IMPROVED INITIALIZATION FOR THE
MULTI LAYER PERCEPTRON

By
ABHISHEK VINAY MAINKAR

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

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERISTY OF TEXAS AT ARLINGTON
May 2018
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advising professor Dr. Michael T Manry for his continuous support throughout my master study and related research, for his patience, guidance and immense knowledge. His guidance helped me throughout my research and writing of this thesis.

I would also like to thank Dr. Ioannis D. Schizas and Dr. R Stephen Gibbs for their time, valuable comments and, for being a member of my thesis defense committee.

Finally, I must express my sincere gratitude to my family for their love and support. I dedicate this thesis to my parents, Mr. Vinay Mainkar and Mrs. Medha Mainkar, my brother, Mr. Anup Mainkar and my sister in law Mrs. Purva Mainkar.

April 26, 2018
ABSTRACT

IMPROVED INITIALIZATION FOR THE
MULTI LAYER PERCEPTRON

ABHISHEK MAINKAR
The University of Texas at Arlington, 2018

Supervising Professor: Dr. Michael T. Manry

A Multilayer Perceptron (MLP) neural network is used for solving nonlinear functional problems like function approximation, classification, data processing etc. MLP neural networks are usually trained using back propagation, which is a non-convex optimization problem for most of the loss functions. As there are multiple local minima, non-convex optimization curves generally converge to different optimal points for different initial conditions. So it not only affects the speed of the convergence but optimality as well. Initial parameters of neural networks are as important as the network architecture and initialization has been thoroughly studied in the past. This report discusses the fusion method and modified sigmoid method which are used for network initialization. Both initialization methods discussed in this report are based on the regular Hidden weight optimization – Multiple optimal learning factors (HWO-MOLF) MLP. Due to non-convex optimization, training an MLP for large networks has the possibility of finding local minima instead of the global minima. The network has a possibility to stick at saddle points when minimizing the error function. Both the initialization procedures in this report, try to avoid the likelihood of finding a local minimum. The training experiments and results obtained are demonstrated in this report. We can see that using both the initialization methods for training HWO-MOLF network, helps mitigate local minima problem, hidden units saturation problem, and dependent hidden units.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS................................................................................................................................. 3

ABSTRACT ......................................................................................................................................................... 4

TABLE OF CONTENTS .................................................................................................................................... 5

LIST OF FIGURES ............................................................................................................................................... 7

LIST OF TABLES .................................................................................................................................................. 8

CHAPTER 1 Introduction ..................................................................................................................................... 9

1.1 Machine Learning ....................................................................................................................................... 9

1.2 Artificial Neural Network (ANN) .............................................................................................................. 9

1.2.1 Structure and Components of an Artificial Neural Network .............................................................. 10

1.2.2 Training of ANN .................................................................................................................................. 14

1.2.3 Properties of ANN ............................................................................................................................... 15

1.3 Parameter initialization of Neural Network Model .................................................................................... 15

CHAPTER 2 Multilayer Perceptron Notation and Training ................................................................................ 17

2.1 Structure of the Multilayer Perceptron ..................................................................................................... 17

2.2 Multilayer Perceptron Notation ................................................................................................................. 19

2.3 MLP Training ............................................................................................................................................ 20

2.3.1 Weight Initialization ............................................................................................................................. 21

2.3.2 Net Control [18] .................................................................................................................................. 21

2.3.3 Solving for Output Weights (Output Weight Optimization (OWO)) .................................................. 21

2.3.4 Schimdt Procedure and Orthogonal Least Squares (OLS) for Output Weights ............................... 23

2.3.5 Backpropagation for Solving Input Weights ..................................................................................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.6 Hidden Weight Optimization (HWO)</td>
<td>26</td>
</tr>
<tr>
<td>2.3.7 Multiple Optimal Learning Factor (MOLF)</td>
<td>27</td>
</tr>
<tr>
<td>CHAPTER 3 Problems and Proposed Work</td>
<td>29</td>
</tr>
<tr>
<td>3.1 Problems with Initialization of MLP</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Proposed Methods</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1 Fusion Method</td>
<td>30</td>
</tr>
<tr>
<td>3.2.2 Modified Sigmoid Method</td>
<td>30</td>
</tr>
<tr>
<td>CHAPTER 4 Fusion Method</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Final Algorithm</td>
<td>51</td>
</tr>
<tr>
<td>4.3 Results</td>
<td>51</td>
</tr>
<tr>
<td>CHAPTER 5 Modified Sigmoid Method</td>
<td>52</td>
</tr>
<tr>
<td>5.1 Algorithm</td>
<td>52</td>
</tr>
<tr>
<td>5.2 Result</td>
<td>54</td>
</tr>
<tr>
<td>CHAPTER 6 Conclusion</td>
<td>55</td>
</tr>
<tr>
<td>Appendix A Description of datasets</td>
<td>56</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>59</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1: Radial Basis Function Network .................................................................................. 12
Figure 1.2: Single Layer Perceptron Network ............................................................................. 13
Figure 1.3: MLP with Single Hidden Layer ................................................................................. 13
Figure 1.4: Supervised Training Method ...................................................................................... 14
Figure 2.1: MLP with single hidden layer .................................................................................... 17
Figure 2.2: Nonlinear Activation Functions ................................................................................... 18
Figure 2.3: Sigmoid Activation Function ....................................................................................... 19
Figure 4.1: Histogram of Error without using Fusion Method ....................................................... 32
Figure 4.2: Histogram of Error with Fusion Method by Average of 3 NNs ................................. 33
Figure 4.3: Histogram of Error With Fusion Method by Combination of 3 NNs ....................... 34
Figure 4.4: Histogram of Error with Different NN Initialization Techniques ............................. 35
Figure 4.5: Histogram of Error with 0 iteration for initial training ............................................... 37
Figure 4.6: Histogram of Error with 1 iteration for initial training ............................................... 38
Figure 4.7: Histogram of Error with 5 iterations for initial training ............................................ 39
Figure 4.8: Histogram of Error with 10 Iterations for Initial Training ......................................... 40
Figure 4.9: Histogram of Error with 25 Iterations for Initial Training ......................................... 41
Figure 4.10: Histogram of Error with 50 Iterations for Initial Training ....................................... 42
Figure 4.11: Histogram of Error with 75 Iterations for Initial Training ....................................... 43
Figure 4.12: Histogram of Error with 100 Iterations for Initial Training ...................................... 44
Figure 4.13: Histogram of Error with 200 Iterations for Initial Training ..................................... 45
Figure 4.14: Histogram of Error with 300 Iterations for Initial Training ...................................... 46
Figure 4.15: Histogram of Error with 400 Iterations for Initial Training ..................................... 47
Figure 4.16: Histogram of Error with 499 Iterations for Initial Training ..................................... 48
Figure 4.17: Mean Error with Different Iterations ....................................................................... 50
Figure 4.18: Std Deviation of Error with Different Iterations ..................................................... 50
LIST OF TABLES

Table 4.1 - Error Analysis for Fusion Method using HWO-MOLF ...................................................... 36
Table 4.2 - Error Analysis for Number of Iterations before Fusion into Single Network using HWO-MOLF ........................................................................................................................................................................... 49
Table 4.3 - Performance Comparison with Fusion Method and without Fusion Method .................... 51
Table 5.1 - Performance Comparison with Modified Sigmoid and without Modified Sigmoid .......... 54
CHAPTER 1

Introduction

1.1 Machine Learning

Machine learning is a field of computer science that gives computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed [1]. Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people [27]. Machine learning is most successful subfield of AI, and has enjoyed remarkable success in recent days on a wide range of tasks – e.g. spam filtering [28], recommender systems [29], image classification [30], speech recognition [31], and anomaly detector [32].

Machine learning algorithms are broadly divided into two categories,

i. Supervised learning [40] - The computer is presented with example inputs and their desired outputs, and the goal is to learn a general rule that maps inputs to outputs. Examples of supervised learning algorithms are Decision Tree [33], Random Forest [34], KNN [35], Logistic Regression [36], SVM [37], Artificial neural network (ANN) [24] etc.

ii. Unsupervised learning [40] - No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Examples of unsupervised algorithms are Apriori algorithm [38], K-means [39] etc.

In recent years, due to increase in computation power, large amounts of data and efficient algorithms, artificial neural networks (ANN) have become a dominant machine learning algorithm in the industry.

1.2 Artificial Neural Network (ANN)

Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and use the results to identify cats in other images. They do this without any a priori knowledge about cats, e.g., that they have fur, tails, whiskers and
cat-like faces. Instead, they evolve their own set of relevant characteristics from the learning material that they process. [2]

Collection of connected units or nodes called artificial neurons form ANN. Each connection between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it. The connections between neuron have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning. [3] ANNs have been used on a variety of tasks, including computer vision [41], speech recognition [31], machine translation [42], social network filtering [43], playing board and video games [44] and medical diagnosis [45].

1.2.1 Structure and Components of an Artificial Neural Network

Components of an Artificial Neural Network [2]

Neurons

A neuron with label $j$ receiving an input $p_j(t)$ from predecessor neurons consists of the following components:

- an activation $a_j(t)$ depending on a discrete time parameter,
- possibly a threshold $\theta_j$, which stays fixed unless changed by a learning function,
- an activation function $f$ that computes the new activation at a given time $t + 1$ from $a_j(t)$, $\theta_j$ and the net input $p_j(t)$ giving rise to the relation
  \[ a_j(t + 1) = f(a_j(t), p_j(t), \theta_j) \]
- and an output function $f_{out}$ computing the output from the activation
  \[ o_j(t) = f_{out}(a_j(t)) \]

Often the output function is simply the Identity function. An input neuron has no predecessor but serves as input interface for the whole network. Similarly an output neuron has no successor and thus serves as output interface of the whole network.
Connections and weights

The network consists of connections, each connection transferring the output of a neuron i to the input of a neuron j. In this sense i is the predecessor of j and j is the successor of i. Each connection is assigned a weight \( w_{ij} \).

Propagation Function

The propagation function computes the input \( p_j(t) \) to the neuron j from the outputs \( o_i(t) \) of predecessor neurons and typically has the form

\[
p_j(x) = \sum_i o_i(t)w_{ij}
\]

Learning rule

The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This learning process typically amounts to modifying the weights and thresholds of the variables within the network.

Structures of an Artificial Neural Networks

Structural arrangement and training procedure are the important factors for classification of Neural Networks. The training algorithm for neural network would differ from network to network based on structural arrangement of neural network. Some of the most commonly used neural networks are mentioned below.
1. **Radial Basis Function Networks**

Radial basis function (RBF) network [4, 5] is a type of neural network which uses the radial basis functions as activation functions. Distance from the input vector $\boldsymbol{x}$ to a center vector $\boldsymbol{m}_k$ is radial basis function.

$$\phi (\boldsymbol{x}) = \phi (||\boldsymbol{x} - \boldsymbol{m}_k||)$$

Radial basis function networks have neurons with nonlinear RBF activations in the hidden layer. For higher dimensional hidden space i.e. when hidden layer has more neurons than the number of inputs, Radial basis functions are the best choice. Such RBF networks can better approximate a smooth input-output mapping. [6]

2. **Single Layer Perceptron**

The simplest kind of neural network is a Single-Layer Perceptron [7,8], which consist of no hidden layer i.e. input layer is directly connected to output layer. Each input neuron is connected to each output neuron by real valued weight. At each output neuron, sum of the products of the weights and inputs is calculated. The single layer perceptron has a unidirectional flow of data, since the data always flows forward and strictly one directional.
3. Multilayer Perceptron (MLP) Neural Networks

A multilayer perceptron (MLP) [9,10] is a feedforward artificial neural network model that maps input data space into desired output space. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Each node in MLP (except input nodes) performs non-linear operation. Due to non-linear operation at each node MLP can distinguish data that is not linearly separable. So, MLP networks are like linear networks but with non-linearity added at each node. [11] MLP is trained using supervised learning algorithm called back propagation. [12, 13]
1.2.2 Training of ANN

Learning process is one of most fundamental thing in Neural Network. Training of ANN differs based on application, training data size, number of parameters etc. After creating Neural Network model based on application, the next step is to train the model based on different learning algorithms like gradient descent [46], stochastic gradient descent [47], conjugate gradient [48], RMSProp [49], AdaGrad [50], momentum [51], Adam [52] etc. Learning of neural network is a process to change parameters of the network so that the model can map inputs of training data very close to outputs of training data. In process of learning, activation function of hidden neuron is kept constant. Learning of neural can be supervised in which correct choice of output is provided in training data or, unsupervised in which no output is provided. In this thesis, training of ANN is done by using supervised learning algorithm.

Minimum training error while training neural network does not give best performance in a test environment. This can be due to overfitting of the model to training data. So, training is about finding the best parameters for the model by using training data, so that testing error is minimum. And, testing is evaluating parameters learned while training, to unseen data. This means that parameters of model found out during training, which give minimum test error are selected as final parameters of the model.

**Supervised Learning**

In supervised learning, inputs and expected outputs are given in data file. The inputs are then fed to the network, and the outputs of the network are compared with the desired output in training data. Error in the comparison is fed back to the network from output to input, and the parameters of the network are adjusted to minimize the error between output of network, and the desired output of the network. This process is repeated multiple times till minimum error is reached.

![Figure 1.4: Supervised Training Method](image)
Supervised learning can be used for both classification and regression [53]. Supervised learning is giving best results in machine learning domain, to learn object behavior in certain tasks. Since this may produce input to output mapping faster and more accurate than humans, the machines trained with supervised learning perform better [54].

1.2.3 Properties of ANN

An ANN is composed of artificial neurons or nodes connected to form a network which basically uses a mathematical or a computational model for processing information. It is capable of changing its structure depending on the information transmitted through the system. Due to this property, ANN can be used to model complex relationships about input and output, and to find patterns in given data.

**Representation Power** – Every Boolean function can be represented exactly by some network with two layers of units. The number of hidden units required may grow exponentially with the number of network inputs.

**Universal Approximation Function** – Any function can be approximated to arbitrary accuracy by a network with three layers of units [55]. A three-layer network can approximate any reasonable function to any degree of required precision as long as the hidden layer can be arbitrarily large.

**Feature Extractor** – ANN particularly Convolution neural network (CNN) are extremely good at finding features in data. This property has reduced a lot of data analysis time and manual finding of the best feature in data set.

Due to these properties, neural networks are extremely successful machine learning algorithms in modern times.

1.3 Parameter initialization of Neural Network Model

In the training process of a neural network, after creation of the model of a neural network, parameters of the model are initialized by using random number generator (normal distribution). The problem with random weight generation is that in this non-convex optimization problem of neural network there is a chance that the model training might get stuck in a local minima instead of reaching the global minima. This happens because parameters are randomly assigned initially on an optimization curve. While training, parameters get stuck to local minima which is near to initialized parameter values on multi-dimensional non-convex
optimization curve. This will not reduce training error further. There are other problems with random initialization like hidden neuron saturation and dependent hidden neuron.

In this thesis, better parameter initialization techniques than random initialization are suggested. In chapter 2, Multilayer Perceptron (MLP) notation and training are discussed. In chapter 3, problems with parameter initialization and proposed work are given. Chapter 4 discusses about the algorithm and results of fusion method of parameter initialization. Chapter 5 describes the algorithm and the results of modified sigmoid method. Lastly, chapter 6 is the conclusion of the thesis.
CHAPTER 2
Multilayer Perceptron Notation and Training

2.1 Structure of the Multilayer Perceptron

Figure 2.1 shows the structure of the multilayer perceptron with a single hidden layer. The structure of the multilayer perceptron in the figure consists of an input layer, one or more hidden layers and one output layer. In general, multilayer perceptron consists of one or more hidden layers. Each input node is connected to each node in first hidden layer, each node in the first layer is connected to each node in second layer and so on till output is generated. In modern literature, a multilayer perceptron with more than one hidden layer is called a Deep Learning architecture. [14] In this thesis, all the study is done for multilayer perceptron with single hidden layer. Single hidden layer multilayer perceptrons are powerful enough to approximate any continuous function. [15]

![Figure 2.1: MLP with single hidden layer](image)

Figure 2.1: MLP with single hidden layer
The multilayer perceptron network is a highly hierarchical system since every layer is connected to the next layer. Also, multilayer perceptron has highly connected topology, since every input is connected to all the nodes in the first hidden layer, and every unit in the hidden layers is connected to all the nodes in the next layer. In a multilayer perceptron, as data flows from input layer to output layer, traversing through each hidden layer connected in sequence, it is called a feed forward network.

The hidden layer neuron performs multiplication of input values and weight matrix. Then adds threshold $\theta$ to form the net function, followed by an application of a nonlinear activation function $f(\text{net})$ to the net function. There are different nonlinear activation functions available for multilayer perceptron, some of them are shown in figure 2.2 [16]

![Activation functions](image)

**Figure 2.2: Nonlinear Activation Functions**

Sigmoid activation function is used as the activation function in this report for all the analysis.

The sigmoid activation function is given as follows

$$ O_p = f(n_p) = \frac{1}{1 + e^{-n_p}} \quad (2.1) $$
2.2 Multilayer Perceptron Notation

In the training data \( \{ x_p, t_p \} \) for a fully connected MLP, \( x_p \) represents \( p \)th input vector having dimension \( N \) and, \( t_p \) represents \( p \)th output vector having dimension \( M \). Let the input vectors be augmented by an extra element \( x_p(N + 1) \) where \( x_p(N + 1) \) such that \( x_p = [x_p(1), x_p(2), \ldots, x_p(N + 1)]^T \). And \( p \) is a pattern number which varies from 1 to \( N \).

Additional parameters are \( w_k(k, n) \), \( w_{oh}(k, n) \) and \( w_{oi}(i, n) \). Input weights \( w_k(k, n) \) connect the \( n \)th input to the \( k \)th hidden unit. Output weights \( w_{oh}(i, k) \) connect the \( k \)th hidden units activation \( O_p(k) \) to the \( i \)th output \( y_p(i) \), which has a linear activation. The bypass weight \( w_{oi}(i, n) \) connects the \( n \)th input to the \( i \)th output.

For the \( p \)th pattern, the \( k \)th hidden units net function \( n_p(k) \) is given as

\[
n_p(k) = \sum_{n=1}^{N+1} w(k, n)x_p(n)
\]

Which when written in matrix notation could be given as

\[
n_p = W \cdot x_p
\]

Here \( n_p \) denotes the \( N \) dimensional column vector of a net function values and \( W \) is \( N \) by \( (N+1) \). For the \( p \)th pattern, the \( k \)th hidden units activation is denoted as \( O_p(k) \) where \( O_p(k) = f(n_p(k)) \) and \( f \) denotes the hidden layer activation.

![Figure 2.3: Sigmoid Activation Function](image)
For the \( p \)th pattern, the \( i \)th element \( y_p(i) \) of the \( M \) dimensional output vector \( y_p \) is given as

\[
y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n) x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k) O_p(k)
\]  

(2.4)

This can also be denoted in matrix form as

\[
y_p = W_{oi} \cdot x_p + W_{oh} \cdot O_p
\]  

(2.5)

where, \( O_p \) is the \( N_h \) dimensional hidden unit activation vector.

The error used in training MLP is the mean – squared error (MSE), which is

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

(2.6)

### 2.3 MLP Training

Training a neural network is a non-convex optimization problem in which weights are adjusted in such a way that error in desired output and actual output is minimum. Learning algorithm is the most important thing in neural network, because it will decide the speed of convergence and ability to find global minima instead of local minima. There are different techniques for optimization like stochastic gradient descent (SGD), SGD with momentum, Adagrad, RMSProp etc. In this report, training algorithm used for MLP is called HWO – MOLF.

Training algorithms can be classified into two types as follows,

a) Two-stage training – Input and output weights are trained alternately  
b) One-stage training – All weights are updated simultaneously

HWO-MOLF algorithm uses OWO-BP [17], which denotes Output Weight Optimization – Backpropagation.

This OWO-BP is a two-stage algorithm. Steps in two-stage algorithm are –

1) Weight Initialization – Randomize input weights, OWO for output weights.  
2) Repeated iterations in which
a) Use steepest descent (BP) to modify input weights
b) OWO for output weights

2.3.1 Weight Initialization

If some inputs have much larger standard deviation than others, they can dominate the training, even if they are relatively useless. By calculating input’s standard deviation, input weights can be normalized later. Let ran(i) denote a zero-mean Gaussian random number with variance of 1, where i denotes self-changing random number seed. These randomly generated weights are divided by the input standard deviation, thus removing dominance of large variance inputs.

2.3.2 Net Control [18]

Training of input weights is strongly dependent on the slopes of hidden unit activation functions in response to inputs. Training of a weight ceases if the unit it feeds into has an activation function derivative of zero for all patterns. Therefore, adjust mean and standard deviations of all hidden unit net functions so that they have values of \( m_d = 0.5 \) and \( \sigma_d = 1 \). This control is accomplished as follows –

a) For hidden layer, make a pass through the training data and calculate \( m(2, k) \) and \( \sigma(2, k) \) which are respectively hidden layer net function mean and standard deviations for the \( k \)th unit.

b) For the \( k \)th hidden unit, multiply the threshold and all incoming weights by \( \frac{\sigma_d}{\sigma(2,k)} \) to adjust the net function standard deviation to the desired value.

c) For the \( k \)th hidden unit, update the threshold as

\[
\theta(2,k) = \theta(2,k) - m(2,k) \cdot \frac{\sigma_d}{\sigma(2,k)} + m_d
\]  

(2.7)

2.3.3 Solving for Output Weights (Output Weight Optimization (OWO))

At this point, we have determined the initial input weights and therefore the initial network basis function. We can now find the output weights [19 – 21].

\[
y_p = W_{ol} \cdot x_p + W_{oh} \cdot O_p
\]  

(2.8)
\[ y_p = W_o \cdot X_p \] \hspace{1cm} (2.9)

where,

\[ W_o = [W_{0l} : W_{0h}] \]

\[ X_p = [x_p^T, O_p^T]^T \]

L is the total number of basis functions i.e. \( N+N_h+1 \). The basis functions are as follows,

\[ X(n) = x(n) \text{ for } n \text{ between } 1 \text{ and } N \]

\[ X(N+1) = 1 \]

\[ X(N+1+k) = O_k \text{ for } k \text{ between } 1 \text{ and } N_h \]

We have M sets of L equations in L unknowns which leads us to,

\[ R \cdot W_o^T = C \] \hspace{1cm} (2.10)

where \( R \) is the Autocorrelation matrix of size \((N+1+N_h)\) by \((N+1+N_h)\) which is given as,

\[ r(k, n) = \frac{1}{N_v} \sum_{p=1}^{N_v} X_p(k) \cdot X_p(n) \] \hspace{1cm} (2.11)

Also \( C \) is the Cross correlation matrix of size \((N+1+N_h)\) by M

\[ c(k, i) = \frac{1}{N_v} \sum_{p=1}^{N_v} X_p(k) \cdot t_p(i) \] \hspace{1cm} (2.12)

Equation 2.10 is often ill-conditioned, meaning that the determinant of \( R \) is close to 0, it is often unsafe to use Gauss-Jordan elimination, so orthogonal least squares (OLS) algorithm is used to solve the equation.
2.3.4 Schmitd Procedure and Orthogonal Least Squares (OLS) for Output Weights

The Schmidt procedure is used for the normalization of the basis functions. The Schmidt procedure can be described as follows [22].

Given the basis functions $X(m)$, form the first orthonormal basis function $X'(1)$ as

$$X'(1) = \frac{X(1)}{||X(1)||} \tag{2.13}$$

which yields $||X'(1)|| = 1$.

The second orthonormal basis function is found as

$$c_1 = <X'(1), X(2) > \tag{2.14}$$

$$X'(2) = X(2) - \frac{c_1 X'(1)}{||X(2) - c_1 X'(1)||} \tag{2.15}$$

Similarly the third orthonormal basis function is found as

$$c_1 = <X'(1), X(3) > \quad c_2 = <X'(2), X(3) > \tag{2.16}$$

$$X'(3) = \frac{X(3) - c_1 X'(1) - c_2 X'(2)}{||X(3) - c_1 X'(1) - c_2 X'(2)||} \tag{2.17}$$

The Schmidt procedure requires at least one pass through the training data file for each new basis function. Since $X'(m)$ is a weighted sum of the $X(j)$, we can calculate all inner products $<X(m), X'(j)>$ as weighted sums of $<X(m), X(k)> = r(m, k)$, which means only one data pass is required to calculate the $R$ matrix given in equation (2.10).

Reformulating the Schmidt procedure we can get the form as,
\[ X'(m) = \sum_{k=1}^{m} a_{mk}X(k) \]  

(2.18)

The above in the matrix form can be given as

\[ X' = A \cdot X \]

Orthogonal least squares training approach makes use of the Schmidt procedure to calculate the output weights and Error for the system. First we calculate the output weights of the orthonormal system and then convert the weights to the original system.

The orthonormal output weights for the system is given as,

\[ w'_o(i, m) = E[X'(m)t(i)] \]  

(2.19)

Substituting \( X'(m) \) from equation (2.18)

\[ w'_o(i, m) = \sum_{k=1}^{m} a_{mk}E[X(k)t(i)] = \sum_{k=1}^{m} a_{mk}c(k, i) \]  

(2.20)

In matrix form we can write the above as

\[ W'_o = C^T \cdot A^T \]  

(2.21)

Now we have the weights for the orthonormal system, we now need to convert the weights from the orthonormal system to our original system, we can achieve that as follows

\[ y(i) = \sum_{k=1}^{N_u} w_o(i, k)X(k) = \sum_{m=1}^{N_u} w'_o(i, m)X'(m) \]  

(2.22)

We can replace \( X'(m) \) from equation (2.18) as follows
\[
\sum_{k=1}^{N_u} \left[ \sum_{m=1}^{N_u} w_o'(i, m) x'(m) \right] x(k) \tag{2.23}
\]

On changing the limits for the inner summation

\[
w_o(i, k) = \sum_{k=m}^{N_u} w_o'(i, m) a_{mk} \tag{2.24}
\]

Therefore we have the output weights for the original system as follows

\[
W_o = W_o' A \tag{2.25}
\]

### 2.3.5 Backpropagation for Solving Input Weights

Backpropagation [23, 24] is a common method of training neural networks, usually used to find the input gradient matrix and to compute the input weights. This report would focus on implementing backpropagation with HWO to compute the overall input gradient. The backpropagation gradient matrix and the delta functions can be calculated as follows,

Consider the mean squared error as mentioned in equation (2.6)

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

The delta functions for the \(p\)th pattern for output and hidden layer is given as

\[
\delta_{po}(i) = 2(t_p(i) - y_p(i)) \tag{2.26}
\]

\[
\delta_{p}(k) = f'(n_p(k)) \sum_{i=1}^{M} \delta_{po}(i) w_{oh}(i, k) \tag{2.27}
\]

Now on partially differentiating the Error function with respect to the input weights we get the gradient as,

\[
g(k, n) = \frac{-\partial E}{\partial W(k, n)} = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_{p}(k) x_p(n) \tag{2.28}
\]

In the matrix form the input gradient equation can be given as,
\[ G = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p (x_p)^T \] (2.29)

2.3.6 Hidden Weight Optimization (HWO)

After calculating the input gradient matrix \( G \) from the backpropagation algorithm, we can use this gradient in training the input weights using HWO as [25],

\[ W = W + z \cdot D \] (2.30)

The \( D \) matrix is the hidden weight changes. The hidden weights are updated by minimizing a separate error function for each hidden unit. This error function is generated by the difference between the desired net function and the actual net function. The input training is similar to OWO-BP but in this case we use the hidden weight changes matrix \( D \). Now for the \( p \)th pattern the desired net function is calculated as [26]

\[ n_{pj} \cong n_{pj} + z \cdot \delta_{pj} \] (2.31)

In this equation \( \delta_{pj} \) is the delta function for the \( j \)th hidden unit as in (2.27)

The hidden weight changes are derived using,

\[ n_{pj} + z \cdot \delta_{pj} \cong \sum_{n=1}^{N+1} [w_h(j,n) + z \cdot e(j,n)] \cdot x(p,n) \] (2.32)

From the above equation we have,

\[ \delta_{pj} = \sum_{n=1}^{N+1} e(j,n) \cdot x(p,n) \] (2.33)

The error function of each hidden unit is taken separately into consideration and the error for the \( j \)th Hidden unit is given as
\[ E_\delta(j) = \frac{1}{N_v} \sum_{p=1}^{N_v} \left[ \delta_p j - \sum_{n=1}^{N+1} e(j, n)x_{pn} \right]^2 \]  

(2.34)

Now on taking the partial derivative of the above error function with respect to \( e(j, n) \) we get

\[ D \cdot R_i = G \]  

(2.35)

\( R_i \) is the same as input autocorrelation matrix from equation (2.10)

The equation (2.35) can be written in form of the hidden weight change matrix as,

\[ D = G \cdot R_i^{-1} \]  

(2.36)

The above hidden weight change matrix can be used to train the input weights as follows,

\[ W = W + z \cdot D \]  

(2.37)

### 2.3.7 Multiple Optimal Learning Factor (MOLF)

Multiple optimal training factor (MOLF) [26] is a higher order training algorithm. In this technique of training we find a vector \( z \) of optimal learning factors which has one element for each hidden unit.

Assuming that a separate OLF \( z_i \) is being used to update each hidden units input weights, \( w(k, n) \), where \( 1 \leq n \leq (N + 1) \). The total error function \( y_p(m) \) to be minimized is given as,

\[ y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m, n)x_p(n) + \sum_{k=1}^{N_h} w_{ok}(m, k)f(\sum_{i=1}^{N+1} (w(k, i) + z_k \cdot g(k, i))x_p(i)) \]  

(2.38)

where, \( g(k, n) \) is an element of the negative Jacobian matrix \( G \). Now the partial of \( E \) with respect to \( z_j \)
\[ \frac{\partial E}{\partial z_j} = -2 \sum_{p=1}^{N_v} \sum_{m=1}^{M} \bar{e}_p(m) - \sum_{k=1}^{N_h} w_{oh}(m, k) O_p(z_k) \cdot w_{oh}(m, j) O_p(j) \Delta n_p(j) \]  

(2.39)

where,

\[ \bar{e}_p(m) = t_p(m) - \sum_{n=1}^{N+1} w(m, n)x_p(n) \]

\[ \Delta n_p(j) = \sum_{n=1}^{N+1} x_p(n) \cdot g(n, j) \]

The Gauss–Newton update guarantees that \( H_{molf} \) is non-negative definite.

Now given the negative gradient vector \( g_{molf} = \left[ \frac{-\partial E}{\partial z_1}, \frac{-\partial E}{\partial z_2}, \ldots, \frac{-\partial E}{\partial z_{N R}} \right]^T \) and the Hessian \( H_{molf} \), we can minimize \( E \) with respect to the vector \( z \) using Newton’s method.

Now using the \( z_k \) vector of optimal learning factors we can update the input weights as follows,

\[ w(k, n) = w(k, n) + z_k \cdot d(k, n) \]  

(2.42)
CHAPTER 3

Problems and Proposed Work

3.1 Problems with Initialization of MLP

To design and train a highly efficient and robust Neural Network involves a lot of fundamental challenges. Initialization of parameters of neural network model is one of them. In this thesis, parameter initialization problem is addressed and, two better parameter initialization methods namely fusion method, and modified sigmoid method are shown.

There are three main problems with random parameter initialization. First is difficulty in finding global minima on optimization curve, second is saturation of hidden units of neural network and third is possibility of dependent hidden units in neural network.

MLP are used to solve optimization problem. But, central problem with this optimization is that optimization curve is of non-convex nature. Plus, optimization curve is continuous and high dimensional. Gradient descent methods are used to solve this optimization problem. So, starting point on this optimization curve is decided by randomly initialized parameter. Gradient descent methods while trying to optimize may get stuck in local minima, hence parameter update by using gradient descent becomes difficult. Thus, gradient descent methods are not sufficient enough to find global minima. This shows that starting point on an optimization curve which is decided by random weight initialization is of prime importance.

Another problem with random parameter initialization is saturation of hidden units. This happens because product of weights and inputs, puts outputs of net function on to the saturation region of sigmoid. Though net control is provided to keep it below saturation, it is not always possible to keep output of net function below saturation. Problem with saturation of sigmoid is that during back propagation gradient will not get passed through hidden units to input weights. This affects learning of input weights of model. This makes those input weights useless in optimization curve. Not saturating hidden units are of prime importance in training of neural network.

Third problem with random initialization is of dependent hidden units. While training, there is chance that due to random initialization some of the hidden units become dependent. Dependent hidden units means
they extract same features from data. So, while optimizing, dependent hidden units will not give new information for optimization.

To overcome this problem with initialization, in this thesis, I have proposed two different algorithm namely fusion method and modified sigmoid method.

3.2 Proposed Methods

3.2.1 Fusion Method

In the fusion method, instead of creating single network model with random initialization for training, create different network models which have the same model structure as a single network model, with random initialization. Train those models for some amount of iterations and then take hidden units from those different networks and create a single network. While taking hidden units from different networks, their weights are copied from the models to create one network. The idea behind this is, due to merging of different network, problem of network converging to local minima may be eliminated. Also, saturated hidden units due to random initialization can be removed while making new single network. Detailed explanation of this algorithm is given in chapter 4.

3.2.2 Modified Sigmoid Method

In this method, after randomly initializing weights, we perform addition of additive and scaling parameters after net function. These additive and scaling parameters are trained using Newton’s method. After training, we update the input weights of the model using these additive and scaling parameters. This helps to keep hidden units in active region of activation function. Details explanation of this algorithm is given in chapter 5.
CHAPTER 4

Fusion Method

4.1 Algorithm

In parameter initialization of MLP, common problems are hidden units saturation, dependent hidden units and network stuck to local minima which were discussed in chapter 3. Instead of creating single network with random initialization and training it for some iterations, multiple networks with the same network model architecture are created, randomly initialized, trained for some iteration and, then merged into a single network by taking average of parameters of models or selecting some hidden units from different networks. This is called fusion process. Final fused network is also trained for some iterations.

The advantage of fusion method is that, initially created individual networks are randomly initialized, so each network is started at different point on optimization curve. Thus, even if some network gets stuck into a local minima, there is a chance that it will be removed from the local minima when the fusion of networks takes place. This mitigates the problem of local minima.

The second problem of saturated hidden units is resolved, because the fused network formed by randomly selecting hidden units from multiple different networks, alleviates the chances of saturation of hidden units. This method also removes dependent hidden units in network, by randomly selecting hidden units to form the final fusion network.

In this algorithm, the method used to fuse the networks together is an important factor. In this thesis, two methods of parameter fusion are discussed,

1. Average of the input weights of three networks
2. Select 1/3 of hidden units from individual network

To find best method for fusion, create 1000 MLP networks with 51 hidden units and perform 500 iterations to train the networks. First, we train 1000 MLP networks without using fusion method on dataset Oh17.
1000 MLP networks are trained using fusion method on dataset Oh17. In this method, initially, three different networks with 51 hidden units are created. These networks are trained with random initialization for 50 iterations. Then, these three networks are fused into one network by taking average of the parameters of the three networks, and then again trained for 450 iterations. Histogram of this is shown in figure 4.2.
1000 MLP networks are trained using fusion method on dataset Oh17. In this method, initially, three different networks with 51 hidden units are created. Trained with random initialization for 50 iterations. Then, these three networks are fused into one network by taking one third hidden units from each network and then again trained for 450 iterations. Histogram of this is shown in figure 4.2
To determine best method for fusion, further analysis of standard deviation and mean is done.
Figure 4.4: Histogram of Error with Different NN Initialization Techniques
Table 4.1 shows that, fusion method with combination of one third hidden neurons from each network gives best result. So, for further study, this method is used in this thesis.

After finalizing method for fusion, we need to decide the number of iterations required to train these initial networks before fusing them into one network. For this analysis, number of initial iterations to train the three different neural networks selected was 0, 1, 5, 10, 25, 50, 75, 100, 200, 300, 400, and 499. But total number of iterations i.e. number of iterations before fusion and after fusion is kept constant at 500. Data file used for training is oh17 and number of hidden units is 51, in each network. That means 17 hidden units from each network are taken and merged into single fused network.
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>500</td>
</tr>
</tbody>
</table>

Figure 4.5: Histogram of Error with 0 iteration for initial training
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>499</td>
</tr>
</tbody>
</table>

Figure 4.6: Histogram of Error with 1 iteration for initial training
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>495</td>
</tr>
</tbody>
</table>

Histogram of Error with 5 Iteration for Initial Training

Figure 4.7: Histogram of Error with 5 iterations for initial training
Data file = Oh7.tra  Inputs = 20  Outputs = 3  Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>490</td>
</tr>
</tbody>
</table>

Figure 4.8: Histogram of Error with 10 Iterations for Initial Training
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>25</td>
<td>1</td>
<td>475</td>
</tr>
</tbody>
</table>

Figure 4.9: Histogram of Error with 25 Iterations for Initial Training
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>50</td>
<td>1</td>
<td>450</td>
</tr>
</tbody>
</table>

Figure 4.10: Histogram of Error with 50 Iterations for Initial Training
Data file = Oh7.tra     Inputs = 20     Outputs = 3     Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>75</td>
<td>1</td>
<td>425</td>
</tr>
</tbody>
</table>

![Histogram of Error with 75 Iteration for Initial Training](image)

Figure 4.11: Histogram of Error with 75 Iterations for Initial Training
Data file = Oh7.tra  Inputs = 20  Outputs = 3  Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>100</td>
<td>1</td>
<td>400</td>
</tr>
</tbody>
</table>

Figure 4.12: Histogram of Error with 100 Iterations for Initial Training
Data file = Oh7.tra  Inputs = 20  Outputs = 3  Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>200</td>
<td>1</td>
<td>300</td>
</tr>
</tbody>
</table>

Figure 4.13: Histogram of Error with 200 Iterations for Initial Training
Data file = Oh7.tra  Inputs = 20  Outputs = 3  Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>300</td>
<td>1</td>
<td>200</td>
</tr>
</tbody>
</table>

Figure 4.14: Histogram of Error with 300 Iterations for Initial Training
Data file = Oh7.tra  Inputs = 20  Outputs = 3  Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>400</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 4.15: Histogram of Error with 400 Iterations for Initial Training
Data file = Oh7.tra    Inputs = 20    Outputs = 3    Nh = 51

<table>
<thead>
<tr>
<th>Total Nit</th>
<th>No Of Networks Before Fusion</th>
<th>Nit before Fusion</th>
<th>No of Network After Fusion</th>
<th>Nit after Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>499</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.16: Histogram of Error with 499 Iterations for Initial Training
Now, analysis for mean and standard deviation for each of above histogram is done. This is shown in table 4.2

**Table 4.2 - Error Analysis for Number of Iterations before Fusion into Single Network using HWO-MOLF**

<table>
<thead>
<tr>
<th>Number of Iterations before Fusing into Single Network</th>
<th>Standard Deviation of Error</th>
<th>Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0276</td>
<td>1.1637</td>
</tr>
<tr>
<td>1</td>
<td>0.0253</td>
<td>1.1457</td>
</tr>
<tr>
<td>5</td>
<td>0.0251</td>
<td>1.1326</td>
</tr>
<tr>
<td>10</td>
<td>0.0254</td>
<td>1.1272</td>
</tr>
<tr>
<td>25</td>
<td>0.0236</td>
<td>1.1244</td>
</tr>
<tr>
<td>50</td>
<td>0.0241</td>
<td>1.1219</td>
</tr>
<tr>
<td>75</td>
<td>0.0237</td>
<td>1.1223</td>
</tr>
<tr>
<td>100</td>
<td>0.0232</td>
<td>1.1230</td>
</tr>
<tr>
<td>200</td>
<td>0.0234</td>
<td>1.1292</td>
</tr>
<tr>
<td>300</td>
<td>0.0216</td>
<td>1.1379</td>
</tr>
<tr>
<td>400</td>
<td>0.0200</td>
<td>1.1526</td>
</tr>
<tr>
<td>499</td>
<td>0.0463</td>
<td>1.4704</td>
</tr>
</tbody>
</table>
Table 4.2 shows that for initial iterations of value 100 to train three different networks before merging into one gives best result considering mean error and standard deviation. So, all three networks are trained for 100 iterations before merging into one network.
4.2 Final Algorithm

1. Decide number of hidden units \( N_h \) (\( N_h = 51 \)) for MLP networks.
2. Create 3 different MLP with given \( N_h \) and initialize parameters randomly for each network.
3. Train 3 networks separately for 100 iterations by using HWO-MOLF Algorithm.
4. Take one third hidden units from each network and create final network which has \( N_h \) same as initial \( N_h \).
5. Train final network for 400 iterations by using HWO-MOLF algorithm.

4.3 Results

Table 4.3 - Performance Comparison with Fusion Method and without Fusion Method

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of inputs</th>
<th>Number of outputs</th>
<th>Training Error Without Fusion Method</th>
<th>Testing Error Without Fusion Method</th>
<th>Training Error With Fusion Method</th>
<th>Testing Error With Fusion Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twod</td>
<td>8</td>
<td>7</td>
<td>0.102378</td>
<td>0.1369</td>
<td>0.098858</td>
<td>0.1360</td>
</tr>
<tr>
<td>Oh7</td>
<td>20</td>
<td>3</td>
<td>0.101961</td>
<td>0.1175</td>
<td>0.091258</td>
<td>0.1136</td>
</tr>
<tr>
<td>Mat</td>
<td>4</td>
<td>4</td>
<td>0.000873</td>
<td>0.0011</td>
<td>0.000419</td>
<td>0.000501</td>
</tr>
<tr>
<td>Weather Forecasting</td>
<td>71</td>
<td>3</td>
<td>255.7176</td>
<td>290.0440</td>
<td>255.42914</td>
<td>289.037872</td>
</tr>
</tbody>
</table>

Table 4.3 – Compares performance of MLP with fusion method and without fusion method. \( N_h = 51 \). \( N_{it} = 500 \). For fusion method, three MLP are created and fused after 100 iterations.
CHAPTER 5

Modified Sigmoid Method

5.1 Algorithm

1. Create MLP network with required hidden units $N_h$.
2. Perform net control.
3. Perform OWO and save net function vector to an unformatted file with $i_e(p)$.
4. Create scaling factor $a_k$ and additive factor $b_k$ for each hidden neuron, where $k$ stands for hidden neuron number. Initialize $a_k$ to 1 and $b_k$ to 0.
5. Choose number of iterations $N_{it}$.
6. Calculate $g_a(k)$ and $g_b(k)$, where $g_a(k)$ is gradient of $a$ for $k^{th}$ hidden neuron and $g_b(k)$ is gradient of $b$ for $k^{th}$ hidden neuron, as

\[
g_a(k) = - \frac{\partial E}{\partial a(k)} = \frac{2}{N_v} \sum_p \sum_{i=1}^M \left[ t_p(i) - y_p(i) \right] \frac{\partial y_p(i)}{\partial a(k)}
\]

\[
\frac{\partial y_p(i)}{\partial a(k)} = w_{oh}(i,k)s'\left(n_p(k)\right)n_p(k)
\]

Where, $s'(net) = s(net) \times (1 - s(net))$

\[
s(net) = \frac{1}{1 + e^{-net}}
\]

\[
g_b(k) = - \frac{\partial E}{\partial b(k)} = \frac{2}{N_v} \sum_p \sum_{i=1}^M \left[ t_p(i) - y_p(i) \right] \frac{\partial y_p(i)}{\partial b(k)}
\]

\[
\frac{\partial y_p(i)}{\partial b(k)} = w_{oh}(i,k)s'\left(n_p(k)\right)
\]

7. For iteration $i_t = 1$ to $N_{it}$, where $N_{it}$ is the total number of iterations
8. Initialize $g_a(k)$ and $g_b(k)$ as zero for all values hidden units.
9. Initialize $E$ equal to zero.
10. Calculate $O(k)$ and $O'(k)$ as
\[ O(k) = s(n_p(k)a(k) + b(k)) \]
\[ O'(k) = O(k) \ast (1 - O(k)) \]

11. Calculate \( y \) as,

\[
y = x \ast Wo'\]

12. For classification, calculate \( t_p \) for \( i_c \) and \( y \) using OR.

13. Calculate \( E \) as,

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

14. Calculate \( g_a(k) \) and \( g_b(k) \) using equation 5.1 and 5.2.

15. Print \( i_t \) and \( E \).

16. Find hessian matrix \( H \) for \( a_k \) and \( b_k \).

17. Use OLS (Orthogonal Least Squares) to solve

\[ H \cdot z = g \]

where, \( z \) is learning factor for \( a_k \) and \( b_k \)

\[
z = [z_{a1}, z_{a2}, ..., z_{bNh}]^T
\]

\( g \) is negative gradient vector

\[
g = \begin{bmatrix} -\frac{\partial E}{\partial z_{a1}}, -\frac{\partial E}{\partial z_{a2}}, ..., -\frac{\partial E}{\partial z_{bNh}} \end{bmatrix}^T
\]

\[
H(m, n) = \frac{\partial y}{\partial z_m} \frac{\partial y}{\partial z_n}
\]

18. Perform OWO (Output Weight Optimization) to find \( Wo \).

19. Increase \( i_t \) by 1. If \( i_t < N_{it} \), go to 4

20. End iterations. Finished getting \( a_k \) and \( b_k \).

21. Update input weights by using \( a_k \) and \( b_k \).

\[
w(k, n) \leftarrow a_k w(k, n)
\]

\[
w(k, N + 1) \leftarrow w(k, N + 1) + b_k
\]
22. $w$ from step 21 is initialized $w$ and use that $w$ to further train MLP.

5.2 Result

Table 5.1 - Performance Comparison with Modified Sigmoid and without Modified Sigmoid

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of inputs</th>
<th>Number of outputs</th>
<th>Training Error Without Modified Sigmoid Method</th>
<th>Testing Error Without Modified Sigmoid Method</th>
<th>Training Error With Modified Sigmoid Method</th>
<th>Testing Error With Modified Sigmoid Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twod</td>
<td>8</td>
<td>7</td>
<td>0.102378</td>
<td>0.1369</td>
<td>0.097071</td>
<td>0.1355</td>
</tr>
<tr>
<td>Oh7</td>
<td>20</td>
<td>3</td>
<td>0.101961</td>
<td>0.1175</td>
<td>0.090968</td>
<td>0.1135</td>
</tr>
<tr>
<td>Mat</td>
<td>4</td>
<td>4</td>
<td>0.000873</td>
<td>0.0011</td>
<td>0.0004</td>
<td>0.000477</td>
</tr>
<tr>
<td>Weather Forecasting</td>
<td>71</td>
<td>3</td>
<td>255.7176</td>
<td>290.0440</td>
<td>253.827720</td>
<td>290.2045</td>
</tr>
</tbody>
</table>
In this thesis, improved parameter initialization techniques like fusion method and modified sigmoid are proposed for training MLP network HWO - MOLF by using a supervised learning approach. The results show a considerable improvement in performance for both algorithms compared to random parameter initialization technique for HWO – MOLF network. A possible reason for this could be that both the algorithms are able to remove problems like local minima convergence, saturated hidden units and dependent hidden units, faced during parameter initialization to some extent. Also, Modified sigmoid method gives slightly better results when compared to fusion method.

Depending upon different configuration of HWO – MOLF network and also on the random initialization of the network, final error in training the HWO – MOLF network differs. In the result of both the algorithms, it is observed that for the same number of training iterations, training error is significantly reduced in comparison to random parameter initialization. Time to train HWO – MOLF network by using fusion method is more than Modified sigmoid method because of creating multiple networks of same architecture and training them individually.
Appendix A

Description of datasets
TWOD

This file has 8 inputs, 7 outputs and 1768 training patterns. 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 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. [56, 57]

OH7

This file has 20 inputs, 3 outputs and 10,453 training patterns. The training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g / cubic cm. [58]

MAT

This file has 4 inputs and 4 outputs and 2000 training patterns. This training file provides the data set for inversion of random two-by-two matrices. Each pattern consists of 4 input features and 4 output features. The input features, which are uniformly distributed between 0 and 1, represent a matrix and the four output features are elements of the corresponding inverse matrix. The determinants of the input matrices are constrained to be between .3 and 2. [61]

WEATHER FORECASTING DATA

This file has 71 inputs and 3 outputs and 72,050 training patterns. The weather data made from the years 2010 to 2013. First 4 inputs are time inputs (encoded in continuous form); Inputs 5 to 8 are spatial variables (latitude, longitude) that indicate the monitoring site/station and city the pattern comes from; Inputs 9 to 71 comprise time delayed data up to 3 days of Daily Mean, Daily Min, and Daily Max values of meteorological
variables (temperature, solar radiation, wind speed and wind direction encoded together in continuous form) and pollutant variables (nitric oxide, nitrogen dioxide, 8-hour average ozone concentration). Outputs are Daily Maximum 8-hour average ozone concentration up to 3 days ahead. [62]
REFERENCES


[35] Campos, Guilherme O.; Zimek, Arthur; Sander, Jörg; Campello, Ricardo J. G. B.; Micenková, Barbora; Schubert, Erich; Assent, Ira; Houle, Michael E. (2016). "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". Data Mining and Knowledge Discovery.


[54] Y. W. Qiong Liu, "Supervised Learning".


