algorithm as

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

ONT is partially affine invariant. □

Because OWO-ONT is partially affine invariant, it satisfies Lemma 3.1.

5.3 Computational Burden

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

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

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

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

Then we must solve the linear equations to calculate $\mathbf{r}_n$ which requires

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

multiplies. The number of multiplies required for the $\mathbf{R}_n\mathbf{D}$ product is

$$M_{product} = N_h(N+1)(2N_h - 1)$$

Putting this together we have

$$M_{ONT} = M_{owo} + M_{OLS} + M_{product} + M_H$$

By comparison MOLF, a sparse version of ONT, requires

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

multiplies where

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

Here M_{owo-bp} denotes the number of multiplies of a two-step algorithm called OWO-BP [60] that alternately uses BP to improve the input weights and uses OWO to solve for the output weights.

5.4 Algorithm Summary

The OWO-ONT algorithm is summarized below

```
  Initialize $\mathbf{W}$,$\mathbf{W}_{oi}$,$\mathbf{W}_{oh}$
Require: MAXITERS > 0
  $k = 0$
  while $k <$ MAXITERS do
    Calculate $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ using OWO
    Calculate a descent direction $\mathbf{D}$
    Calculate $\mathbf{g}_n$ elements using (5.3)
    Calculate $\mathbf{H}_n$ elements using (5.4)
    Solve (5.5) for $\mathbf{r}_n$
    $\mathbf{R}_n = vec^{-1}(\mathbf{r}_n)$
    Update the input weight matrix as $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{R}_n\mathbf{D}$
    if stopping criteria reached then
      STOP
    end if
    $k \leftarrow k + 1$
  end while
```

Algorithm 5.1: OWO-ONT Algorithm

5.5 Comments

From cases 2 and 3 of Lemma 5.2, equation (5.9) can be solved for an arbitrary weight change matrix $\mathbf{E}_N$. This suggests that we should use OWO-Newton rather than finding $\mathbf{E}_N$.

Figure 5.1 compares the performance of OWO-ONT against OWO-Newton for $N_h = N + 1$. As illustrated, the plots are right on top of another. This is one experimental demonstration of the validity of Lemma 5.2.

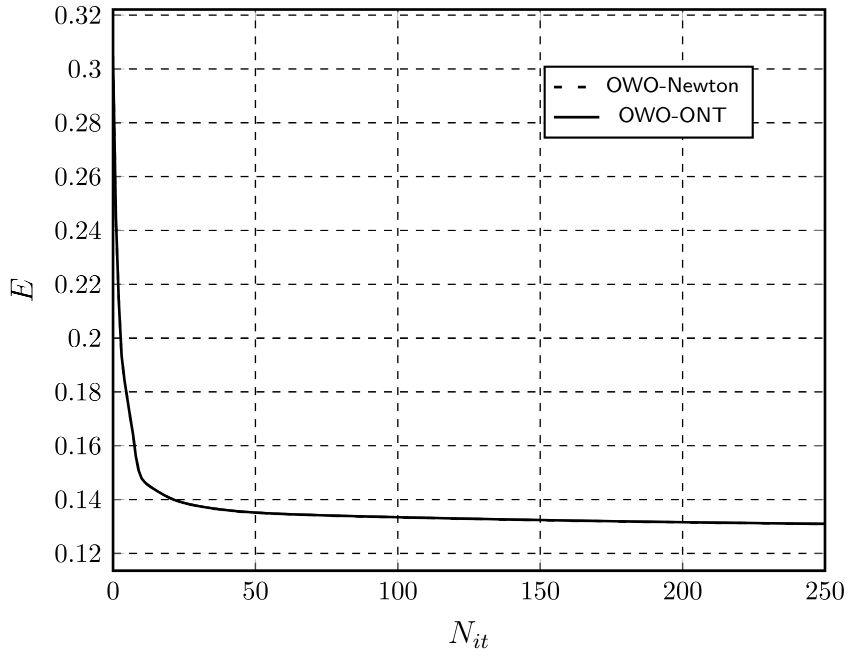

Figure 5.1. Comparison of performance for OWO-Newton and OWO-ONT for $N_h = N + 1$.

CHAPTER 6

MULTISTEP METHODS

In this chapter we analyze Newton MLP training and motivate multistep training.

6.1 Motivation

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 [55]. Applying this principle to (1.7) gives us

$$E_H(\mathbf{w}) = E_0 + (\mathbf{w} - \tilde{\mathbf{w}})^T \mathbf{g} + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^T \mathbf{H} (\mathbf{w} - \tilde{\mathbf{w}}) \tag{6.1}$$

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

When applied to an MSE as in (1.7), 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 $\mathbf{w}$.

6.2 Piecewise affine model of a single hidden layer MLP

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

$$\tilde{E} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - \tilde{y}_p(i)]^2 \tag{6.2}$$

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

$$\tilde{y}_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)\left[O_p(k) + O'_p(k)(n_p(k) - \tilde{n}_p(k))\right] \tag{6.3}$$

and

$$O'_p(k) = \frac{\partial O_p(k)}{\partial n_p(k)},$$
$$\tilde{n}_p(k) = \sum_{n=1}^{N+1} \tilde{w}(k,n)x_p(n)$$

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 affine model of $y_p(i)$.

The validity of the piecewise affine model is demonstrated by,

$$\frac{\partial E}{\partial w(u,v)} = \frac{\partial \tilde{E}}{\partial w(u,v)} \tag{6.4}$$

and

$$\frac{\partial^2 E}{\partial w(u,v)\partial w(m,j)} = \frac{\partial^2 \tilde{E}}{\partial w(u,v)\partial w(m,j)} \tag{6.5}$$

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

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

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

$$\tilde{\mathbf{y}}_p = \left[\mathbf{W}_{oi} + \mathbf{W}_{oh}diag(\mathbf{O}'_p)\mathbf{W}\right]\mathbf{x}_p + \mathbf{W}_{oh}\left[\mathbf{O}_p - diag(\mathbf{O}'_p)\tilde{\mathbf{n}}_p\right] \tag{6.6}$$

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 $E_H(\mathbf{w})$ in (6.1) and $\tilde{E}$ in (6.2). 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 (6.6) are first degree and the discrepancy vanishes as seen in the input weight case in (6.4) and (6.5).

6.3 Extension to Multiple Hidden Layers

Assume that a fully connected network has N_L layers. For the p^{th} pattern, let $\mathbf{O}_{mp}$ and $\mathbf{O'}_{mp}$ respectively denote the m^{th} layer's output vector and its derivative with respect to the net functions. Let $\mathbf{W}_{mn}$ denote the weight matrix connecting $\mathbf{O}_{np}$ to the m^{th} layer's net function vector, and let $\mathbf{W}^t_{mn}$ represent the total gain from layer n to layer m, which consists of $\mathbf{W}_{mn}$ and products of other matrices. For the p^{th} pattern, our piecewise affine model for the output vector $\mathbf{y}_p$ is

$$\tilde{\mathbf{y}}_p = \mathbf{W}^t_{N_L,1}\mathbf{x}_p + \mathbf{m}_{yp}$$
$$\mathbf{m}_{yp} = \sum_{m-2}^{N_L-1} \mathbf{W}^t_{N_L,m}\mathbf{O}_{mp}$$

where

$$\begin{aligned}
\mathbf{W}^t_{N_L,N_L-1} &= \mathbf{W}_{N_L,N_L-1} \\
\mathbf{W}^t_{N_L,1} &= \mathbf{W}^t_{N_L,N_L-2} diag(\mathbf{O'}_{N_L-2,p}) \\
\mathbf{W}^t_{N_L,1} &= \mathbf{W}^t_{N_L,N_L-3} diag(\mathbf{O'}_{N_L-3,p}) \\
&\vdots
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{W}^1_{mn} &= \mathbf{W}_{mn} diag(\mathbf{O'}_{np}) \\
\mathbf{W}^2_{m,m-2} &= \mathbf{W}^1_{m,m-2} + \mathbf{W}^1_{m,m-1}\mathbf{W}^1_{m-1,m-2} \\
\mathbf{W}^3_{m,m-3} &= \mathbf{W}^1_{m,m-3} + \mathbf{W}^2_{m,m-2}\mathbf{W}^1_{m-2,m-3} + \mathbf{W}^1_{m,m-1}\mathbf{W}^2_{m-1,m-3} \\
&+ \mathbf{W}^1_{m,m-1}\mathbf{W}^1_{m-1,m-2}\mathbf{W}^1_{m-2,m-3} \\
&\vdots
\end{aligned}$$

Clearly $\tilde{\mathbf{y}}_p$ is a degree $N_L - 1$ function of the unknowns, and $\tilde{E}$ is a degree $2(N_L - 1)$ function of the unknowns.

We can make the following points:

(P1) When $\mathbf{w}$ includes all network weights, E_H is a second degree model of E, but $\tilde{E}$ is of degree $2(N_L - 1)$ in $\mathbf{w}$, and this discrepancy increases with N_L.

(P2) The terms causing the discrepancy correspond to products of weights from different layers, that are not well-modeled by E_H.

6.4 Implications for MLP training

The main implication of the work of the previous sections is that if all the unknown weights are in the same layer, the discrepancy is absent and $E_H = \tilde{E}$. This suggests that when training a MLP with Newton's algorithm, we should try a "layer by layer" or multistep approach called block coordinate descent (BCD) [61].

The full network Hessian which can be expressed in block form as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_R & \mathbf{H}_{oi}^T \\ \mathbf{H}_{oi} & \mathbf{H}_o \end{bmatrix} \tag{6.7}$$

The elements of the Gauss-Newton input weight Hessian, $\mathbf{H}_R$, are given by

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

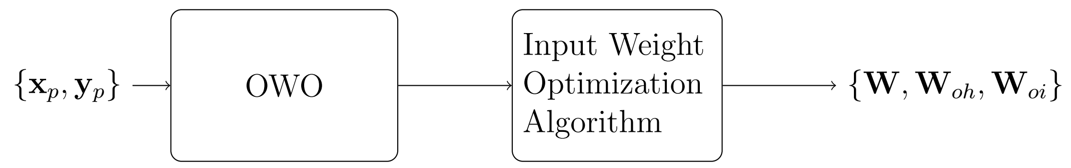
$\{\mathbf{x}_p, \mathbf{y}_p\}$
OWO
Input Weight Optimization Algorithm
$\{\mathbf{W}, \mathbf{W}_{oh}, \mathbf{W}_{oi}\}$

Figure 6.1. Illustration of Block Coordinate Descent methodology.

The elements of $\mathbf{H}_{oi}$ are calculated by

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

The block diagonal output weight Hessian matrix $\mathbf{H}_o$ is specified as

$$\mathbf{H}_o = \begin{bmatrix} 2\mathbf{R} & 0 & 0 & \cdots & 0 \\ 0 & 2\mathbf{R} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & \vdots & 0 & 2\mathbf{R} & 0 \\ 0 & 0 & 0 & 0 & 2\mathbf{R} \end{bmatrix}$$

where $\mathbf{R}$ is the autocorrelation matrix given in (2.2).

6.5 Block Coordinate Descent Methods for the MLP

Our multistep methods employ Block Coordinate Descent techniques which work by optimizing an objective function with respect to a subset of a network's weights while keeping the remainder fixed[62, 61]. Proceeding in this fashion over several iterations we can optimize the objective function with respect to all weights. These training algorithms minimize the output weight matrices $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ using Output Weight Optimization (OWO) as discussed in section §2.1. Figure 6.5 illustrates our BCD framework.

An iteration consists of two parts. In the first part of the iteration we fix the input weights, $\mathbf{W}$, and minimize output weights $\mathbf{W}_{oh}$ and $\mathbf{W}_{oi}$ by using the Output

Weight Optimization algorithm. During the second half of the iteration, we fix the output weights just found and optimize using equations (1.7) and (1.5) over $\mathbf{W}$.

We continue to iterate until we have met some predetermined stopping criterion. Algorithm 6.1 details the technique of our multistep methods.

```
  Initialize $\mathbf{W}$,$\mathbf{W}_{oi}$,$\mathbf{W}_{oh}$
Require: MAXITERS $> 0$
  $k = 0$
  while $k <$ MAXITERS do
    Fix $\mathbf{W}$
    Minimize output weights using OWO
    Fix $\mathbf{W}_{oi}$ and $\mathbf{W}_{oh}$
    $\mathbf{D} \leftarrow$ descent direction to minimize input weights $\mathbf{W}$ using first or second order
    algorithm
    $z \leftarrow$ linesearch($\mathbf{D}$)
    $\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D}$
    if stopping criteria reached then
      STOP
    end if
    $k \leftarrow k + 1$
  end while
  if $k =$ MAXITERS then
    print Program reached maximum number of iterations
  end if
```
Algorithm 6.1: Multistep training in Neural Networks

CHAPTER 7

THE OWO-NEWTON ALGORITHM

One BCD approach to avoid the violating the assumptions of Newton's method discussed in §6.1 is to use Newton's algorithm separately for input and output weights. We give details of this method in this section.

7.1 Details

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:

$$\begin{aligned} g(j,k) &= -\frac{\partial E}{\partial w(j,k)} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)] \frac{\partial y_p(i)}{\partial w(j,k)} \\ \frac{\partial y_p(i)}{\partial w(j,k)} &= w_{oh}(i,j) O'_p(j) x_p(k) \end{aligned} \tag{7.1}$$

We use orthogonal least squares [46] to solve:

$$\mathbf{H}_R \mathbf{d} = \mathbf{g} \tag{7.2}$$

for $\mathbf{d}$ and update the input weight matrix $\mathbf{W}$ as $\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D}$ where $\mathbf{g} = vec(\mathbf{G})$ and $\mathbf{d} = vec(\mathbf{D})$. Here the $vec()$ operator performs a lexicographic ordering of $\mathbf{W}$.

Here z is the learning factor resulting from a line search. A line search is necessary for OWO-Newton because (1.7) is not a quadratic function of $\mathbf{W}$.

7.2 Learning Factor Calculation

As mentioned in §7.1, OWO-Newton updates the input weight matrix as $\mathbf{W} \leftarrow \mathbf{W} + z\mathbf{D}$. Because E is not quadratic in $\mathbf{W}$, z can have values other than one as seen in an example in Fig. 7.1; consequently we must determine a suitable value. For this problem $\mathbf{W}$ and $\mathbf{D}$ are fixed and E is a function of z. Minimizing $E(z)$ is also known as the line search subproblem and there are two approaches: exact line searches and inexact line searches.

Exact line searches solve the following optimization problem:

$$Z_{OLF} = \arg\min_{z} \; E\left(\mathbf{W} + z\mathbf{D}\right) \tag{7.3}$$

which often has the solution

$$\frac{\partial E\left(\mathbf{W} + z\mathbf{D}\right)}{\partial z} = 0$$

In practice, instead of finding an analytical solution to (7.2), we use iterative solution methods to approximate Z_{OLF} to a chosen tolerance. Such inexact line searches lead to an equivalent minimization performance [63]. The first step in approximating (7.3) is calculating the partial derivatives with respect to z. The first partial of E with respect to z is

$$\frac{\partial E}{\partial z} = -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - y_p(i)\right]\frac{\partial y_p(i)}{\partial z} \tag{7.4}$$

$$\frac{\partial y_p(i)}{\partial z} = \sum_{k=1}^{N+1} w_{oh}(i,k)O'(n_p(k))\sum_{n=1}^{N+1} d(k,n)x_p(n)$$

The Gauss-Newton approximation of the second partial of E with respect to z is

$$\frac{\partial^2 E}{\partial z^2} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[\frac{\partial y_p(i)}{\partial z}\right]^2$$

Given these partials, we then use the Newton-Raphson method [64] to determine the learning factor update as:

$$\Delta z = -\frac{\dfrac{\partial E}{\partial z}}{\dfrac{\partial^2 E}{\partial z^2}}$$

and then update z as $z \leftarrow z + \Delta z$.

It is important to note that we must initialize Newton-Raphson with z to some small positive number other than zero so that we can correctly evaluate $E(\mathbf{W} + z\mathbf{D})$. For our line searches we use $z = 0.1$.

Iterative methods such as Newton-Raphson must have some stopping criteria. Because OWO-Newton is itself an iterative method, we do not require a high degree or accuracy in z therefore in this work we stop after 2 iterations.

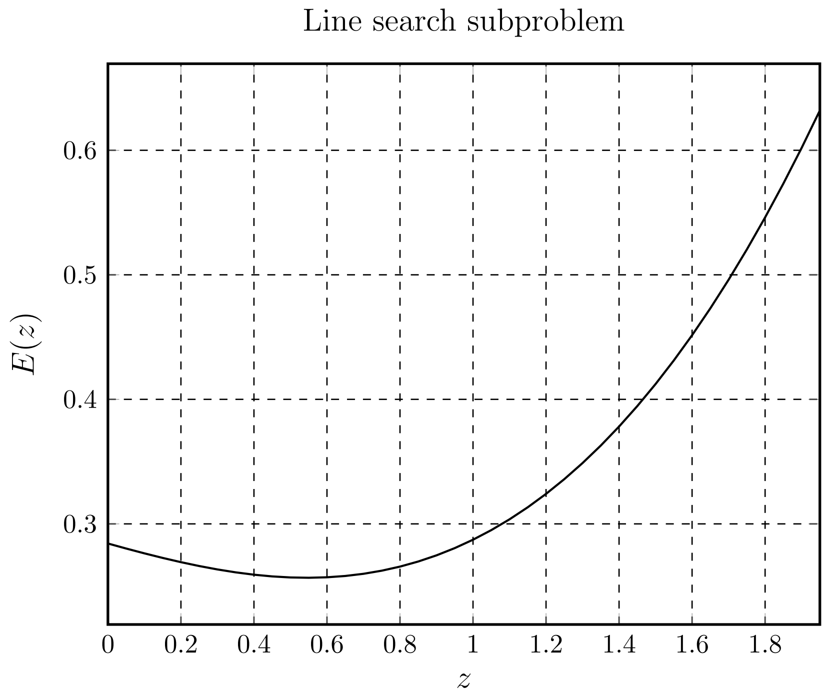

Figure 7.1. Illustration of $E(z)$ in one iteration of OWO-Newton using the Remote Sensing training file..

7.3 Computational Burden

When analyzing the computational complexity of one iteration of OWO-Newton we first use OWO which requires

$$M_{owo} = N_v(N_u + 1)\left(M + \frac{N_u}{2}\right) \tag{7.5}$$

multiplies. The input gradient must be calculated requiring

$$M_G = MN_v\left(N_u + 2\right)$$

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

$$M_H = N_v N_{iw}\left(N_{iw} + \frac{3}{2}\right) + \frac{M(N+1)(N+2)}{2} \tag{7.6}$$

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

$$M_{ols} = N_{iw}(N_{iw} + 1)\left[M + \frac{N_{iw}(N_{iw} + 1)}{6}\right] + \frac{3}{2} \tag{7.7}$$

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

$$M_{OWO-Newton} = M_{owo} + M_H + M_{ols} + M_G \tag{7.8}$$

7.4 Properties of OWO-Newton

Lemma 7.1. *In OWO-Newton, both input weight training and output weight training are individually affine invariant via Lemma 2.1.*

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

We are not claiming that OWO-Newton, as a whole, is affine invariant. In multistep algorithms we train subsets of weights so we need to define an appropriate type of affine invariance for this case.

Because (2.1) is quadratic in $\mathbf{W}_o$, the minimization is accomplished in one step.

Lemma 7.2. *The OWO-Newton algorithm is partially affine invariant.*

Proof. Since Newton's method individually trains $\mathbf{W}_o$ and $\mathbf{W}$ there exist matrices $\mathbf{T}_{OWO}$ and $\mathbf{T}_W$. As a result, we can construct a sparse $\mathbf{T}$ for the entire algorithm as

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_{OWO} & \vdots & \mathbf{0} \\ \cdots & & \cdots \\ \mathbf{0} & \vdots & \mathbf{T}_W \end{bmatrix} \tag{7.9}$$

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

7.5 Summary

The OWO-Newton algorithm can be summarized as follows:

```
Require: MAXITERS > 0
  Initialize $\mathbf{W}$, $\mathbf{W}_{oi}$ and $\mathbf{W}_{oh}$
  for k=1 to MAXITERS do
      Perform OWO
      Calculate $\mathbf{G}$
      $\mathbf{g} \leftarrow vec(\mathbf{G})$
      Calculate $\mathbf{H}_R$
      Solve (7.2) 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
```

Algorithm 7.1: OWO-Newton Algorithm

CHAPTER 8

HYBRID ONE STEP TRAINING ALGORITHM

A second technique to avoid the model mismatch described in §6.1 is to set the elements of $\mathbf{H}_{oi}$ to zero and to use the modified Hessian to train with Newton's method.

8.1 Details

The Hybrid One Step Training algorithm, HOST, first calculates $\mathbf{H}_w$ from $\mathbf{H}$ as

$$\mathbf{H}_w = \begin{bmatrix} \mathbf{H}_R & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_o \end{bmatrix} \tag{8.1}$$

where the subscript w denotes a windowing operation. Next the algorithm solves

$$\mathbf{H}_w \mathbf{d} = \mathbf{g} \tag{8.2}$$

where

$$\mathbf{g} = \begin{bmatrix} vec(\mathbf{G}) \\ vec(2\mathbf{C} - 2\mathbf{R}\mathbf{W}_o^T) \end{bmatrix} \tag{8.3}$$

Here $\mathbf{W}_o$ is the output weight matrix from the previous iteration.

After solving (8.2), $\mathbf{d}$ can be reshaped into the input weight change matrix $\mathbf{D}$ and the bypass and output weight change matrices $\mathbf{D}_{oi}$ and $\mathbf{D}_{oh}$. Because our

MSE objective is not a quadratic function of $\mathbf{w}$, we can multiply these matrices by a separate learning factor before updating the weights as

$$\begin{aligned} \mathbf{W} &\leftarrow \mathbf{W} + z_1\mathbf{D} \\ \mathbf{W}_{oh} &\leftarrow \mathbf{W}_{oh} + z_2\mathbf{D}_{oh} \\ \mathbf{W}_{oi} &\leftarrow \mathbf{W}_{oi} + z_3\mathbf{D}_{oi} \end{aligned} \tag{8.4}$$

where z_1, z_2 and z_3 are the learning factors for the input, output and bypass weight change arrays respectively.

8.2 Learning factor calculation

Typical optimization algorithms use a scalar learning factor. For HOST1 then, $z_1 = z_2 = z_3$. Such a technique can be suboptimal as it fails to take into account the different scales of the elements of the descent vector. Given descent vectors for the input weights, bypass weights and output weights, $\mathbf{D}$, $\mathbf{D}_{oi}$ and $\mathbf{D}_{oh}$ respectively, we can calculate a three-dimensional learning factor vector

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

To calculate the learning factors we formulate the new output equation

$$\begin{aligned} y_p(i) = &\sum_{n=1}^{N+1} \left[w_{oi}(i,n) + z_3 d_{oi}(i,n) \right] x_p(n) \\ &+ \sum_{k=1}^{N_h} \left[w_{oh}(i,k) + z_2 d_{oh}(i,k) \right] f\left(\sum_{n=1}^{N+1} \left[w(k,n) + z_1 d(k,n) \right] x_p(n) \right) \end{aligned}$$

We use Newton's method to obtain the learning factor vector $\mathbf{z}$ by solving

$$\mathbf{H}_3\mathbf{z} = \mathbf{g}_3$$

Table 8.1. Example learning factors generated by HOST3.

z_1	z_2	z_3
0.419782	0.309262	0.308763
0.422062	0.196758	0.198456
0.383299	0.330176	0.329678
0.454192	0.400011	0.403339
0.408844	0.291923	0.294441

where

$$h_3(m,n) = \frac{\partial^2 E}{\partial z_m \partial z_n} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \frac{\partial y_p(i)}{\partial z_m} \frac{\partial y_p(i)}{\partial z_n}$$

and

$$g_3(n) = -\frac{\partial E}{\partial z_n} = \frac{2}{N_v} \sum_{i=1}^{M} \sum_{p=1}^{N_v} \left[t_p(i) - y_p(i)\right] \frac{\partial y_p(i)}{\partial z_n},$$

where

$$\frac{\partial y_p(i)}{\partial z_1} = \sum_{k=1}^{N_h} w_{oi}(i,k) O_p{}'(k) \sum_{m=1}^{N+1} d(k,m) x_p(m),$$

$$\frac{\partial y_p(i)}{\partial z_2} = \sum_{i=1}^{N_h} d_{oh}(i,k) O_p(k),$$

$$\frac{\partial y_p(i)}{\partial z_3} = \sum_{i=1}^{N+1} d_{oi}(i,k) x_p(k)$$

Table 8.1 provides some example values for $\mathbf{z}$. Though z_2 and z_3 in these cases are about the same, they are not equal to z_1 which indicates the usefulness of the 3 dimensional learning factor.

8.3 Computational Burden

In this section we analyze the computational burden for one iteration of HOST3. Calculating the Gauss-Newton input weight Hessian $\mathbf{H}_R$ requires

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

multiplies. The numbers of required multiplies for calculating the learning factor Hessian and gradient are respectively

$$M_{H3} = 2MN_v \left[2(N+1)N_h + N_u\right]$$

and

$$M_{g3} = MN_v \left[2(N+1)N_h + N_u\right]$$

Solving the linear equations for $\mathbf{D}$ with OLS requires

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

multiplies. Additionally, calculating $\mathbf{D}_{oi}$ and $\mathbf{D}_{oh}$ require the same number of multiplies as OWO, which is

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

Pulling the components together we arrive at the following expression for the HOST3 computational burden for one iteration,

$$M_{HOST3} = M_H + M_{H3} + M_{g3} + M_{ols} + M_{owo}$$

8.4 Properties of HOST

Lemma 8.1. *The HOST algorithm is partially affine invariant*

Proof. Newton's method solves equations of the form $\mathbf{Hd} = \mathbf{g}$ and updates weights as $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{H}^{-1}\mathbf{g}$. Letting $\mathbf{T}$ be a nonsingular transform matrix as in definition 3.2, substitute $\mathbf{T}^{-1}\mathbf{Td}$ for $\mathbf{d}$, yielding $\mathbf{HT}^{-1}\mathbf{Td} = \mathbf{g}$. Multiplying by $(\mathbf{T}^{-1})^T$ we get

$$\mathbf{H}'\mathbf{d}' = \mathbf{g}'$$

where

$$\begin{aligned}\mathbf{H}' &= (\mathbf{T}^{-1})^T\mathbf{H}\mathbf{T}^{-1}\\ \mathbf{g}' &= (\mathbf{T}^{-1})^T\mathbf{g}\\ \mathbf{d}' &= \mathbf{T}\mathbf{d}\end{aligned} \tag{8.5}$$

Affine invariance of Newton's algorithm implies that $\mathbf{H}'$ is the Hessian matrix for the equivalent network using weight vector $\mathbf{w}'$.

HOST first obtains the full MLP Hessian $\mathbf{H}$. Then it is windowed yielding

$$\mathbf{H}_w\mathbf{d} = \mathbf{g} \tag{8.6}$$

For an equivalent network trained by HOST we form $\mathbf{H}'$ as in (8.5) and window it yielding

$$(\mathbf{H}')_w\mathbf{d}' = \mathbf{g}' \tag{8.7}$$

Substituting $\mathbf{T}^{-1}\mathbf{T}\mathbf{d}$ for $\mathbf{d}$ in (8.6) and multiplying by $(\mathbf{T}^{-1})^T$ yields

$$(\mathbf{H}_w)'\mathbf{d}' = \mathbf{g}' \tag{8.8}$$

Assuming that HOST is affine invariant, (8.7) and (8.8) imply that

$$\mathbf{H}'_w = (\mathbf{H}_w)' \tag{8.9}$$

Without loss of generality, express $\mathbf{T}^{-1}$ as

$$\mathbf{U} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\\ \mathbf{U}_3 & \mathbf{U}_4\end{bmatrix}$$

where the blocks of $\mathbf{U}$ are the same size as the corresponding blocks of $\mathbf{T}$. Now (8.9) requires that

$$\begin{aligned}\mathbf{U}_3\mathbf{H}_3\mathbf{U}_1 + \mathbf{U}_1\mathbf{H}_2\mathbf{U}_3 &= \mathbf{0},\\ \mathbf{U}_4\mathbf{H}_3\mathbf{U}_2 + \mathbf{U}_2\mathbf{H}_2\mathbf{U}_4 &= \mathbf{0}\end{aligned} \tag{8.10}$$

$$\begin{aligned}\mathbf{U}_2\mathbf{H}_1\mathbf{U}_1 + \mathbf{U}_4\mathbf{H}_4\mathbf{U}_3 &= \mathbf{0}, \\ \mathbf{U}_1\mathbf{H}_1\mathbf{U}_2 + \mathbf{U}_3\mathbf{H}_4\mathbf{U}_4 &= \mathbf{0}\end{aligned} \tag{8.11}$$

We can see that (8.10) and (8.11) are satisfied only when the elements of $\mathbf{U}_2$ and $\mathbf{U}_3$ equal 0. This means that $\mathbf{U}$ and therefore $\mathbf{T}$ are sparse and therefore HOST is partially affine invariant. □

8.5 Summary

The HOST3 algorithm can be summarized as follows:

```
Require: MAXITERS > 0
  Initialize $\mathbf{W}$, $\mathbf{W}_{oi}$ and $\mathbf{W}_{oh}$
  for k=1 to MAXITERS do
    Calculate $\mathbf{R}$, $\mathbf{C}$, $\mathbf{G}$
    Calculate $\mathbf{g}$ as in (8.3)
    Calculate $\mathbf{H}_R$
    Solve (8.2) for $\mathbf{d}$
    $\{\mathbf{D}, \mathbf{D}_{oi}, \mathbf{D}_{oh}\} = vec^{-1}(\mathbf{d})$
    Calculate $\mathbf{H}_3$ and $\mathbf{g}_3$
    Calculate $\mathbf{z}$
    Update network weights as in (8.4)
  end for
```

Algorithm 8.1: HOST3 Algorithm

If the HOST algorithm is tried in a network having two or more hidden layers, E is again of degree four or more, so the discrepancy of §6.1 reappears. OWO-Newton does not have this constraint.

CHAPTER 9

EXPERIMENTAL RESULTS

In this chapter we use ten-fold training and validation for each datafile. We demonstrate the performance of the new algorithms against reference algorithms. For OWO-ONT, we have chosen reference algorithms CG, LM and OWO-MOLF. For OWO-Newton and HOST3 we have chosen LM and CG as the reference algorithms.

9.1 Remote Sensing: Twod

We now demonstrate the new algorithms on the IPNNL remote sensing dataset [65] against reference algorithms. The goal is to predict certain measurements related to electromagnetic scattering such as surface permittivity, normalized surface rms roughness and surface correlation length[60]. 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 featuring OWO-ONT are shown in Figure 9.1. The training error of LM is slightly better than OWO-ONT in this example. Figure 9.2 shows a slightly better training and validation error for LM over OWO-Newton and HOST3, but requiring at least an order of magnitude greater cumulative multiplies.

Table 9.4 shows that the validation error for LM is slightly better than that of ONT. It is also important to note that Figure 9.2 shows the closeness of the OWO-Newton and HOST3 curves.

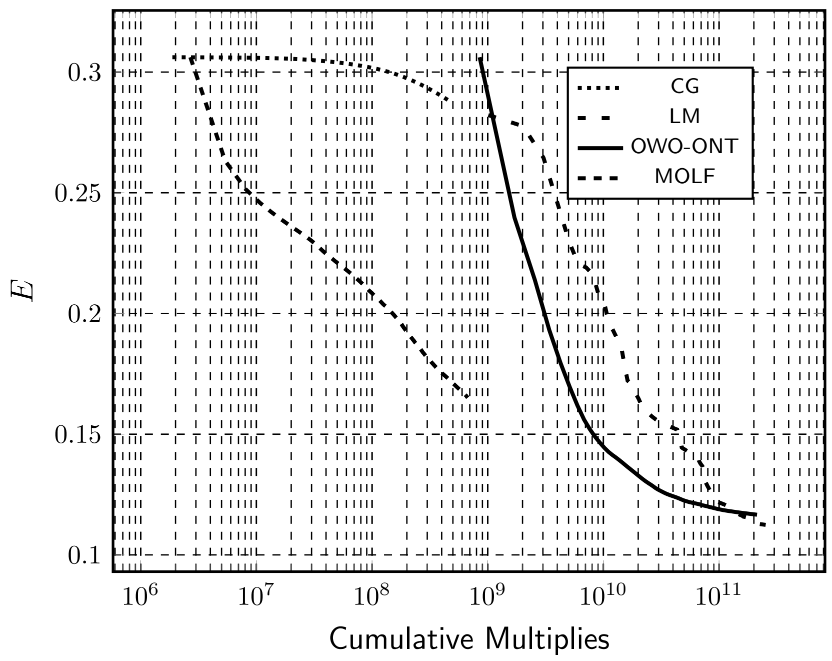

Figure 9.1. Computational cost of 10-fold training of the Remote Sensing dataset featuring OWO-ONT.

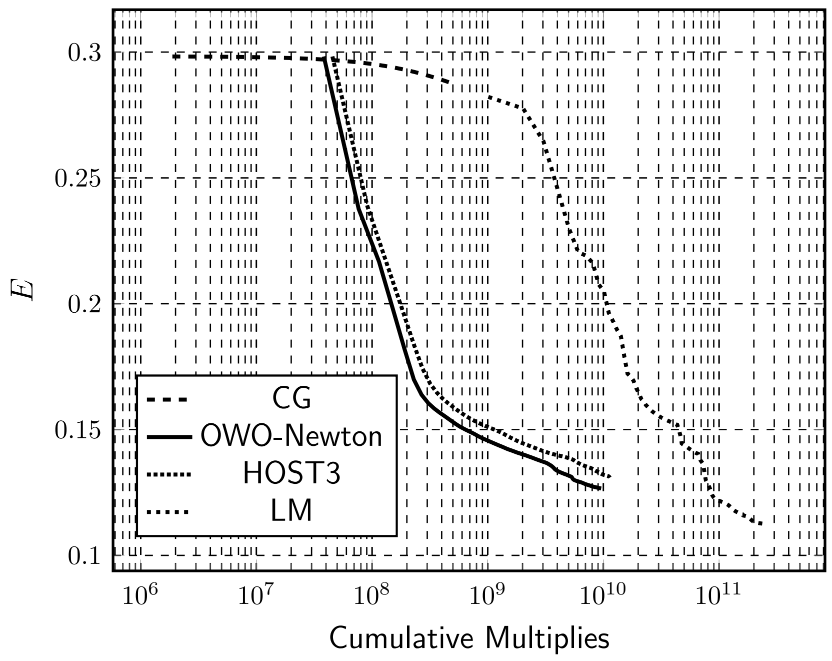

Figure 9.2. 10-fold training in Remote Sensing featuring OWO-Newton and HOST3.

9.2 Remote sensing: Oh7

We demonstrate the new algorithms on the Oh7 dataset [65]. More information on the details can be found in Appendix B and [66]. Oh7 has 10453 training patterns. We train an MLP with 15 hidden units for 200 iterations. Figure 9.3 shows a slightly better training error for LM at the cost of many more multiplies. Figure 9.4 shows very similar performance training performance for OWO-Newton/HOST3 and LM.

Table 9.4 shows that the validation error for LM is slightly better than that of ONT, OWO-Newton and HOST3.

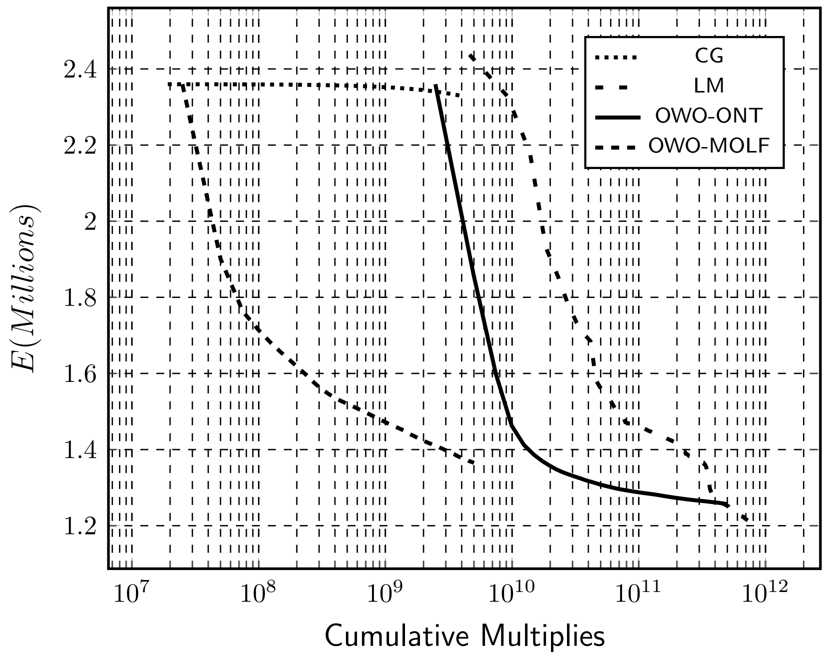

Figure 9.3. 10-fold training of the Oh7 dataset featuring OWO-ONT.

9.3 Prognostics training file

We demonstrate the new training algorithms on the prognostics dataset. The prognostics training file contains parameters that are available in a helicopter's health

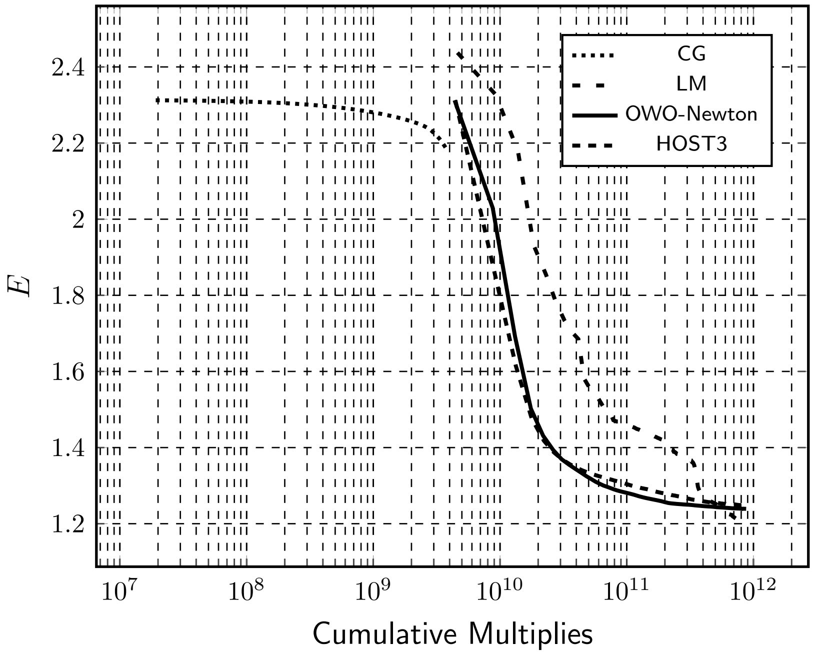

Figure 9.4. 10-fold training of the Oh7 dataset featuring OWO-Newton and HOST3.

usage monitoring system (HUMS)[67]. 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.

Figure 9.5 shows the 10-fold training error of OWO-ONT against reference algorithms OWO-MOLF, CG and LM. Figure 9.6 shows almost identical performance for OWO-Newton and HOST3 which both perform significantly better than LM.

Table 9.4 shows that the training and validation error for new algorithms: OWO-Newton, OWO-ONT and HOST3 is significantly better than that of LM.

9.4 Wine Quality training file

We demonstrate OWO-Newton and HOST3 on the wine quality UCI dataset [68]. The goal is to predict the quality of wine from objective measures [69]. The training file consists of 11 features, and 1 target (scored from 0-12) with 4898 training

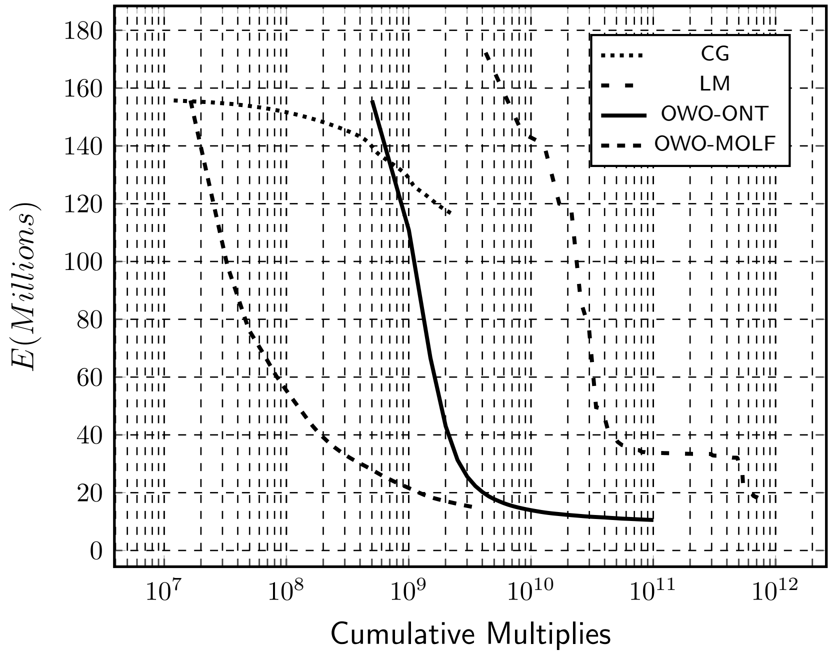

Figure 9.5. 10-fold training of the Prognostics training file featuring OWO-ONT.

patterns. We train an MLP with 15 hidden units for 200 iterations. The 10-fold training error results are shown in Table 9.4. 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 9.7 illustrates the computational burden for this training characteristic. Table 9.4 shows that both ONT and MOLF have a better training and validation error than LM.

Figure 9.8 shows that both MOLF and OWO-ONT train better than LM. It also illustrates that some instability due to round-off error near the minimum.

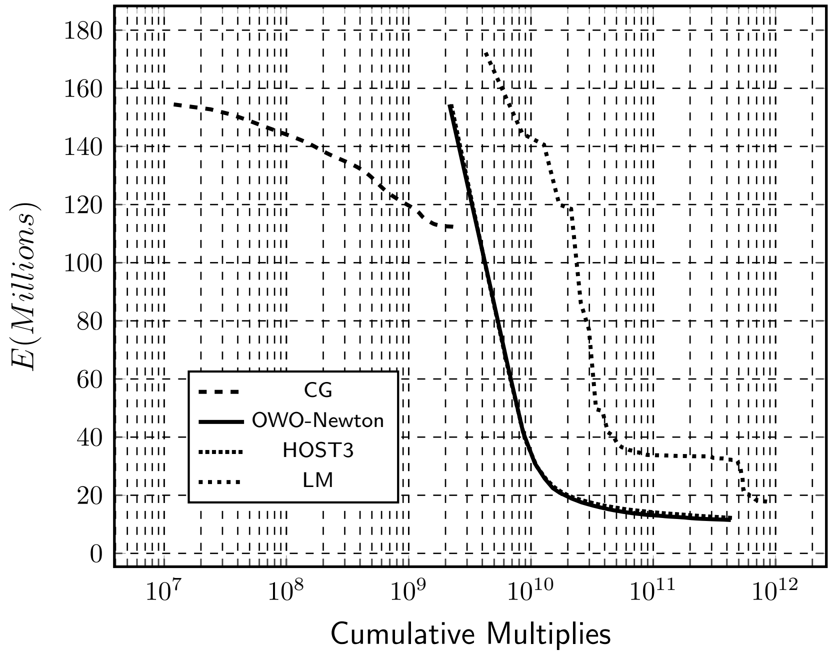

Figure 9.6. 10-fold training of the Prognostics training file featuring OWO-Newton and HOST3.

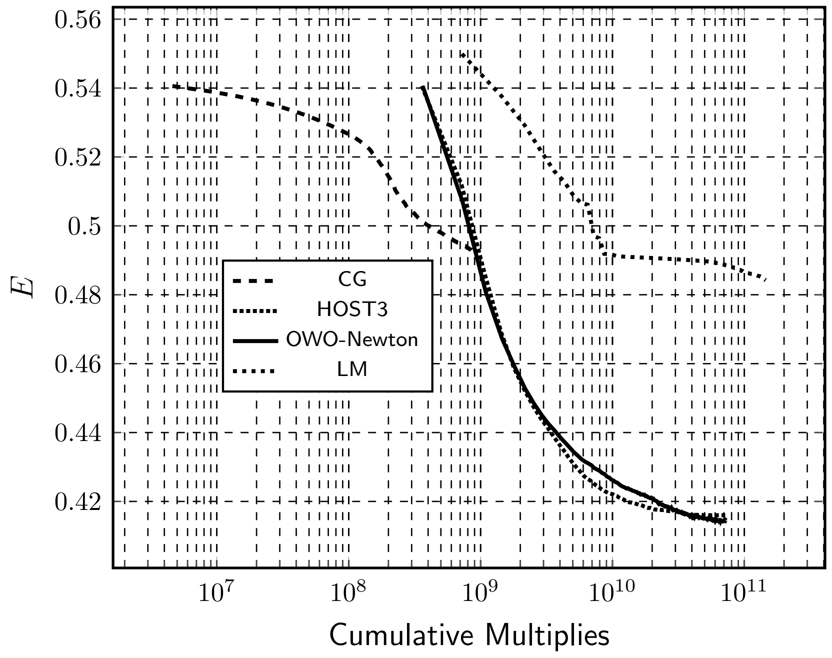

Figure 9.7. 10-fold training in the Wine dataset featuring OWO-Newton and HOST3.

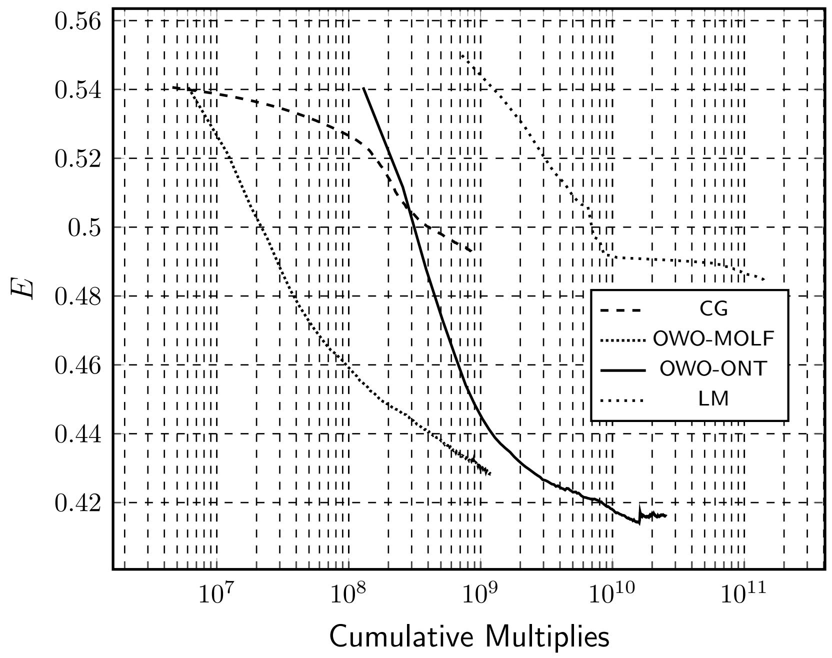

Figure 9.8. 10-fold training in the Wine dataset featuring OWO-ONT.

Table 9.1. Training and Validation Error Summary

Training File	Training Method	Training Error	Validation Error
Remote	ONT	0.1299	0.1406
	LM	0.1124	0.1375
	MOLF	0.1651	0.1853
	Newton	0.1255	0.1446
	CG	0.2746	0.2847
	HOST	0.1309	0.1515
Oh7	ONT	1.2568	1.4324
	LM	1.2091	1.4684
	MOLF	1.3651	1.4804
	Newton	1.2391	1.4078
	CG	2.1801	2.2082
	HOST	1.2466	1.4454
Prognostics	ONT	1.04×10^7	1.37×10^7
	LM	1.7648×10^7	1.75×10^7
	MOLF	1.5124×10^7	1.7586×10^7
	Newton	1.15×10^7	1.48×10^7
	CG	1.1618×10^8	1.1937×10^8
	HOST	1.2197×10^7	1.4844×10^7
Wine	ONT	0.4142	0.5028
	LM	0.4842	0.523
	MOLF	0.4277	0.5094
	Newton	0.3992	0.5336
	CG	0.5248	0.5507
	HOST	0.4145	0.5127

CHAPTER 10

CONTRIBUTIONS AND CONCLUSIONS

10.1 Contributions

We have discussed several second order MLP training problems. Interesting analysis and effective training algorithms have arisen from that discussion. We have also used equivalent network theory to extend the concept of affine invariance to neural networks and introduced the concept of partial affine invariance to neural networks. The resulting algorithms perform very well when compared to the reference training algorithms in popular use today.

10.1.1 Analysis of Newton Training for the MLP

We have demonstrated the implicit assumptions made when training an MLP using Newton's method. The assumptions are that there is an underlying piecewise affine model for an MLPs output. Training all networks simultaneously with Newton's method results in erroneous cross terms in the output that violate the assumptions.

10.1.2 OWO-Newton Algorithm

OWO-Newton is a multi-step second order training algorithm that is partially affine invariant. It has comparable training and validation performance to LM but with fewer multiplies. It features training the input and output weights separately.

10.1.3 OWO-ONT Algorithm

OWO-ONT is a multistep second order training algorithm that is partially affine invariant. It builds on equivalent network theory and the MOLF algorithm to achieve comparable training and validation performance to LM but with fewer multiplies. It features training the output weights with OWO and then transforming a chosen descent direction into direction that yields better performance.

10.1.4 HOST Algorithm

In OWO-Newton we have shown that we can train the input weights with Newton's algorithm. By observing that OWO is Newton's algorithm for the output weights, we can combine both into one Newton step. We have also developed a new line search algorithm that takes into account the relative scales of the weights. Finally, we have shown that the HOST algorithm is partially affine invariant.

10.2 Future Work

In order to take advantage of the vast amount of research in the area of Deep Learning the new second order algorithms will need to be implemented and analyzed for use in multiple hidden layer MLPs.

The work in this dissertation focuses on the regression problem, but an obvious extension would be to the classification problem. In doing so, an objective function other than the MSE error function may be used with great success.

APPENDIX A

OPTIMIZATION FUNDAMENTALS

First order optimization methods

In this section we cover optimization methods that require only first derivative information. These algorithms typically have slow convergence, but are useful building blocks for more complex techniques.

Gradient Descent

Gradients of the error function with respect to network weights are calculated and then the weights are updated after a line search. In some cases, a learning factor of unity is assumed. Note that though the negative gradient is the direction of fastest descent, but might not always point to the minimum of the error surface.

Conjugate Gradient

Conjugate Gradient algorithm or CG was invented in 1952 by Hestenes and Stiefel[70] and is one of a family of algorithms known as Krylov methods. CG is an iterative method that can be used to minimize equations of the form

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \tag{A.1}$$

When $\mathbf{A}$ is symmetric and positive definite this minimization is equivalent to solving $\mathbf{A}\mathbf{x} = \mathbf{b}$. The gradient of (A.1), $\nabla E(\mathbf{w})$ is simply $\mathbf{A}\mathbf{w} - \mathbf{b}$ and when set to zero we see that $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$. The caveat that $\mathbf{A}$ be symmetric positive definite is important because it ensures that E has a global minimum. Of course, in the case that $\mathbf{A}$ is not positive definite then it will be singular and thus noninvertible. Even in the indefinite case, CG can arrive at a solution and thus it provides a more robust means for solving for $\mathbf{w}$ than computing $\mathbf{A}^{-1}$.

We now discuss the concepts and mechanics of CG. CG works in a similar fashion to most iterative algorithms by computing a set of direction vectors

$\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$. On the first iteration, $\mathbf{p}_1$ is taken to be $-\nabla E(\mathbf{w})$. Each subsequent iteration produces a new direction vector which is a linear combination of the previous direction vectors in the set and the current gradient. The vectors have a unique property called A-Conjugacy. Two vectors, $\mathbf{p}_i$ and $\mathbf{p}_j$ are A-Conjugate to each other if $\mathbf{p}_i \mathbf{A} \mathbf{p}_j = 0$. CG converges to the minimum in N iterations. Because neural network problems are in general non-convex we will not have obtained the minimum in N iterations. Thus in practice we should impose alternate stopping criteria. A popular choice is $||\mathbf{g}||_2^2 < \epsilon$: the gradient having a small 2-norm.

Line search subproblem

A very important part of solving an iterative optimization problem is the line search subproblem. The goal is to minimize the objective function along an ray. Specifically we would like to calculate:

$$\underset{z \in \mathbb{R}^+}{\text{minimize}} \quad E(\mathbf{W} + z\mathbf{D}) \tag{A.2}$$

If $\mathbf{D}$ is a descent direction, we are guaranteed a reduction in the objective function. Figure A.1 is an illustration taken from one iteration of training an MLP. We must numerically determine the minimum and this can be costly. The function is clearly neither even nor is it quadratic.

There are several techniques that can be used to find the bottom of the bowl illustrated in Figure A.1 in this one dimensional minimization problem: golden search, fibonacci search, bold driver, Newton-Raphson and backtracking just to name a few.

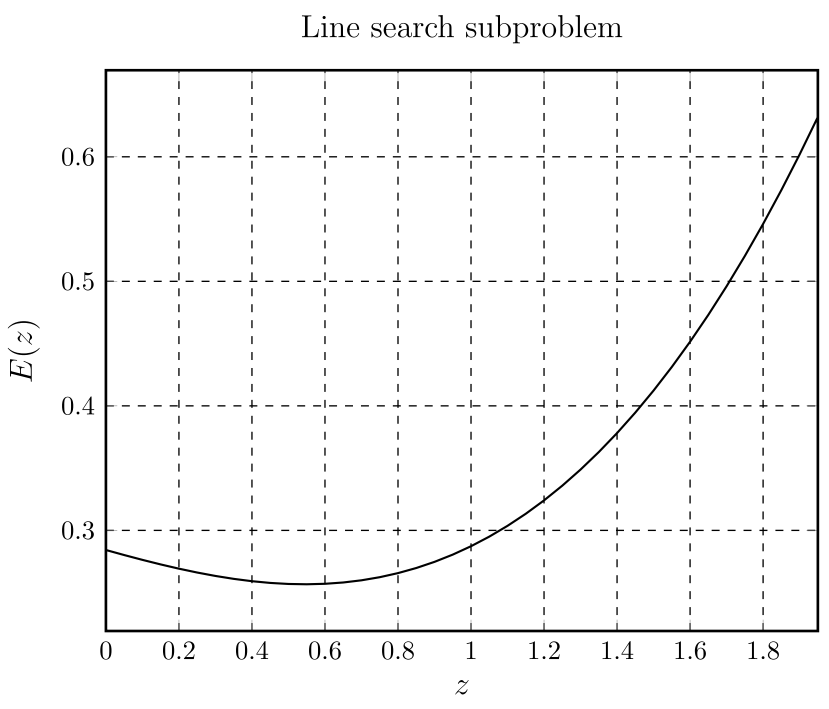

Figure A.1. Illustration of $E(z)$ in one iteration in training..

Second order optimization methods

This section outlines two popular second order optimization algorithms, Newton's Method and Levenberg-Marquardt. Second order algorithms, algorithms that require calculation of a Hessian, are the focus of this proposal.

Newton's Method

Newton's method is one of the most successful unconstrained optimization algorithms and has wide applicability to linear and nonlinear objective functions. Newton's method locally approximates the objective function at every iteration by a quadratic function[64]. This can be done using a Taylor series approximation near the current $\mathbf{w}$.

Newton's algorithm is a recursive algorithm where the next iterate is calculated as:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \mathbf{H}(\mathbf{w}^k)^{-1}\mathbf{g}^k \tag{A.3}$$

where $\mathbf{H}$ and $\mathbf{g}$ are the Hessian and gradient of the objective function respectively. The quantity $\mathbf{H}(\mathbf{w}^k)^{-1}\mathbf{g}^k$ is called the Newton direction and should be calculated by solving linear equations instead of inverting the Hessian:

$$\nabla^2 E(\mathbf{w})\Delta\mathbf{w} = -\nabla E(\mathbf{w}) \tag{A.4}$$

so that (A.3) becomes:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \Delta\mathbf{w}^k$$

The Newton direction, $\Delta\mathbf{w}$, is not guaranteed to produce a descent direction, particularly when $\mathbf{w}$ is far from the minimum. Also, given that in general our objective function is not quadratic in the variables, in practice we will need to solve a line search sub-problem. This is called the Damped Newton's Method.

Newton's algorithm has a very well defined stopping criterion. We may stop when a quantity called the Newton decrement has fallen below a chosen threshold. Of course, if the application requires we may terminate at some maximum iteration number.

$$\lambda^2(x) = \nabla E(\mathbf{w})^T \nabla^2 E(\mathbf{w})^{-1} \nabla E(\mathbf{w}) \tag{A.5}$$

These steps are summarized below:

```
loop
    Compute the gradient and Hessian of the objective function
    Compute the Newton step $\Delta\mathbf{w}$ as in (A.4)
    Calculate the Newton decrement $\lambda^2$ as in (A.5)
    if $\lambda^2 < \epsilon$ then
        STOP
    end if
    Line search: use iterative numerical method to calculate $z$
    $\mathbf{w} \leftarrow \mathbf{w} + z\Delta\mathbf{w}$
end loop
```

Algorithm A.1: Damped Newton's Method

An important property of Newton's algorithm is that it is affine invariant. That is, the descent direction produced by Newton's method is transformed accordingly. The proof below is adapted from Boyd and Vandenberge[55]:

Lemma A.1. *Let $x \in \mathbb{R}^N$ and $f\colon \mathbb{R}^N \to \mathbb{R}^M$. The Newton step, Δx_{nt} is affine invariant.*

Proof. Let $A \in \mathbb{R}^{N\times M}$ be a non-singular matrix and $y = Ax$ and $g(x) = f(Ax)$ then

$$\begin{aligned} \nabla g(x) &= A^T \nabla f(y) \qquad \text{(A.6)} \\ \nabla^2 g(x) &= A^T \nabla^2 f(y) A \end{aligned}$$

The Newton step for g is

$$\begin{aligned}\Delta x &= -(\nabla^2 g(x))^{-1}\nabla g(x) \qquad (A.7)\\ &= -A^{-1}(\nabla f(y))^{-1}A^{-T}A^{T}\nabla f(y)\\ &= -A^{-1}(\nabla f(y))^{-1}\nabla f(y)\\ &= A^{-1}\Delta y\end{aligned}$$

and $x + \Delta x = A(y + \Delta y)$ $\square$

Levenberg-Marquardt

If the Hessian matrix is not positive definite then it is singular and the $\mathbf{\Delta w}$ direction may not make sense as there are many possible solutions to (A.4).

Levenberg[51] and Marquardt [71] showed that the addition of a number to the diagonal of the Hessian $\mu\mathbf{I}$ ensures a descent direction for a large enough μ and makes the Hessian positive definite. The choice of the parameter μ makes the solution of (A.4) vary somewhere between a pure Newton's step ($\mu = 0$) to steepest descent (μ very large).

```
**loop**
    Compute the gradient and Hessian, $\mathbf{H}$ of the objective function
    Choose $\mu$
    Modify the Hessian: $\mathbf{H} \leftarrow \mathbf{H} + \mu\mathbf{I}$
    Compute the Newton step $\Delta\mathbf{w}$ (A.4)
    Calculate the Newton decrement $\lambda^2$ (A.5)
    **if** $\lambda^2 < \epsilon$ **then**
        STOP
    **end if**
    Line search: Choose $z$ according to many existing algorithms
    $\mathbf{w} \leftarrow \mathbf{w} + z\Delta\mathbf{w}$
**end loop**
```

Algorithm A.2: Levenberg-Marquardt Algorithm

APPENDIX B

TRAINING FILE DESCRIPTIONS

B.1 Remote Sensing: Twod

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 inhomogeneous irregular layer above a homogeneous half space from back scattering measurements.

The training data file contains 1768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform pdf.

Table B.1. Features for the Remote Sensing: Twod

Feature		**Target**
Polarization	Incident Angle	
V	10°	effective permittivity (ϵ)
H	10°	upper surface height ($k\sigma_1$)
V	30°	lower surface height ($k\sigma_2$)
H	30°	normalized upper surface correlation length (kL_1)
V	50°	normalized lower surface correlation length (kL_2)
H	50°	optical depth (τ)
V	70°	single scattering albedo (ω)
H	70°	

Table B.2. Features for training file Remote Sensing: Oh7

Feature			**Target**
Polarization	Band	Incident Angle	
HH	L	30°	correlation length
	L	40°	dielectric constant
	C	10°	rms height
	C	30°	
	C	40°	
	C	50°	
	C	60°	
	X	30°	
	X	40°	
	X	50°	
VV	L	30°	
	L	40°	
	C	10°	
	C	30°	
	C	40°	
	C	50°	
	C	60°	
	X	30°	
	X	40°	
	X	50°	

Remote sensing: Oh7

This training data file consists 10453 patterns of 20 inputs and 3 outputs and consists of polarimetric radar measurements for bare soil surfaces under a variety of roughness and moisture conditions at $L-$, $C-$ and $X-$band frequencies (center frequencies of 1.25, 4.75, and 9.5 GHz, respectively) at incidence angles ranging from 10° to 70°. The goal is to determine, from the soil backscatter data, characteristics of the surface such as the rms height, the correlation length and the dielectric constant.

Prognostics

The prognostics data file consists of parameters that are available in the basic health usage monitoring system (HUMS), plus some others. The data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995. The prognostics data set consists of 17 input features and 9 target parameters described as follows:

Table B.3. Features for Prognostics

Features	**Targets**
CG F/A load factor	fore/aft cyclic boost tube oscillatory axial load (OAL)
CG lateral load factor	lateral cyclic boost tube OAL
CG normal load factor	collective boost tube OAL
pitch attitude	main rotor (MR) pitch link OAL
pitch rate	MR mast oscillatory perpendicular bending sta.
roll attitude	MR yoke oscillatory beam bending sta.
roll rate	MR blade oscillatory beam bending sta.
yaw rate	MR yoke oscillatory chord bending sta.
corrected airspeed	resultant mast bending, sta.
rate of climb	
longitudinal cyclic stick position	
pedal position	
collective stick	
lateral cyclic stick position	
main rotor mast torque	
main rotor mast rpm	
density ratio	

Wine Quality

The wine quality data file consists of features related to white variants of the Portuguese ”Vinho Verde” wine. The wines in this set are not ordered and there are more normal wines than poor or excellent ones. In this file there are 11 features and 1 target, the wine quality. The wine quality is a discrete integer value from 0-12. The detail is listed below

Table B.4. Features for training file Wine Quality

Features	**Targets**
fixed acidity	Quality (0-12)
volatile acidity	
citric acid	
residual sugar	
chlorides	
free sulfur dioxide	
total sulfur dioxide	
density	
pH	
sulphates	
alcohol	

APPENDIX C

PARTIAL DERIVATIVES FOR THE MLP

In this chapter we derive the derivatives of the MSE objective function that is used to train a MLP. The notation in this section is consistent with that used in §1.3.

First derivatives for a MLP

The first partial derivatives with respect to the weights for a fully connected single hidden layer MLP are given in the following equations.

The first derivative with respect to the input weights is

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

where

$$\frac{\partial y_p(i)}{\partial w(j,k)} = w_{oh}(i,j)O'(n_p(j))x_p(k)$$

collecting the terms:

$$\frac{\partial E}{\partial w(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - y_p(i)]w_{oh}(i,j)O'(n_p(j))x_p(k)$$

The first derivative with respect to the output weights is

$$\frac{\partial E}{\partial w_{oh}(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p(j) - y_p(j)]\frac{\partial y_p(j)}{\partial w_{oh}(j,k)}$$

where

$$\frac{\partial y_p(j)}{\partial w_{oh}(j,k)} = O(n_p(k))$$

$$\frac{\partial E}{\partial w_{oh}(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p(j) - y_p(j)]O(n_p(k))$$

Finally, the first derivative with respect to the bypass weights is

$$\frac{\partial E}{\partial w_{oi}(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p(j) - y_p(j)]\frac{\partial y_p(j)}{\partial w_{oi}(j,k)} \tag{C.1}$$

where

$$\frac{\partial y_p(j)}{\partial w_{oi}(j,k)} = x_p(k)$$

combining terms

$$\frac{\partial E}{\partial w_{oi}(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p(j) - y_p(j)]x_p(k)$$

Hessian calculations for a MLP

To use 2nd order training techniques in this proposal we must compute the second partial derivatives. The submatrices are listed in (6.7). Below we give the equations to calculate each block of the MLP full Hessian.

$$\frac{\partial^2 y_p(i)}{\partial w(j,k)\partial w(l,m)} = w_{oh}(i,j)x_p(m)x_p(k)O''(n_p(j)) \tag{C.2}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w(j,k)\partial w(l,m)} &= -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - y_p(i)]\frac{\partial y_p(i)}{\partial w(j,k)\partial w(l,m)} - \frac{\partial y_p(i)}{\partial w(j,k)}\frac{\partial y_p(i)}{\partial w(l,m)}\\ &= -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M} w_{oh}(i,j)x_p(m)x_p(k)O''(n_p(j))\\ &- w_{oh}(i,j)f'(n_p(j))x_p(k)w_{oh}(i,j)O'(n_p(j))x_p(m)\end{aligned} \tag{C.3}$$

(C.2) and (C.3) give us elements of the Hessian submatrix $\mathbf{H}_R$

$$\begin{aligned}\frac{\partial^2 E}{\partial w(j,k)\partial w_{oh}(l,m)} &= -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p(l) - y_p(l)]\frac{\partial y_p(l)}{\partial w(j,k)\partial w_{oh}(l,m)} - \frac{\partial y_p(l)}{\partial w(j,k)}\frac{\partial y_p(l)}{\partial w_{oh}(l,m)}\\ &\approx \frac{2}{N_v}\sum_{p=1}^{N_v} w_{oh}(l,j)O'(n_p(j))x_p(k)O(n_p(m))\end{aligned} \tag{C.4}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w(j,k)\partial w_{oi}(l,m)} &= \frac{2}{N_v}\sum_{p=1}^{N_v}\frac{\partial y_p(l)}{\partial w(j,k)}\frac{\partial y_p(l)}{\partial w_{oi}(l,m)}\\ &= \frac{2}{N_v}\sum_{p=1}^{N_v} w_{oh}(l,j)O'(n_p(j))x_p(k)x_p(m)\end{aligned} \tag{C.5}$$

The mixed partials in (C.4) and (C.5) make up the Hessian submatrix $\mathbf{H}_{oi}$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{oi}(j,k)\partial w_{oi}(l,m)} &= \frac{2}{N_v}\sum_{p=1}^{N_v}\frac{\partial y_p(j)}{\partial w_{oi}(j,k)}\frac{\partial y_p(l)}{\partial w_{oi}(l,m)} \\ &= \frac{2}{N_v}\sum_{p=1}^{N_v} x_p(k)x_p(m)\end{aligned} \tag{C.6}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{oh}(j,k)\partial w_{oh}(l,m)} &= \frac{2}{N_v}\sum_{p=1}^{N_v}\frac{\partial y_p(j)}{\partial w_{oh}(j,k)}\frac{\partial y_p(l)}{\partial w_{oh}(l,m)} \\ &= \frac{2}{N_v}\sum_{p=1}^{N_v} O(n_p(k))O(n_p(m))\end{aligned} \tag{C.7}$$

(C.6) and (C.7) give us elements of the Hessian submatrix $\mathbf{H}_o$

REFERENCES

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[2] D. O. Hebb, "The organization of behavior," in *Neurocomputing: foundations of research*, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, pp. 43–54.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Reviews*, vol. 65, no. 6, pp. 386–408, November 1958.

[4] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. dissertation, Harvard University, Cambridge, MA, 1974.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors.* Cambridge, MA, USA: MIT Press, 1988, pp. 696–699.

[6] R. Fletcher, *Practical methods of optimization; (2nd ed.).* New York, NY, USA: Wiley-Interscience, 1987.

[7] S. Haykin, *Neural Networks and Learning Machines (3rd Edition)*, 3rd ed. Prentice Hall, Nov. 2008.

[8] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *Circuits, Devices and Systems, IEE Proceedings G*, vol. 139, no. 3, pp. 301 –310, jun 1992.

[9] Y. LeCun, L. Bottou, G. Orr, and K. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Orr and K.-R. Müller, Eds. Springer Berlin / Heidelberg, 1998, vol. 1524, pp. 546–546.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, July 1989.

[11] H. White, "Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings," *Neural Networks*, vol. 3, no. 5, pp. 535 – 549, 1990.

[12] S. S. R. Malalur, "A family of robust second order training algorithms," Ph.D. dissertation, The University of Texas at Arlington, Arlington, Texas, 2009.

[13] A. R. Barron, "Approximation and estimation bounds for artificial neural networks," in *Proceedings of the fourth annual workshop on Computational learning theory*, ser. COLT '91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 243–249.

[14] D. Ruck, S. Rogers, M. Kabrisky, M. Oxley, and B. Suter, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," *Neural Networks, IEEE Transactions on*, vol. 1, no. 4, pp. 296 –298, dec 1990.

[15] E. Wan, "Neural network classification: a Bayesian interpretation," *Neural Networks, IEEE Transactions on*, vol. 1, no. 4, pp. 303–305, Dec 1990.

[16] N. Morgan and H. Bourlard, "Neural networks for statistical recognition of continuous speech," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 742 –772, May 1995.

[17] S. Nazeer, N. Omar, and M. Khalid, "Face recognition system using artificial neural networks approach," in *Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on*, feb. 2007, pp. 420 –425.

[18] J. Lin and R. Inigo, "Hand written zip code recognition by back propagation neural network," in *Southeastcon '91., IEEE Proceedings of*, vol. 2, apr 1991, pp. 731–735.

[19] Y. Saifullah and M. Manry, "Classification-based segmentation of zip codes," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 5, pp. 1437 –1443, Sep/Oct 1993.

[20] R. Brause, "Medical analysis and diagnosis by neural networks," *Medical Data Analysis, Springer-Verlag*, vol. 20, pp. 1–13, 2001.

[21] W. G. Baxt, "Use of an artificial neural network for the diagnosis of myocardial infarction," *Annals of Internal Medicine*, vol. 115, no. 11, pp. 843–848, 1991.

[22] G.-P. Economou, C. Spiropoulos, N. Economopoulos, N. Charokopos, D. Lymberopoulos, M. Spiliopoulou, E. Haralambopulu, and C. Goutis, "Medical diagnosis and artificial neural networks: a medical expert system applied to pulmonary diseases," in *Neural Networks for Signal Processing [1994] IV. Proceedings of the 1994 IEEE Workshop*, sep 1994, pp. 482 –489.

[23] S. Polak, A. Skowron, J. Brandys, and A. Mendyk, "Artificial neural networks based modeling for pharmacoeconomics application," *Applied Mathematics and Computation*, vol. 203, no. 2, pp. 482 – 492, 2008.

[24] M. Voultsidou, S. Dodel, and J. Herrmann, "Neural networks approach to clustering of activity in fMRI data," *Medical Imaging, IEEE Transactions on*, vol. 24, no. 8, pp. 987 –996, Aug. 2005.

[25] M. T. Manry, S. J. Apollo, L. S. Allen, W. D. Lyle, W. Gong, M. Dawson, and A. K. Fung, "Fast training of neural networks for remote sensing," *Remote Sensing Reviews*, vol. 9, pp. 77–96, July 1994.

[26] P. M. Atkinson and A. R. L. Tatnall, “Introduction Neural networks in remote sensing,” *International Journal of Remote Sensing*, vol. 18, no. 4, pp. 699–709, March 1997.

[27] T. Kavzoglu and P. M. Mather, “Pruning artificial neural networks: An example using land cover classification of multi-sensor images,” *International Journal of Remote Sensing*, vol. 20, no. 14, pp. 2787–2803, 1999.

[28] M. Egmont-Petersen, D. de Ridder, and H. Handels, “Image processing with neural networks—a review,” *Pattern Recognition*, vol. 35, no. 10, pp. 2279 – 2301, 2002.

[29] M. Akram and A. Usman, “Computer aided system for brain tumor detection and segmentation,” in *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on*, july 2011, pp. 299 –302.

[30] S. M. Bhandarkar, J. Koh, and M. Suk, “Multiscale image segmentation using a hierarchical self-organizing map,” *Neurocomputing*, vol. 14, no. 3, pp. 241 – 272, 1997.

[31] *A review of ANN-based short-term load forecasting models*, 1995.

[32] K. Y. Lee, Y. T. Cha, and J. H. Park, “Short-term load forecasting using an artificial neural network,” *IEEE Transactions on Power Systems*, vol. 7, no. 1, pp. 124–132, 1992.

[33] K. Liu, S. Subbarayan, R. Shoults, M. Manry, C. Kwan, F. Lewis, and J. Naccarino, “Comparison of very short-term load forecasting techniques,” *Power Systems, IEEE Transactions on*, vol. 11, no. 2, pp. 877 –882, may 1996.

[34] T. Florio, S. Einfeld, and F. Levy, “Neural networks and psychiatry: Candidate applications in clinical decision making,” *Australian and New Zealand Journal of Psychiatry*, vol. 28, no. 4, pp. 651–666, 1994.

[35] J. Patra, G. Panda, and R. Baliarsingh, “Artificial neural network-based nonlinearity estimation of pressure sensors,” *Instrumentation and Measurement, IEEE Transactions on*, vol. 43, no. 6, pp. 874 –881, dec 1994.

[36] T. Parisini and R. Zoppoli, “Neural networks for nonlinear state estimation,” *International Journal of Robust and Nonlinear Control*, vol. 4, no. 2, pp. 231–248, 1994.

[37] J. Luxhøj, “An artificial neural network for nonlinear estimation of the turbine flow-meter coefficient,” *Engineering Applications of Artificial Intelligence*, vol. 11, no. 6, pp. 723–734, 1998.

[38] J. Wang and J. Huang, “Neural network enhanced output regulation in nonlinear systems,” *Automatica*, vol. 37, no. 8, pp. 1189 – 1200, 2001, neural Network Feedback Control.

[39] S. Kulluk, L. Ozbakir, and A. Baykasoglu, “Training neural networks with harmony search algorithms for classification problems,” *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 11 – 19, 2012.

[40] C. Oz and M. C. Leu, “American sign language word recognition with a sensory glove using artificial neural networks,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 7, pp. 1204 – 1213, 2011, infrastructures and Tools for Multiagent Systems.

[41] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 5, pp. 455–455, 1992.

[42] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, “Layered neural networks with gaussian hidden units as universal approximations,” *Neural Computation*, vol. 2, no. 2, pp. 210–215, 2012/03/24 1990.

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

[44] M. Manry, X. Guan, S. Apollo, L. Allen, W. Lyle, and W. Gong, "Output weight optimization for the multi-layer perceptron," in *Signals, Systems and Computers, 1992. 1992 Conference Record of The Twenty-Sixth Asilomar Conference on.* IEEE, 1992, pp. 502–506.

[45] M. Sartori and P. Antsaklis, "A simple method to derive bounds on the size and to train multilayer neural networks," *Neural Networks, IEEE Transactions on*, vol. 2, no. 4, pp. 467–471, 1991.

[46] S. Chen, S. Billings, and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[47] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[48] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, 0-0 1989, pp. 593 –605 vol.1.

[49] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550 –1560, oct 1990.

[50] T. Kim, M. Manry, and J. Maldonado, "New learning factor and testing methods for conjugate gradient training algorithm," in *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 3, july 2003, pp. 2011 – 2016 vol.3.

[51] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathematics*, vol. II, no. 2, pp. 164–168, 1944.

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

[53] J. Wille, "On the structure of the Hessian matrix in feedforward networks and second derivative methods," in *Neural Networks,1997., International Conference on*, vol. 3, jun 1997, pp. 1851 –1855 vol.3.

[54] R. Lengellé and T. Denoeux, "Training MLPs layer by layer using an objective function for internal representations," *Neural Networks*, vol. 9, no. 1, pp. 83–97, 1996.

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

[56] S. Malalur and M. Manry, "Multiple optimal learning factors for feed-forward networks," in *SPIE Defense, Security, and Sensing.* International Society for Optics and Photonics, 2010, pp. 77 030F–77 030F.

[57] S. Saarinen, R. Bramley, and G. Cybenko, "Ill-conditioning in neural network training problems," *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 693–714, 1993.

[58] P. P. v. d. Smagt and G. Hirzinger, "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.* London, UK: Springer-Verlag, 1998, pp. 193–206.

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

[60] M. Dawson, A. Fung, and M. Manry, "Surface parameter retrieval using fast learning neural networks," *Remote Sensing Reviews*, vol. 7, no. 1, pp. 1–18, 1993.

[61] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *J. Optim. Theory Appl.*, vol. 109, no. 3, pp. 475–494, Jun. 2001.

[62] D. P. Bertsekas, *Nonlinear Programming.* Belmont, MA: Athena Scientific, 1999.

[63] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16).* Soc for Industrial & Applied Math, 1996.

[64] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization)*, 3rd ed. Wiley-Interscience, Feb. 2008.

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

[66] Y. Oh, K. Sarabandi, and F. Ulaby, "An empirical model and an inversion technique for radar scattering from bare soil surfaces," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 30, no. 2, pp. 370–381, 1992.

[67] M. Manry, C. Hsieh, and H. Chandrasekaran, "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.* IEEE, 1999, pp. 535–544.

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

[69] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.

[70] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, Dec. 1952.

[71] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *SIAM Journal on Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

BIOGRAPHICAL STATEMENT

Melvin D. Robinson was born in Mt. Pleasant, Texas, in 1969, the son of Melvin and Evelyn Robinson. He received the Bachelor of Science degree in Applied Mathematics from the University of Houston-Downtown in 2002. In 2006 and 2013 respectively, he received the Master of Science and Doctor of Philosophy degrees in Electrical Engineering from the University of Texas at Arlington. Dr. Robinson has over 15 years of experience in the Information Technology field. His current research interests are in the areas of machine learning, numerical analysis, numerical optimization, high performance computing.