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

LIST OF FIGURES

LIST OF TABLES

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 θ_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 $a_j(t)$, θ_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 $p_i(t)$ to the neuron j from the outputs $o_i(t)$ of predecessor neurons and typically has the form

$$\boldsymbol{p_j(x)} = \sum_i o_i(t) w_{ij}$$

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.

$$\emptyset(\boldsymbol{x}) = \emptyset(||\boldsymbol{x} - \boldsymbol{m_k}||)$$

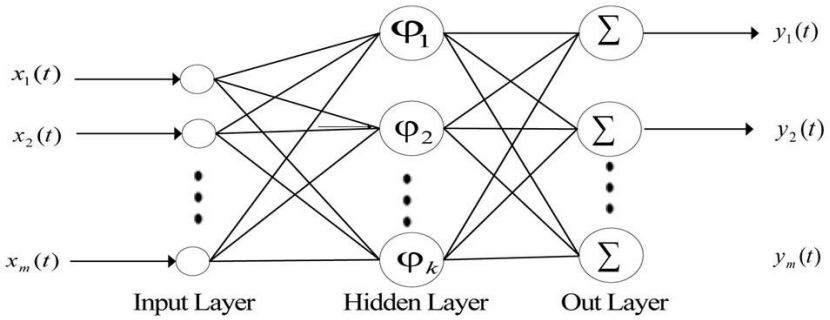

Figure 1.1: Radial Basis Function Network

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.

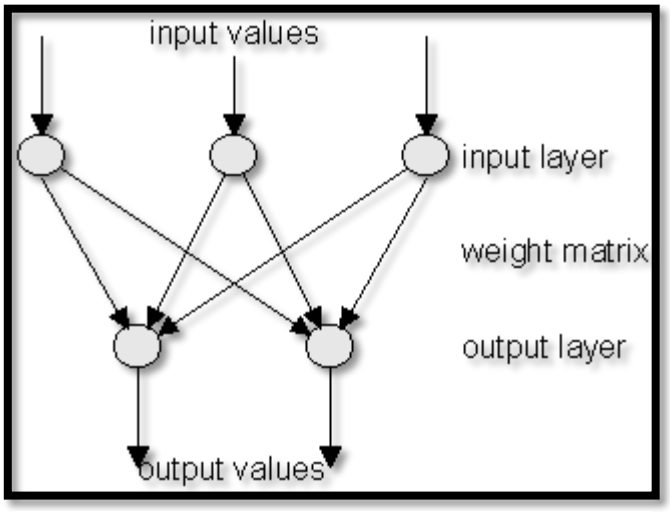

Figure 1.2: Single Layer Perceptron Network

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 MPL 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]

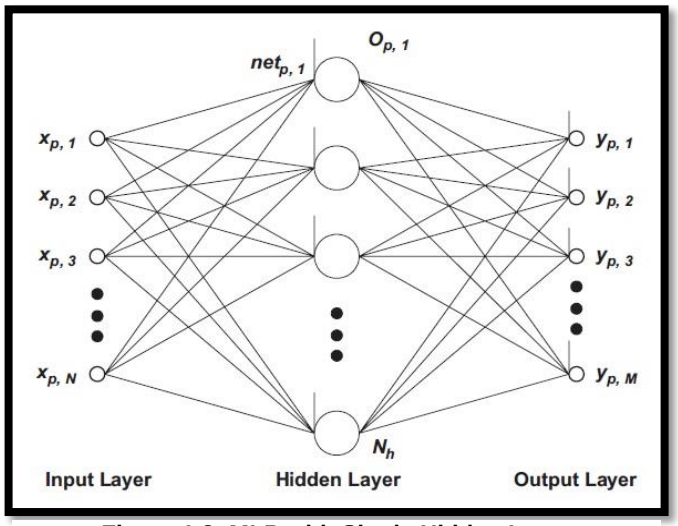

Figure 1.3: MLP with Single Hidden Layer

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.

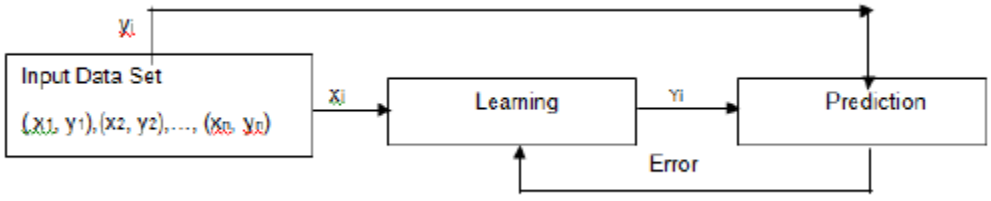

Figure 1.4: Supervised Training Method

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]

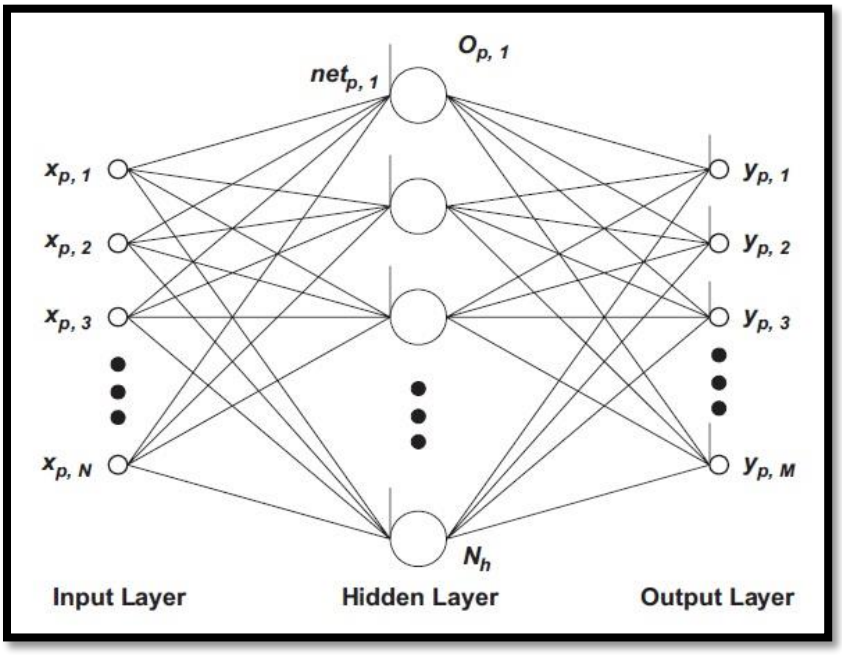

Figure 2.1: MLP with single hidden layer

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 θ to form the net function, followed by an application of a nonlinear activation function f(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]

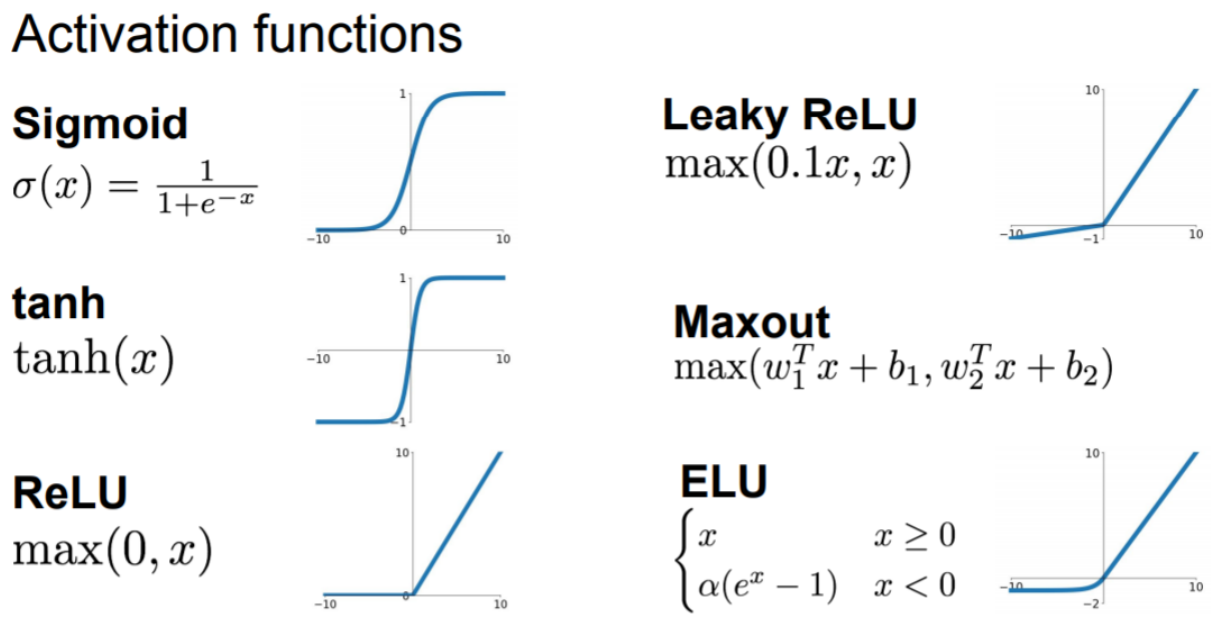

Figure 2.2: Nonlinear Activation Functions

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

The sigmoid activation function is given as follows

$$O_p = f(n_p) = \frac{1}{1 + e^{-n_p}} \tag{2.1}$$

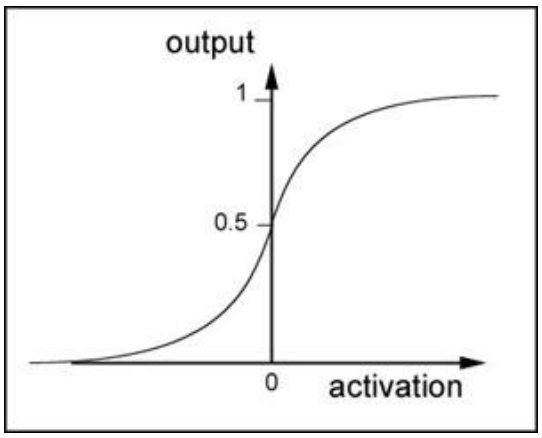

Figure 2.3: Sigmoid Activation Function

2.2 Multilayer Perceptron Notation

In the training data $\{\boldsymbol{x_p}, \boldsymbol{t_p}\}$ for a fully connected MLP, $\boldsymbol{x_p}$ represents p[th] input vector having dimension N and, $\boldsymbol{t_p}$ 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 $\boldsymbol{x_p} = \left[x_p(1), x_p(2), \ldots\ldots, x_p(N+1)\right]^T$. And p is a pattern number which varies from 1 to N_v.

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

$$n_p(k) = \sum_{n=1}^{N+1} w(k,n) x_p(n) \tag{2.2}$$

Which when written in matrix notation could be given as

$$\boldsymbol{n_p} = \boldsymbol{W} \cdot \boldsymbol{x_p} \tag{2.3}$$

Here $\boldsymbol{n_p}$ denotes the N_h dimensional column vector of a net function values and **W** is N_h 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.

For the p[th] pattern, the i[th] element $y_p(i)$ of the M dimensional output vector $\boldsymbol{y_p}$ is given as

$$y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)\, x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k) O_p(k) \tag{2.4}$$

This can also be denoted in matrix form as

$$\boldsymbol{y_p} = \boldsymbol{W_{oi}} \cdot \boldsymbol{x_p} + \boldsymbol{W_{oh}} \cdot \boldsymbol{O_p} \tag{2.5}$$

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

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[t_p(i) - y_p(i) \right]^2 \tag{2.6}$$

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

$$\theta(2,k) = \theta(2,k) - m(2,k).\frac{\sigma_d}{\sigma(2,k)} + m_d \qquad (2.7)$$

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].

$$\boldsymbol{y_p} = \boldsymbol{W_{oi}} \cdot \boldsymbol{x_p} + \boldsymbol{W_{oh}} \cdot \boldsymbol{O_p} \qquad (2.8)$$

$$\boldsymbol{y_p} = \boldsymbol{W_o} \cdot \boldsymbol{X_p} \tag{2.9}$$

where, $\boldsymbol{W_o} = [\boldsymbol{W_{oi}} : \boldsymbol{W_{oh}}]$

$$\boldsymbol{X_p} = \left[\boldsymbol{x_p^T}, \boldsymbol{O_p^T}\right]^T$$

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

X(n) = x(n) for n between 1 and N

X(N+1) = 1

$X(N+1+k) = O_k$ for k between 1 and N_h

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

$$\boldsymbol{R}.\boldsymbol{W_o^T} = \boldsymbol{C} \tag{2.10}$$

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

$$r(k,n) = \frac{1}{N_v}\sum_{p=1}^{N_v} X_p(k).X_p(n) \tag{2.11}$$

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

$$c(k,i) = \frac{1}{N_v}\sum_{p=1}^{N_v} X_p(k).t_p(i) \tag{2.12}$$

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 Schimdt 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,

$$X'(m) = \sum_{k=1}^{m} a_{mk} X(k) \tag{2.18}$$

The above in the matrix form can be given as

$$\boldsymbol{X' = A \cdot X}$$

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,

$$w_o'(i,m) = E[X'(m)t(i)] \tag{2.19}$$

Substituting X'(m) from equation (2.18)

$$w_o'(i,m) = \sum_{k=1}^{m} a_{mk} E[X(k)t(i)] = \sum_{k=1}^{m} a_{mk} c(k,i) \tag{2.20}$$

In matrix form we can write the above as

$$\boldsymbol{W_o' = C^T \cdot A^T} \tag{2.21}$$

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

$$y(i) = \sum_{k=1}^{N_u} w_o(i,k) X(k) = \sum_{m=1}^{N_u} w_o'(i,m) X'(m) \tag{2.22}$$

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

$$\boldsymbol{W_o = W_o'A} \tag{2.25}$$

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,

$$\boldsymbol{G} = \frac{1}{N_v}\sum_{p=1}^{N_v} \boldsymbol{\delta_p}\left(\boldsymbol{x_p}\right)^T \tag{2.29}$$

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],

$$\boldsymbol{W} = \boldsymbol{W} + z \cdot \boldsymbol{D} \tag{2.30}$$

The $\boldsymbol{D}$ 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 $\boldsymbol{D}$. Now for the pth pattern the desired net function is calculated as [26]

$$n_{pj} \cong n_{pj} + z \cdot \delta_{pj} \tag{2.31}$$

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

The hidden weight changes are derived using,

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

From the above equation we have,

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

The error function of each hidden unit is taken separately into consideration and the error for the j[th] Hidden unit is given as

$$E_\delta(j) = \frac{1}{N_v}\sum_{p=1}^{N_v}\left[\delta_{pj} - \sum_{n=1}^{N+1} e(j,n)x_{pn}\right]^2 \tag{2.34}$$

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

$$\boldsymbol{D \cdot R_i = G} \tag{2.35}$$

$\mathbf{R_i}$ 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,

$$\boldsymbol{D = G \cdot R_i^{-1}} \tag{2.36}$$

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

$$\boldsymbol{W = W + z \cdot D} \tag{2.37}$$

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_k 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,

$$y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k) f\Big(\sum_{i=1}^{N+1}\big(w(k,i) + z_k \cdot g(k,i)\big)x_p(i)\Big) \tag{2.38}$$

where, $g(k,n)$ is an element of the negative Jacobian matrix G. Now the partial of E with respect to z_j

$$\frac{\partial E}{\partial z_j} = -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{m=1}^{M}[\,\bar{t_p}\,(m) - \sum_{k=1}^{N_h} w_{oh}(m,k)O_p(z_k)\,] \cdot w_{oh}(m,j)O_p(j)\Delta n_p(j) \tag{2.39}$$

where,

$$\bar{t_p}(m) = t_p(m) - \sum_{n=1}^{N+1} w(m,n)x_p(n)\ , \qquad \Delta n_p(j) = \sum_{n=1}^{N+1} x_p(n)\cdot g(n,j)$$

$$O_p(z_k) = f(\sum_{n=1}^{N+1}\big(w(k,n) + z_k \cdot g(k,n)\big)x_p(n)$$

Using the Gauss – Newton updates, the second partial derivative elements of the Hessian H_{molf} are

$$h_{molf}(l,j) \approx \frac{\partial^2 E}{\partial z_l \partial z_j} = \frac{2}{Nv}\sum_{m=1}^{M} w_{oh}(m,l)w_{oh}(m,j)\sum_{p=1}^{N_v} O'_p(l)O'_p(j)\Delta n_p(l)\Delta n_p(j) \tag{2.40}$$

$$= \sum_{i=1}^{N+1}\sum_{n=1}^{N+1}[\frac{2}{N_v}u(l,j)\sum_{p=1}^{N_v} x_p(i)x_p(n)O'_p(l)O'_p(j)]\ g(l,i)\cdot g(j,n) \tag{2.41}$$

The Gauss-Newton update guarantees that H_{molf} is non-negative definite. Now given the negative gradient vector $g_{molf} = \left[\frac{-\partial E}{\partial z_1}, \frac{-\partial E}{\partial z_2} \ldots\ldots, \frac{-\partial E}{\partial z_{Nh}}\right]^T$ and the Hessian H_{molf} , we can minimize E with respect to the vector z using Newton's method.

Now using the $\boldsymbol{z_k}$ vector of optimal learning factors we can update the input weights as follows,

$$w(k,n) = w(k,n) + z_k \cdot d(k,n) \tag{2.42}$$

where the vector z can be given as

$$\boldsymbol{z = H_{molf}^{-1} \cdot g_{molf}} \tag{2.43}$$

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.

Histogram of this is shown in figure 4.1

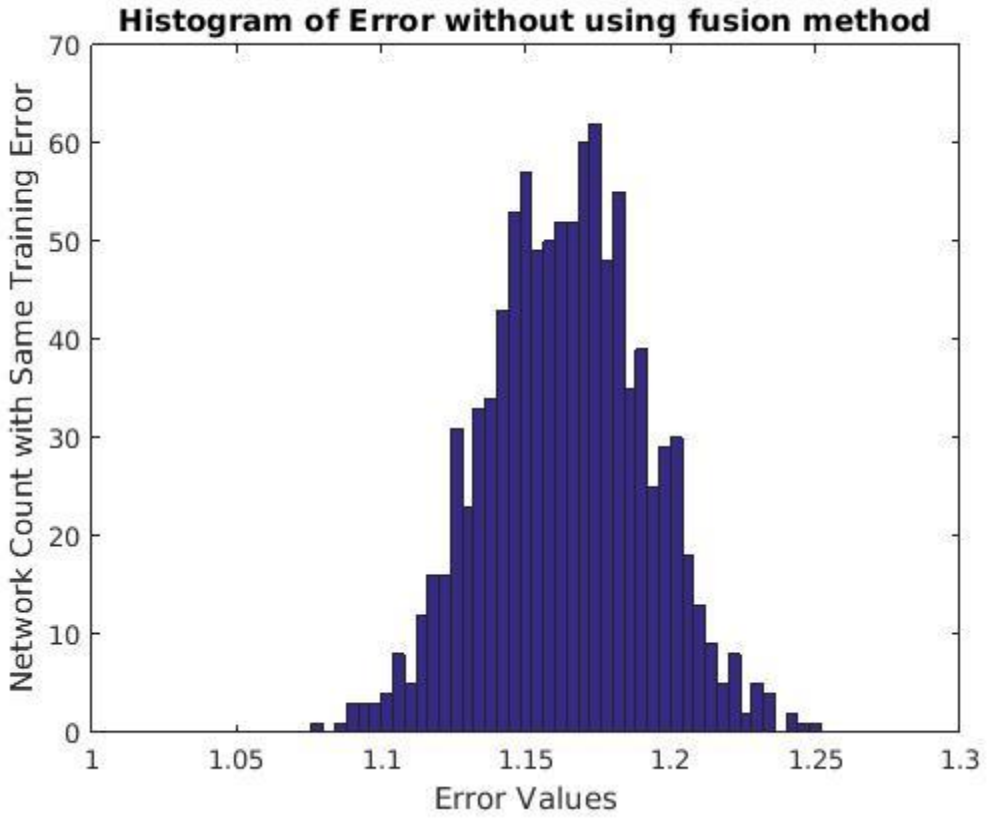

Figure 4.1: Histogram of Error without using Fusion Method

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

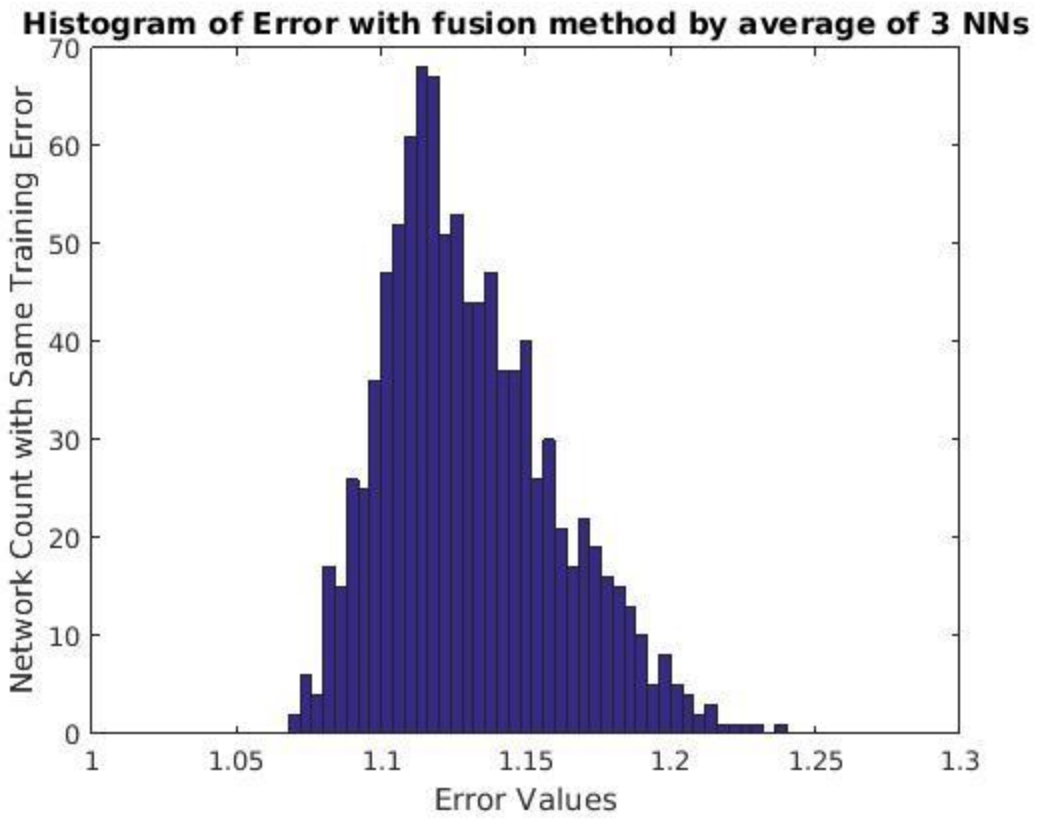

Figure 4.2: Histogram of Error with Fusion Method by Average of 3 NNs

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

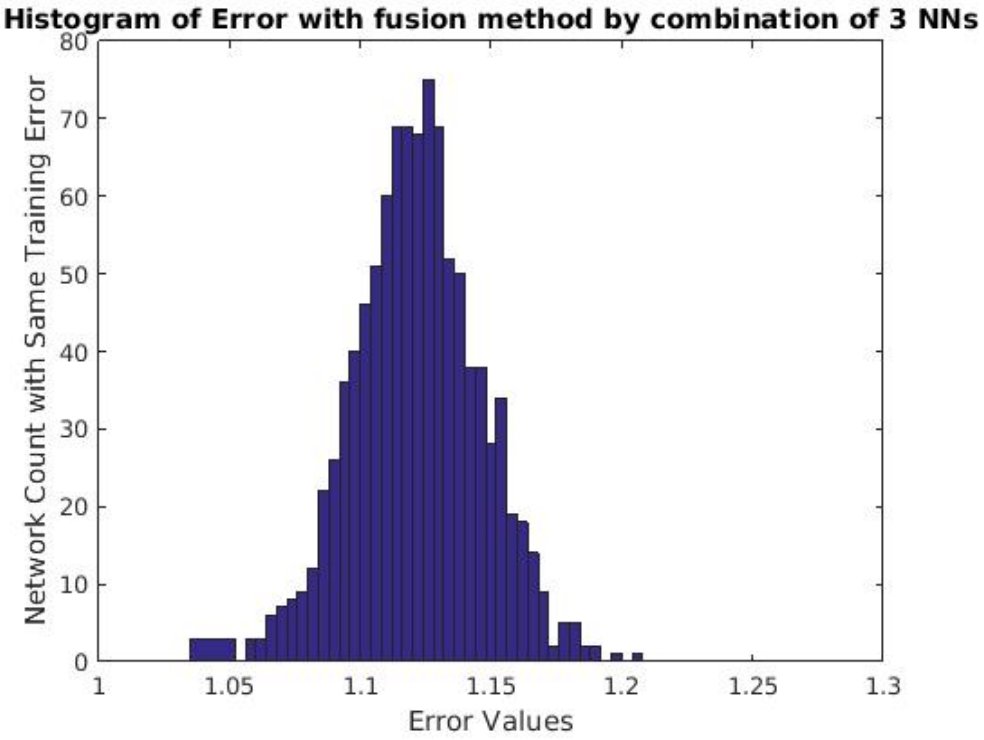

Figure 4.3: Histogram of Error With Fusion Method by Combination of 3 NNs

To determine best method for fusion, further analysis of standard deviation and mean is done.

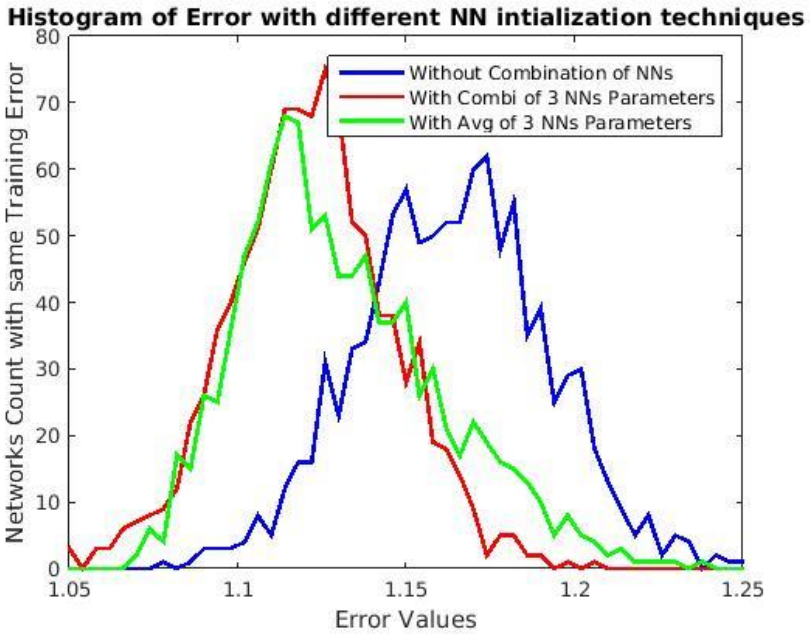

Figure 4.4: Histogram of Error with Different NN Initialization Techniques

Table 4.1 - Error Analysis for Fusion Method using HWO-MOLF

Table 4.1 – Error Analysis for Fusion Method using HWO-MOLF		
Method	Standard Deviation of Error	Error Mean
Without Fusion Method	0.0276	1.1637
With Average of 3 NNs	0.0295	1.1306
With Combination of 3 NNs	0.0241	1.1219

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				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	0	1	500

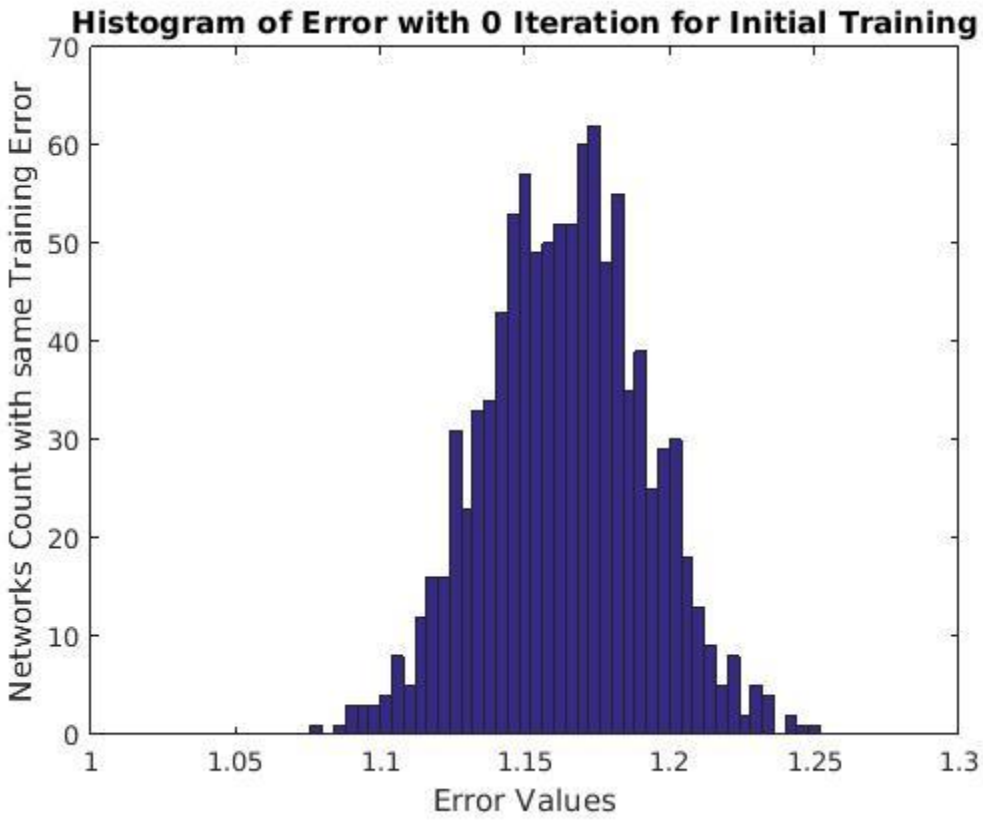

Figure 4.5: Histogram of Error with 0 iteration for initial training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	1	1	499

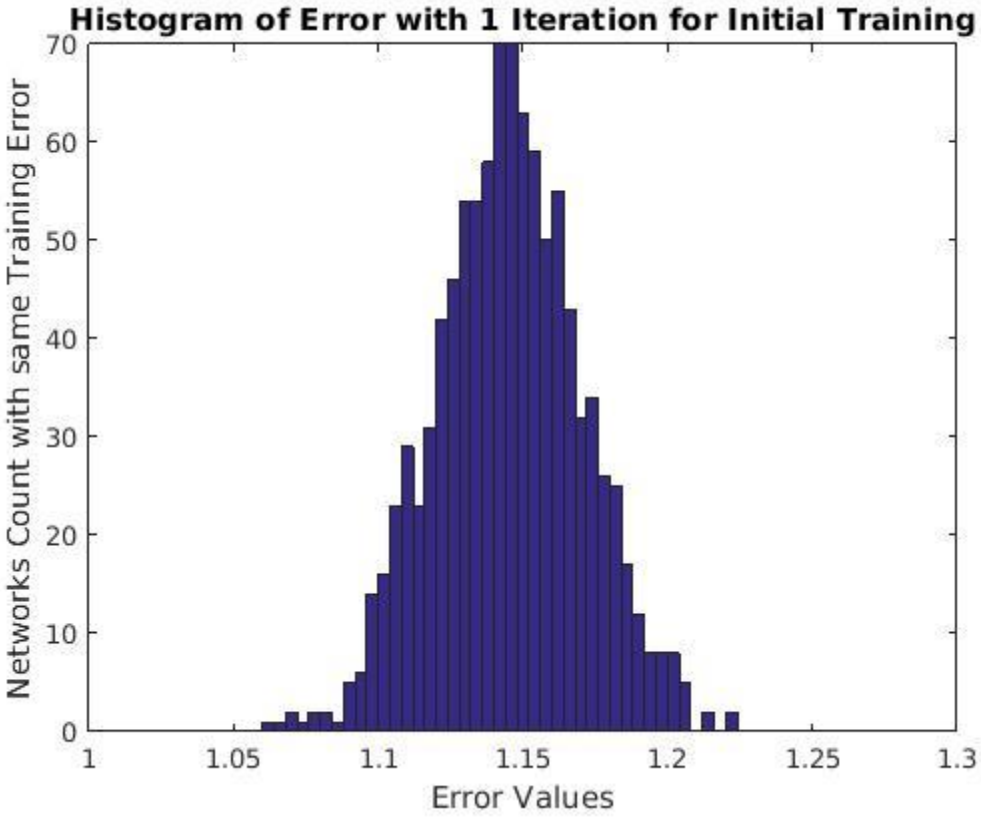

Figure 4.6: Histogram of Error with 1 iteration for initial training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	5	1	495

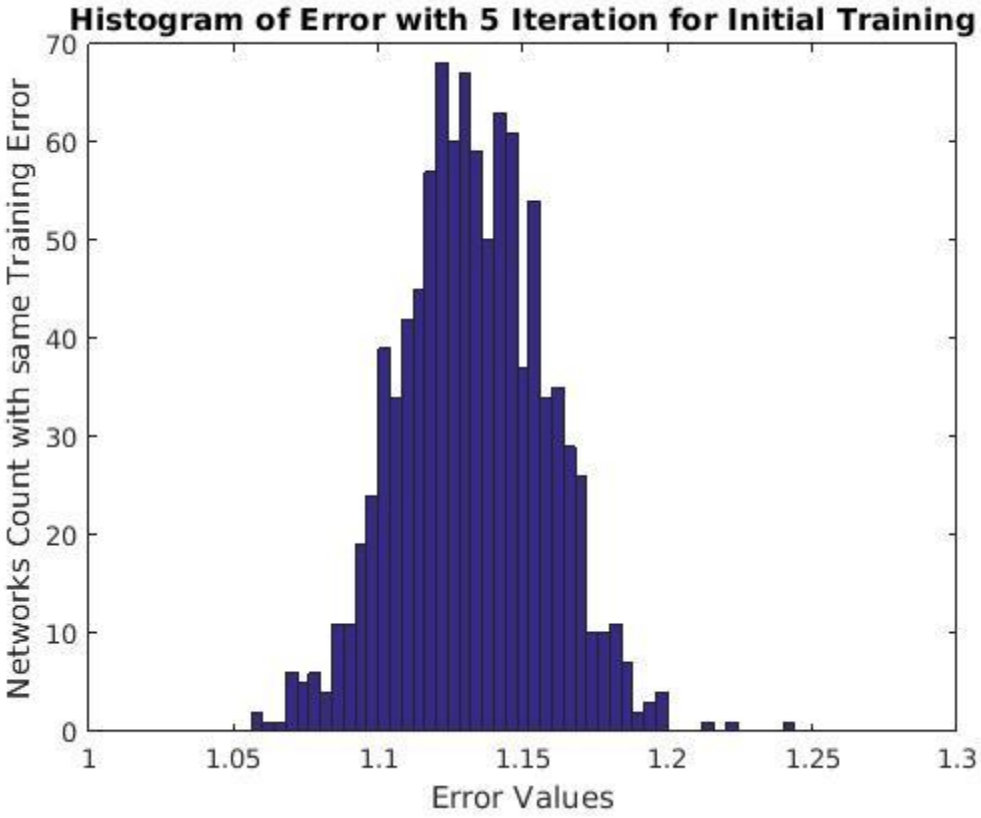

Figure 4.7: Histogram of Error with 5 iterations for initial training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	10	1	490

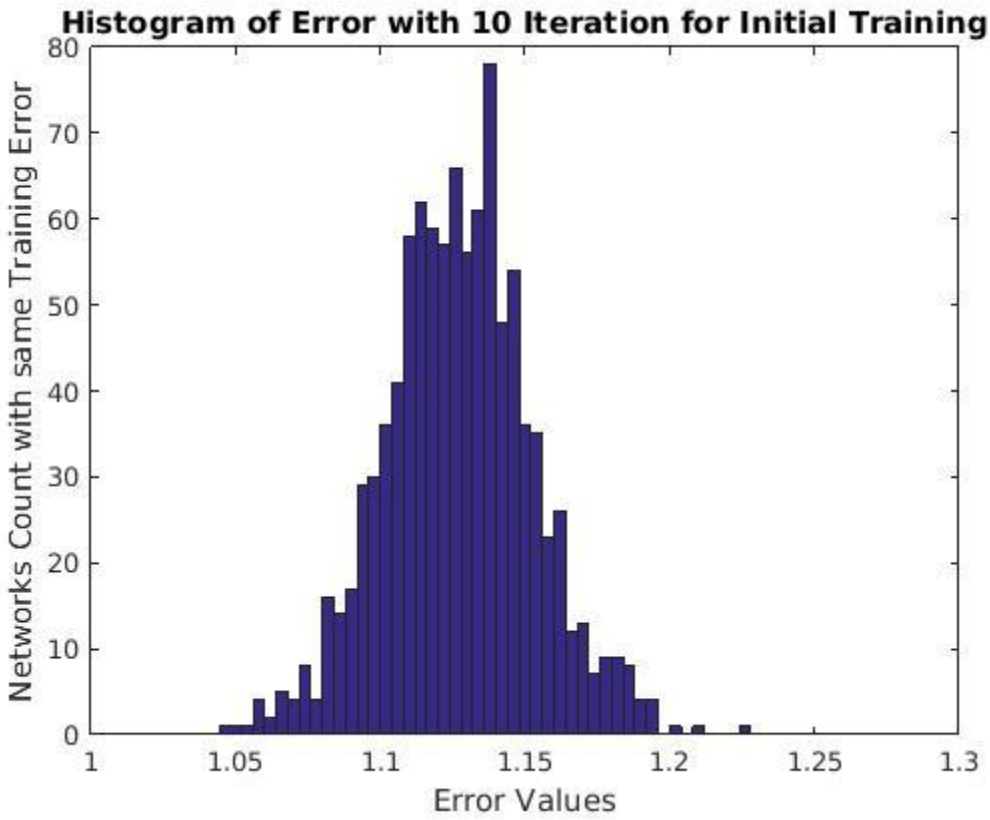

Figure 4.8: Histogram of Error with 10 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	25	1	475

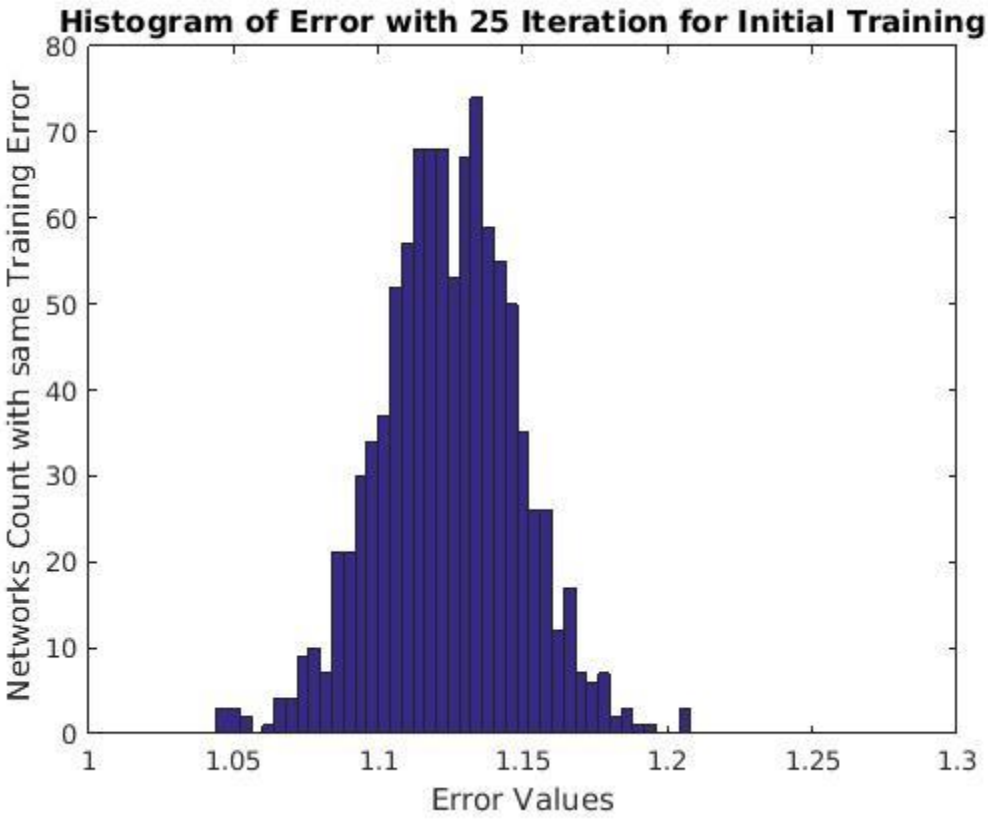

Figure 4.9: Histogram of Error with 25 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	50	1	450

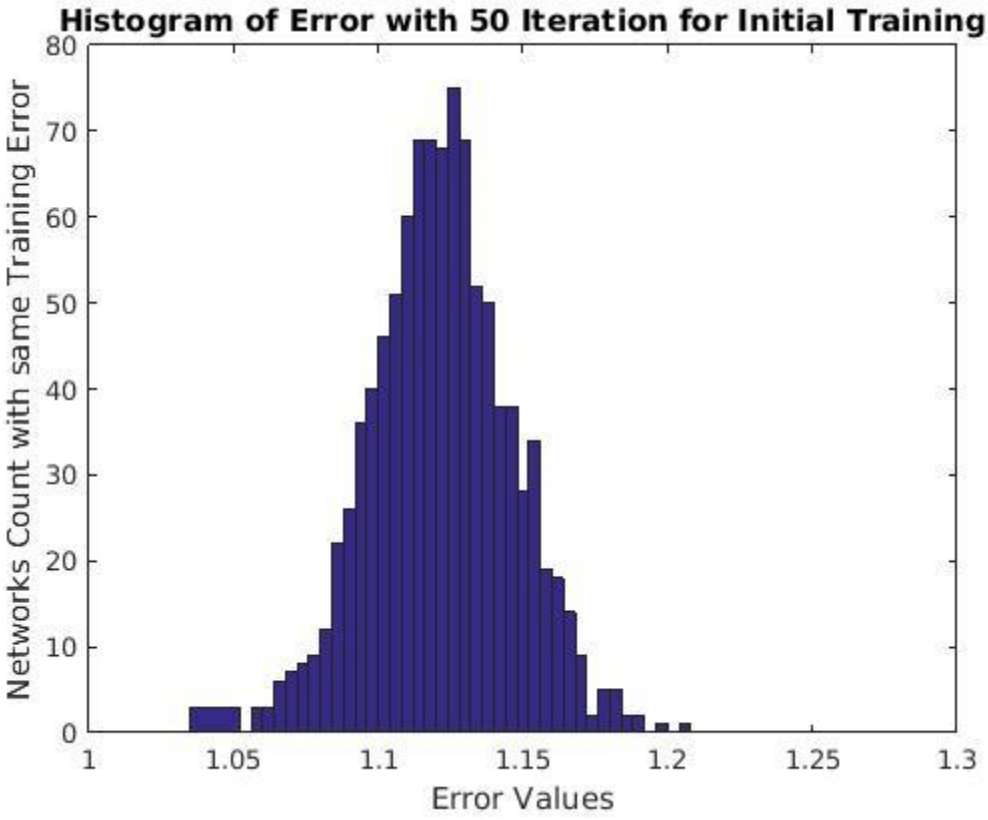

Figure 4.10: Histogram of Error with 50 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	75	1	425

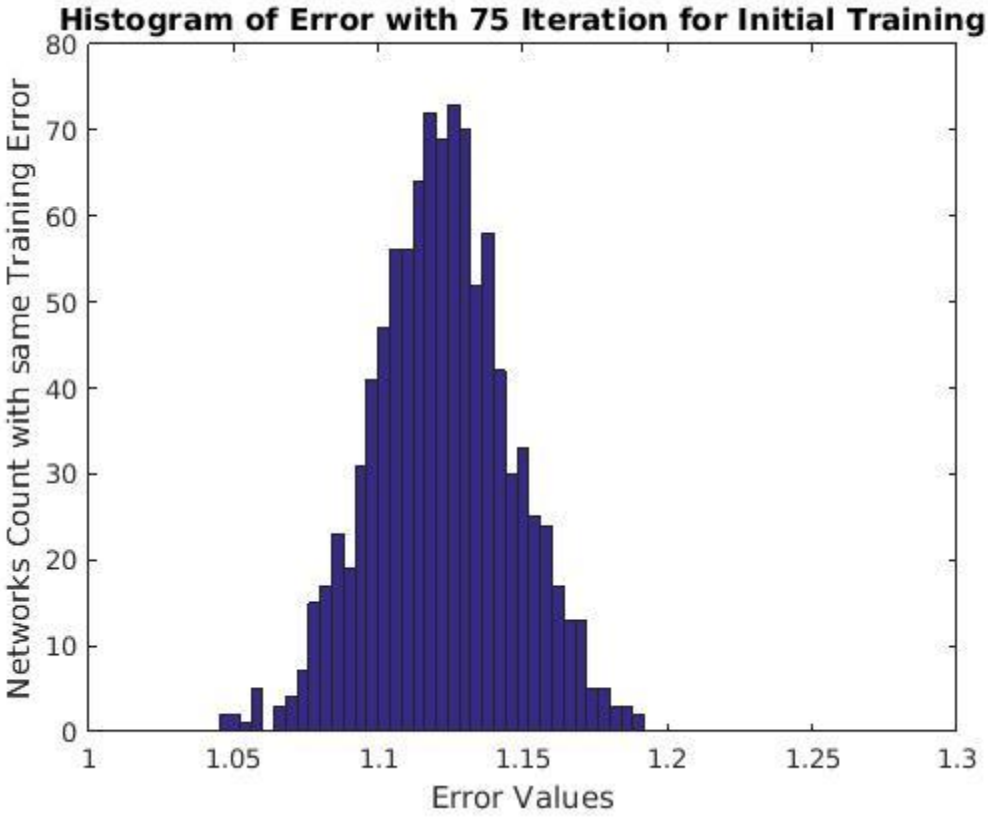

Figure 4.11: Histogram of Error with 75 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	100	1	400

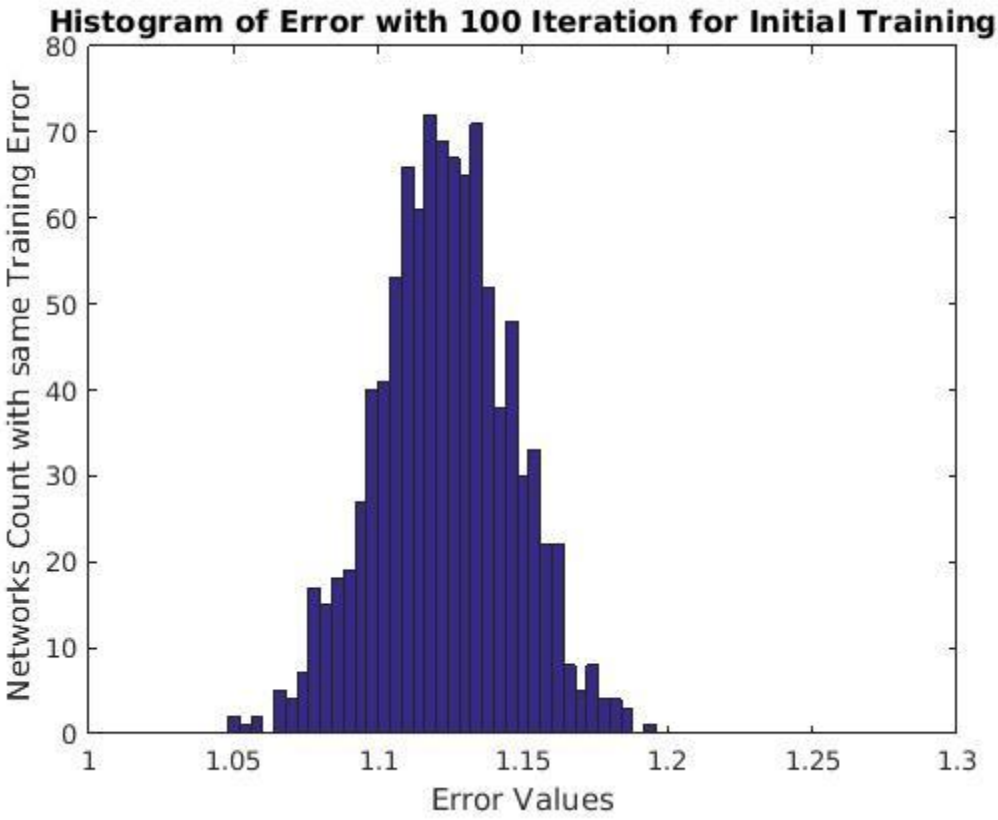

Figure 4.12: Histogram of Error with 100 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	200	1	300

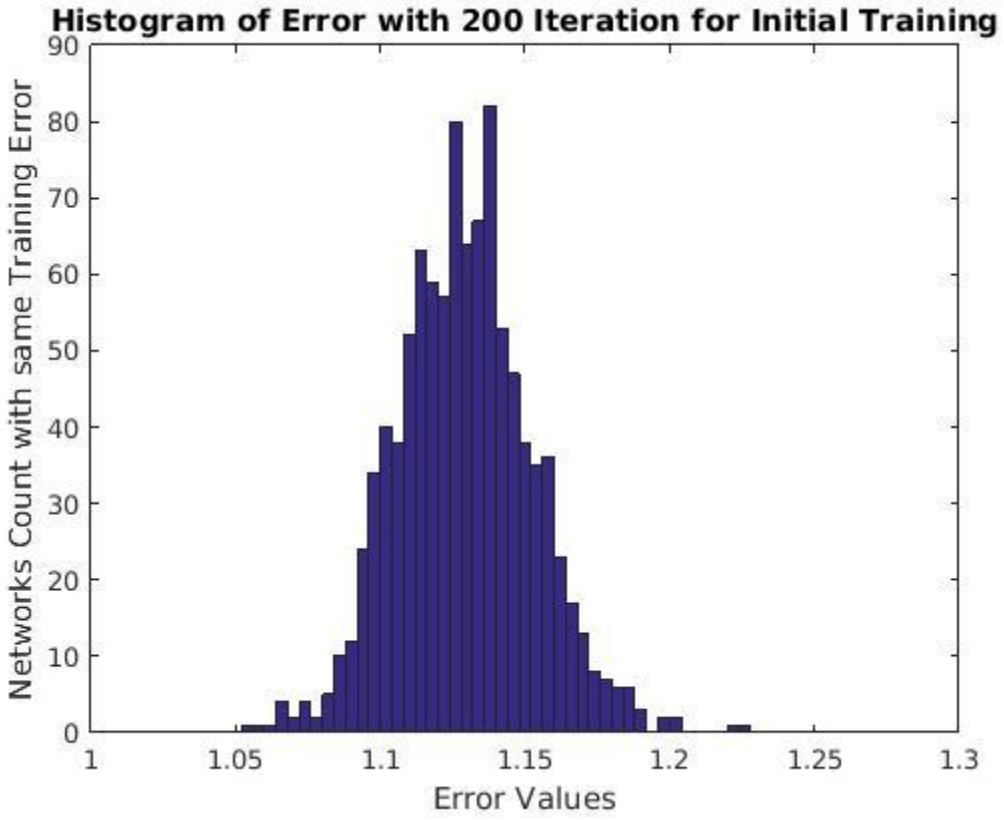

Figure 4.13: Histogram of Error with 200 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	300	1	200

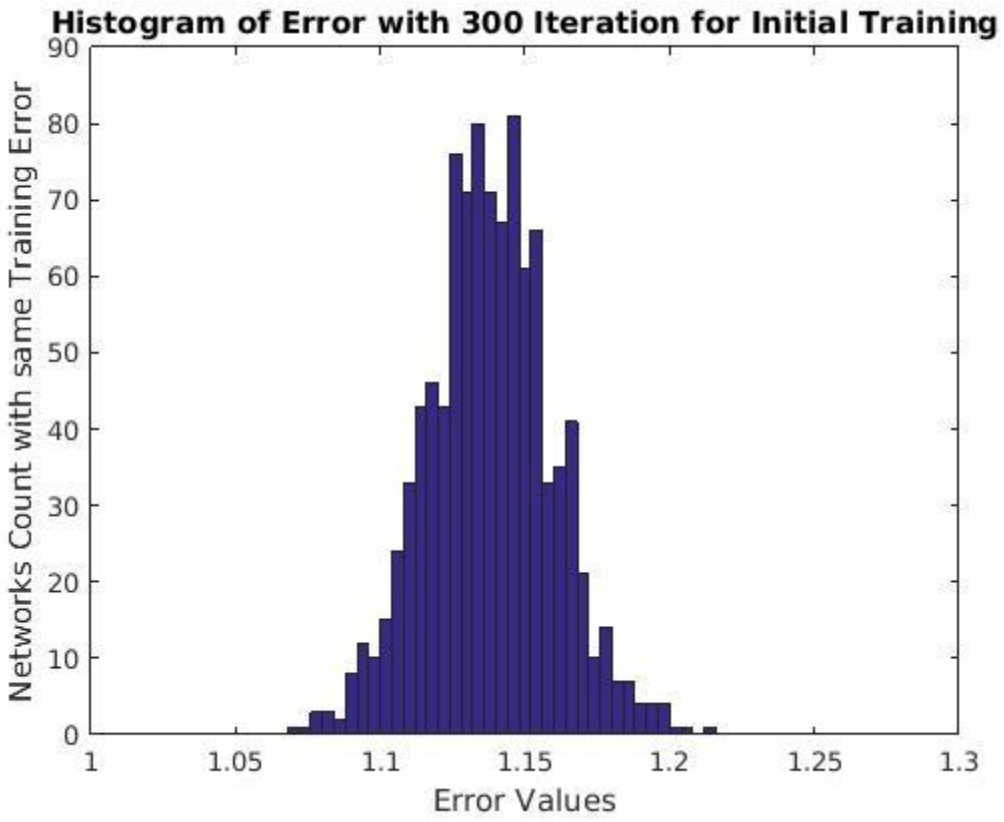

Figure 4.14: Histogram of Error with 300 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	400	1	100

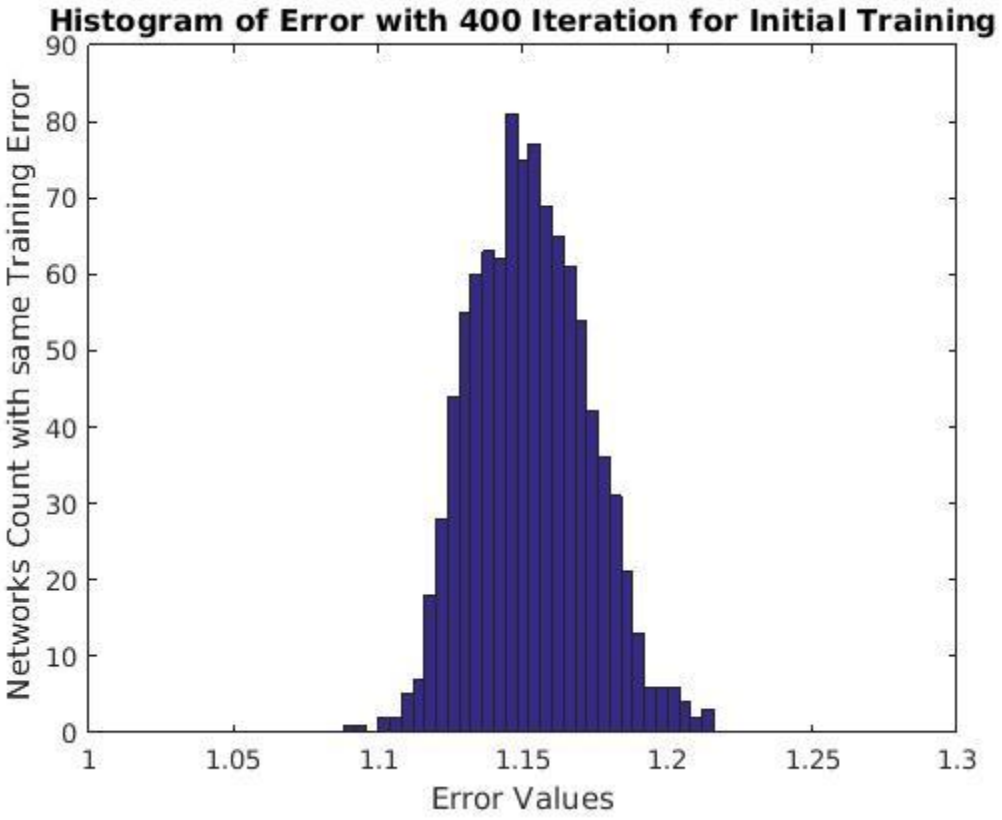

Figure 4.15: Histogram of Error with 400 Iterations for Initial Training

Data file = Oh7.tra Inputs = 20 Outputs = 3 Nh = 51				
Total Nit	**No Of Networks Before Fusion**	**Nit before Fusion**	**No of Network After Fusion**	**Nit after Fusion**
500	3	499	1	1

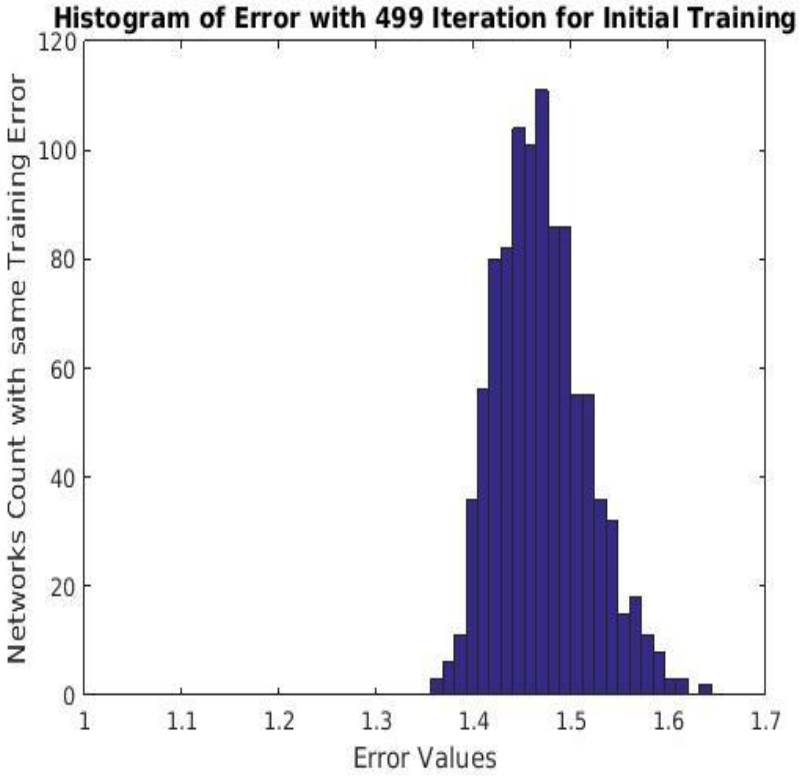

Figure 4.16: Histogram of Error with 499 Iterations for Initial Training

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 4.2 – Error Analysis for Number of Iterations Before Fusion into Single Network using HWO-MOLF		
Number of Iterations before Fusing into Single Network	Standard Deviation of Error	Error Mean
0	0.0276	1.1637
1	0.0253	1.1457
5	0.0251	1.1326
10	0.0254	1.1272
25	0.0236	1.1244
50	0.0241	1.1219
75	0.0237	1.1223
100	0.0232	1.1230
200	0.0234	1.1292
300	0.0216	1.1379
400	0.0200	1.1526
499	0.0463	1.4704

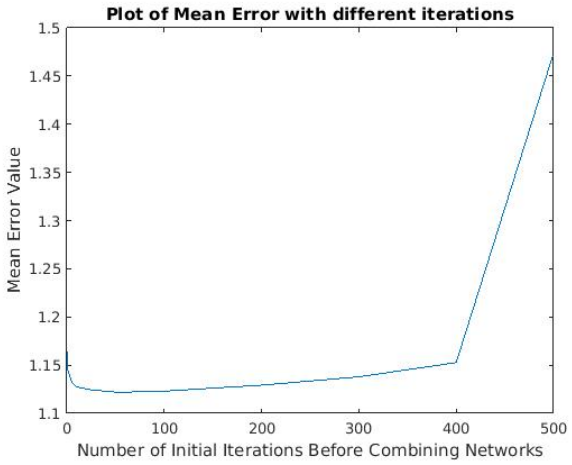

Figure 4.17: Mean Error with Different Iterations

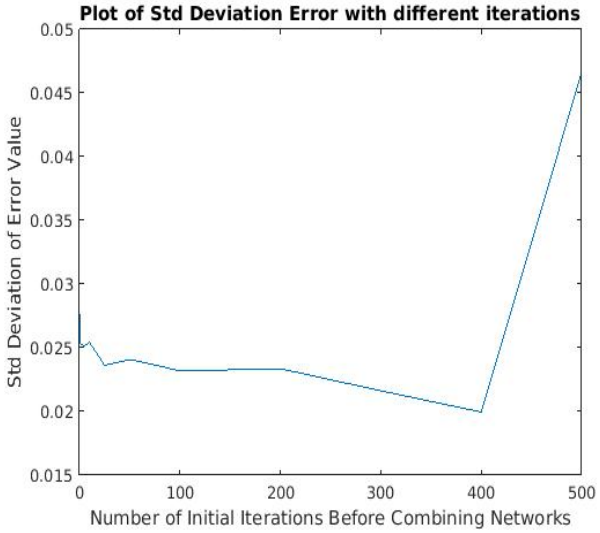

Figure 4.18: Std Deviation of Error with Different Iterations

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 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.

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

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 $i_c(p)$.
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

$$g_a(k) = -\frac{\partial E}{\partial a(k)} = \frac{2}{N_v}\sum_p \sum_{i=1}^{M} \left[t_p`(i) - y_p(i)\right] \frac{\partial y_p(i)}{\partial a(k)} \qquad (5.1)$$

$$\frac{\partial y_p(i)}{\partial a(k)} = w_{oh}(i,k)s`\left(n_p(k)\right) n_p(k)$$

Where, $s`(net) = s(net) * \left(1 - s(net)\right)$

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

$$g_b(k) = -\frac{\partial E}{\partial b(k)} = \frac{2}{N_v}\sum_p \sum_{i=1}^{M} \left[t_p`(i) - y_p(i)\right] \frac{\partial y_p(i)}{\partial b(k)} \qquad (5.2)$$

$$\frac{\partial y_p(i)}{\partial b(k)} = w_{oh}(i,k)s`\left(n_p(k)\right)$$

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

$$O(k) = s(n_p(k)a(k) + b(k))$$

$$O`(k) = O(k) * (1 - O(k))$$

11. Calculate y as,

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{Wo}`$$

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 $\boldsymbol{H}$ for $a_k$and b_k.

17. Use OLS (Orthogonal Least Squares) to solve

$$\boldsymbol{H} \cdot \boldsymbol{z} = \boldsymbol{g}$$

where, z is learning factor for a_k and b_k

$$\boldsymbol{z} = [z_{a1}, z_{a2,} \dots\dots, z_{bNh}]^T$$

$\boldsymbol{g}$ is negative gradient vector

$$\boldsymbol{g} = \left[\frac{-\partial E}{\partial z_{a1}}, \frac{-\partial E}{\partial z_{a2}} \dots\dots, \frac{-\partial E}{\partial z_{bNh}}\right]^T$$

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

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 5.1 – Compares performance of MLP with modified sigmoid method and without modified sigmoid method. N_h = 51. N_{it} = 500. For modified sigmoid method, scaling and addition factor for each hidden neuron are created and trained for 200 iteration.

Dataset	Number of inputs	Number of outputs	Training Error Without Modified Sigmoid Method	Testing Error Without Modified Sigmoid Method	Training Error With Modified Sigmoid Method	Testing Error With Modified Sigmoid Method
Twod	8	7	0.102378	0.1369	0.097071	0.1355
Oh7	20	3	0.101961	0.1175	0.090968	0.1135
Mat	4	4	0.000873	0.0011	0.0004	0.000477
Weather Forecasting	71	3	255.7176	290.0440	253.827720	290.2045

CHAPTER 6

Conclusion

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

[1] Wikipedia contributors. “Machine Learning” Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia.

[2] Wikipedia contributors. “Artificial Neural Network” Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia.

[3] Sudhirkumar Menon, “Growing Training And Initialization For Multilayer Perceptron Neural Network” (2016)

[4] Park, Jooyoung, and Irwin W. Sandberg. "Universal approximation using radial-basis-function networks." *Neural computation* 3.2 (1991): 246-257.

[5] Musavi, Mohamad T., et al. "On the training of radial basis function classifiers." *Neural networks* 5.4 (1992): 595-603.

[6] Niyogi, Partha, and Federico Girosi. "On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions." *Neural Computation* 8.4 (1996): 819-842.

[7] Raudys, Šarūnas. "Evolution and generalization of a single neurone: I. single-layer perceptron as seven statistical classifiers." *Neural Networks*11.2 (1998): 283-296.

[8] Knerr, Stefan, Léon Personnaz, and Gérard Dreyfus. "Single-layer learning revisited: a stepwise procedure for building and training a neural network."*Neurocomputing*. Springer Berlin Heidelberg, 1990. 41-50.

[9] Gardner, Matt W., and S. R. Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences."*Atmospheric environment* 32.14 (1998): 2627-2636.

[10] Ruck, Dennis W., et al. "The multilayer perceptron as an approximation to a Bayes optimal discriminant function." *Neural Networks, IEEE Transactions on* 1.4 (1990): 296-298.

[11] Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems* 2.4 (1989): 303-314.

[12] Rosenblatt, Frank. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. No. VG-1196-G-8. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[13] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. No. ICS-8506. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, 1985.

[14] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature*521.7553 (2015): 436-444.

[15]Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems* 2.4 (1989): 303-314.

[16] http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf page 96

[17] Manry, M. T., Dawson, M. S., Fung, A. K., Apollo, S. J., Allen, L. S., Lyle, W. D., & Gong, W. (1994). Fast training of neural networks for remote sensing. *Remote sensing reviews*, *9*(1-2), 77-96.

[18] Olvera, J., X. Guan, and M. T. Manry. "Theory of monomial networks." In*Proc. Symp. Implicit and Nonlinear Systems*, pp. 96-101. 1992.

[19] Barton, Simon A. "A matrix method for optimizing a neural network." *Neural Computation* 3.3 (1991): 450-459.

[20] Sartori, Michael A., and Panos J. Antsaklis. "A simple method to derive bounds on the size and to train multilayer neural networks." *Neural Networks, IEEE Transactions on* 2.4 (1991): 467-471.

[21] Rohani, Kamyar, Mu-Song Chen, and Michael T. Manry. "Neural subnet design by direct polynomial mapping." *Neural Networks, IEEE Transactions on* 3.6 (1992): 1024-1026.

[22] Dettman, John W. *Mathematical methods in physics and engineering*. Courier Corporation, 2013.

[23] Werbos, Paul. "Beyond regression: New tools for prediction and analysis in the behavioral sciences." (1974).

[24] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. No. ICS-8506. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, 1985.

[25] Yu, Changhua, and Michael T. Manry. "A modified hidden weight optimization algorithm for feedforward neural networks." *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*. Vol. 2. IEEE, 2002.

[26] Jesudhas, P., Manry, M. T., Rawat, R., & Malalur, S. (2011, July). Analysis and improvement of multiple optimal learning factors for feed-forward networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on* (pp. 2593-2600). IEEE.

[27] Pedro Domingos, “A few useful things to know about machine learning” Magazine Communications of the ACM, Volume 55 Issue 10, October 2012 Pages 78-87

[28] M. N. Marsono, M. W. El-Kharashi, and F. Gebali, “Binary LNS-based naïve Bayes inference engine for spam control: Noise analysis and FPGA synthesis”, IET Computers & Digital Techniques, 2008

[29] G. Adomavicius, A. Tuzhilin, “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions”, IEEE Transactions on Knowledge and Data Engineering (Volume: 17, Issue: 6, June 2005)

[30] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[31] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMS. In ICASSP, pages 4688–4691. IEEE, 2011. ISBN 978-1-4577- 0539-7

[32] Chandola, V.; Banerjee, A.; Kumar, V. (2009). "Anomaly detection: A survey". ACM Computing Surveys. 41 (3): 1–58.

[33] Quinlan, J. R. (1987). "Simplifying decision trees". International Journal of Man-Machine Studies.

[34] Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

[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.

[36] Cox, DR (1958). "The regression analysis of binary sequences (with discussion)". J Roy Stat Soc B. 20: 215–242.

[37] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[38] Rakesh Agrawal and Ramakrishnan Srikant, “Fast algorithms for mining association rules.” Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[39] Hartigan, J. A.; Wong, M. A. (1979). "Algorithm AS 136: A K-Means Clustering Algorithm". Journal of the Royal Statistical Society. Series C (Applied Statistics). 28 (1): 100–108.

[40] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) Foundations of Machine Learning, The MIT Press

[41] Dana H. Ballard; Christopher M. Brown (1982). Computer Vision. Prentice Hall.

[42] Rowe, Sam Del (2017-06-12). "SDL Adds Neural Machine Translation to Its Enterprise Translation Server". CRM Magazine. Retrieved 2017-06-23.

[43] D.T. Nguyen et al. “Automatic Image Filtering on Social Networks Using Deep Learning and Perceptual Hashing During Crises”, Proceedings of the 14th ISCRAM Conference – Albi, France, May 2017

[44] "Google's AI beats human champion at Go". CBC News. 27 January 2016. Retrieved 28 January 2016.

[45] F. Amato et al. “Artificial neural networks in medical diagnosis” , J Appl Biomed. 11: 47–58, 2013

[46] LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48

[47] Hinton, Geoffrey. "Overview of mini-batch gradient descent" (PDF). pp. 27–29. Retrieved 27 September 2016.

[48] Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems". Journal of Research of the National Bureau of Standards. 49 (6).

[49] Tieleman, Tijmen and Hinton, Geoffrey (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning

[50] Perla, Joseph (2014). "Notes on AdaGrad"

[51] Sutskever, Ilya; Martens, James; Dahl, George; Hinton, Geoffrey E. (June 2013). Sanjoy Dasgupta and David Mcallester, ed. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th international conference on machine learning (ICML-13). 28. Atlanta, GA. pp. 1139–1147.

[52] Diederik, Kingma; Ba, Jimmy (2014). "Adam: A method for stochastic optimization"

[53] B. G. Don R. Hush, "Progress in Supervised Neural Networks," IEEE SIGNAL PROCESSING MAGAZINE, 1993.

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

[55] Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

[56] M. S. Dawson, A. K. Fung and M. T. Manry, "Surface parameter retrieval using fast learning neural networks," Remote Sensing Reviews, 1993, Vol. 7(1), pp. 1-18.

[57] M. S. Dawson, J. Olvera, A. K. Fung and M. T. Manry, "Inversion of surface parameters using fast learning neural networks," Proc. of IGARSS'92, Houston, Texas, May 1992, Vol II, pp 910 - 912.

[58] An Empirical Model and an Inversion Technique for Radar Scattering from Bare Soil Surfaces," in IEEE Trans. on Geoscience and Remote Sensing, pp. 370-381, 1992.

[59] http://www.uta.edu/faculty/manry/new_mapping.html MAT.TRA data file on the webpage.

[60] Gautam R. Eapi (2015). Comprehensive neural network forecasting system for ground level ozone in multiple regions (Doctoral dissertation).