MULTILAYER PERCEPTRON WITH ADAPTIVE ACTIVATION FUNCTIONS

By

CHINMAY RANE

Presented to the Faculty of 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 UNIVERSITY OF TEXAS AT ARLINGTON

May 2016

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr.Michael T Manry for continuous support and guidance. I am also grateful to Dr. Michael Manry for giving me opportunity to work on different techniques in the IPNNL lab which helped me throughout my research.

I would also like to thank Dr. Alan Davis, Schizas Ioannis for taking time to serve on my thesis committee.

I would like to take this opportunity to thank all members of IPNNL labs and friends.

A very special thank you to Rohit Rawat, Kanishka Tyagi, Sudhirkumar Menon and Son Nguyen for guiding me with all the problems I faced throughout my research.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and encouragement throughout my years of academics. This accomplishment would not have be possible without them.

2016

ABSTRACT

MULTILAYER PERCEPTRON

WITH ADAPTIVE ACTIVATION FUNCTIONS

CHINMAY APPA RANE

The University of Texas at Arlington, 2016

Supervising Professor: Dr. Michael T. Manry

A Multilayer perceptron typically has a fixed nonlinear activation function for each hidden unit. In this thesis, an adaptive activation function for individual hidden unit is designed, where the network learns these activation functions at every iteration using a modern second order algorithm. Methods and algorithms for these adaptive activation functions along with several other techniques for training a multilayer perceptron's weights are discussed.

Comparisons between a multilayer perceptron with sigmoidal activation functions and a multilayer perceptron with piecewise linear activation functions are also discussed. The common activation function used is the sigmoidal activations, but it is still not proven that the sigmoidal activations works best for all the applications. Hence the adaptive activation technique described in this thesis can be used, which learns independently as it passes through the data.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES

Figure Page

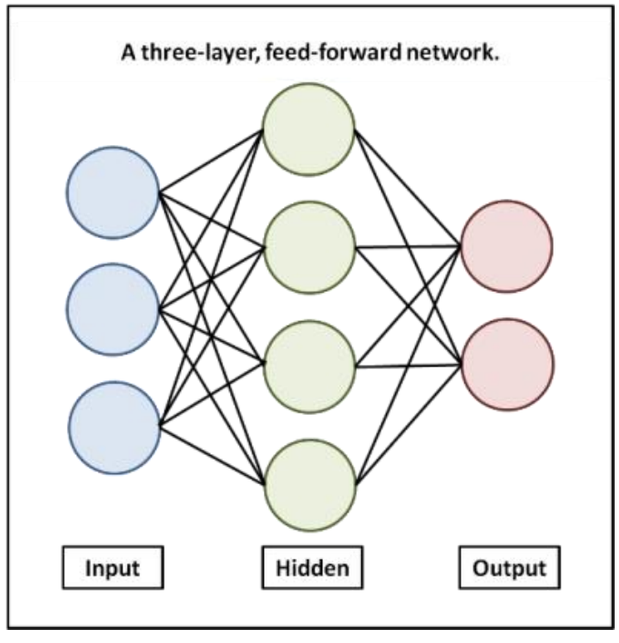
Chapter 1

INTRODUCTION

1.1 Neural Networks

A neural network is a highly interconnected processing element working to solve specific problems [1]. Neural networks are typically organized in 3 different layers. Input layer, hidden layer and the output layer. These layers are interconnected with a number of ‘nodes’ which contain an activation function in the hidden layer. Inputs in the form of patterns are presented to the network through the input layer which is then connected to a hidden layer, where it is actually processed using weighted connections. Similarly the hidden layer is connected to the output layer and is again processed using the same weighted connections, which is the output of the network.

The figure below is a three layer neural network



1.1 Neural Network model

Neural Networks are used in approximation problems [2] such as stock market time series forecasting [3], Currency exchange rate prediction [4], Data Mining [5 6] and Control applications [7], fitness approximation and modeling [54]. They are also used in classification problems such as speech recognition [8], Fingerprint recognition [9], character recognition [10], and face detection [11]. Neural networks are now widely used for Deep learning applications [25].predicting outcome for a patient with colorectal cancer with more accuracy than the current clinical methods [4].

1.3 Properties of the Multilayer Perceptron

In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed-forward network such as multilayer perceptron (MLP) with a single hidden layer containing a finite number of neurons is capable of approximating any measurable continuous function with any desired degree of accuracy. There are no theoretical constraints for the success of the feedforward network. However, lack of success of the feed forward network can be due to inadequate learning, insufficient number of hidden units or lack of a deterministic relationship between inputs and outputs [12]. A neural network's approximation of the Bayes classifier depends upon the training error for the multilayer perceptron which is E(w)

$$E(w) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} \left[t_p(i) - d_{i\,(x_p)} \right]^2 \tag{1.1}$$

and the expected squared error between the networks outputs and Bayes discriminants which is e(w)

$$e(w) = \sum_{j=1}^{N_c} E\left[\left(d_j(x) - b_j(x) \right)^2 \right] \tag{1.2}$$

where,

$d_j(x)$ = the jth multilayer perceptron or functional link output for the p^{th} input vector **x.**

$b_j(x)$ = Type B_3 Bayes optimal discriminant, $\mathrm{P}(j \mid x)$, the probability that the given vector **x** belongs to the jth class.

The theorem states that as the number of patterns N_v increases, the training error $\mathrm{E}(w)$, approaches $\mathrm{e}(w)$ + C, where C is a constant. Here if both the errors are small, it does not mean that the probability of error P_e is small. And the multilayer perceptron tries to approximate a type B_3 bayes classifier, during training [13], Also, a neural net is a type B_3 Bayes discriminant [13, 14] while the Bayes Gaussian is a type B_2 Bayes discriminant [13, 14]. Also good performance of neural net classifiers comes partially from the error criteria used in training [13].

1.5 Problems with Neural Networks

Neural Networks have several problems including the following:

1. Neural network training is usually approached as a nonlinear optimization problem to minimize an error function [41]. For training a large network various gradient techniques often lead to local minimum problems. The local minimum problems are reached not only because of the optimization technique but also because of the initialization of the weights.
2. Hidden layers and the number of hidden neurons plays a vital role in the performance of Back Propagation Neural Network especially in the case where problems related to the arbitrary decision boundary to arbitrary accuracy with rational activation functions are encountered. Also, multiple hidden layers can approximate any smooth mapping to any accuracy. The process of deciding the number of hidden layers and number of neurons in each hidden layer is still confusing.[15]
3. The activation function is one of the elements in a neural network. Selection of the activation functions in a neural network plays an essential role on the network performance. A lot of studies have been conducted by researchers to investigate special activation functions to solve different

kinds of problems [16]. For different applications, different activation functions are found to be suitable. So one cannot always pick the best activation function for a particular application.

1.6 Objectives of this Thesis

In this thesis, a method for solving problem 3 by using a piecewise linear activation function that is potentially different for each hidden unit is presented**.** In Chapter 2, introduction to notation and a basic training algorithm for a multilayer perceptron is presented. In Chapter 3 Introduction of an initial piecewise linear network model for the hidden units, including a method for training the hidden unit activations are discussed. In Chapter 4 improvement in training our networks due to adaptable piecewise linear activations, Training and testing errors are found for the new method and the original sigmoid networks for several different data sets are shown. In Chapter 5 final conclusions and possible enhancements to this work are discussed.

Chapter 2

BASIC MULTILAYER PERCEPTRON

In this chapter a multilayer perceptron [16 - 22] with sigmoidal activation functions, its notations and input, output weights training method are discussed. In the end of this chapter a detailed multilayer perceptron algorithm used throughout this thesis is discussed.

2.1 Notation and Processing for a Multilayer Perceptron

Three layer artificial neural networks consist of an N dimensional input layer storing the input vector, $\boldsymbol{x_p}$, an M dimensional output layer storing the output vector, $\boldsymbol{y_p}$, and one hidden layer storing the N_h dimensional net vector $\boldsymbol{n_p}$ and the activation vector $\boldsymbol{o_p}$. Let the input vectors be augmented by an extra element x_p (N + 1) =1, in order to handle hidden and output layer thresholds, so $\boldsymbol{x_p}$ = $[x_p\,(1), x_p\,(2) \ldots x_p\,(\mathrm{N}+1)]^T$ .The pattern number p varies from 1 to N_v. Additional parameters are $w(\mathrm{k}, \mathrm{n})$, $w_{oh}(\mathrm{i}, \mathrm{k})$ and $w_{oi}(\mathrm{i}, \mathrm{n})$. Input weights w(k, n) connect the n[th] input to the k[th] hidden unit .Output weights w_{oh}(i, k) connect the k[th] hidden unit's 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 unit's net function is

$$n_p(k) = \sum_{n=1}^{N+1} w(k,n) \cdot x_p(n) \tag{2.1}$$

which can be summarized as

$$\boldsymbol{n_p} = \boldsymbol{W} \cdot \boldsymbol{x_p} \tag{2.2}$$

here $\boldsymbol{n_p}$ denotes the N_h dimensional column vector of the net function values and $\boldsymbol{W}$ is N_h by (N+1) matrix. For the p[th] pattern the corresponding k[th] hidden unit activation $o_p(k)$ is $o_p(k)$ = f($n_p(k)$)), where f denotes the hidden layer activation and $\boldsymbol{o_p}$ is the N_h dimensional hidden unit activation vector. The activation function used is the sigmoidal activation.

The output of the k[th] hidden unit is given by

$$o_p(k) = f\left(n_p(k)\right) = \frac{1}{1 + e^{-n_p(k)}} \tag{2.3}$$

where the output of the k[th] hidden unit lies between '0' and '1'

For the p[th] pattern, the i[th] element $y_p(i)$ of the M-dimensional output vector $\boldsymbol{y_p}$ thus becomes,

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

which can be summarized as,

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

where the last rows of $\boldsymbol{W_{oh}}$ and $\boldsymbol{W_{oi}}$ respectively store the hidden unit and output threshold values.

A typical error function used in batch mode MLP training is the mean-squared error (MSE) described as

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

2.2 Basic Initialization

The first step towards training a multilayer perceptron [16-22] is randomly initializing the input weights of the network. If some inputs have much larger standard deviations than others, they can dominate the training. So to avoid that the input standard deviations and normalizing it by using the net control method is done.

2.2.1 Net Control

Training of input weights is strongly dependent on the slopes of hidden unit activation functions in response to inputs [23]. Training of the weights ceases if the unit it feeds into has an activation function derivative of zero for all patterns. Therefore it is important to adjust the mean and standard deviations of all hidden unit net functions so that they have values of m_d = 0.5 and σ_d = 1, Here m_d denotes the desired mean and σ_d is the desired standard deviation. The following procedure is referred to as net control. This net control can be used for the multiple hidden layer case, but in this section net control for one hidden layer case is discussed.

The algorithm for the net control for one hidden layer is described below as,

- For a given hidden layer, make a pass through the data, where $m(k)$ and $\sigma(k)$ are calculated which are the net function mean and standard deviation of the kth hidden units.

- For the k[th] hidden unit, multiply the threshold and all the input weights by $\frac{\boldsymbol{\sigma_d}}{\boldsymbol{\sigma(k)}}$ to adjust the net functions standard deviation to the desired value.
- The final step is to update the thresholds using the following equation.

$$\theta(k) = \theta(k) - m(k) \cdot \frac{\sigma_d}{\sigma(k)} + m_d \tag{2.7}$$

where, θ denotes threshold and k denotes hidden unit number.

2.2.2 Output Weight Optimization (OWO)

Here after net control, the output weights are found using output weight optimization [24, 25] as follows First the output vector can be defined as,

$$\boldsymbol{y_p} = \boldsymbol{W_o} \cdot \boldsymbol{X_p} \tag{2.8}$$

where,

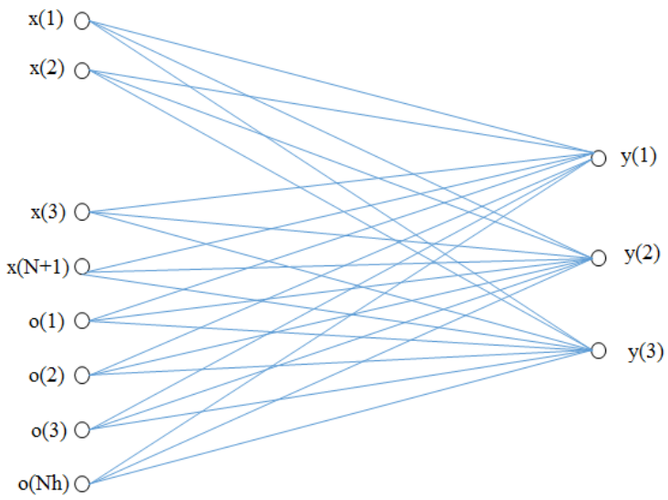
$$\boldsymbol{W_o} = [\boldsymbol{W_{oi}} : \boldsymbol{W_{oh}}] \tag{2.9}$$

Here the bias and hidden weight matrices are augmented to form $\mathbf{W_o}$ to use functional link net type OWO calculation for better results,

$$\boldsymbol{X_p} = \left[\boldsymbol{x_p}^T, \boldsymbol{o_p}^T\right]^T \tag{2.10}$$

As the weights are augmented in equation (2.9), similarly inputs are also augmented with respective to the weight matrices.

To support the above equations the output of the MLP can be drawn as an FLN type.



2.1– FLN type training

The total number of basis functions L is N + N_h + 1, where N is the number of network inputs, N_h is the number of hidden units in a hidden layer, Also a threshold is added. This basis function L is defined as follows,

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

X(N + 1) = 1, and

X(N+1+k) = $o_p(k)$ for k between 1 andN_h

here M (output) sets of L equations in L unknowns are proven, so for solving the output weights, the gradient of equation (2.6) with respect to the output weights is found using,

$$\frac{\partial E}{\partial w_o} = -\frac{2}{N_v}\sum_{p=1}^{N_v}[t_p - y_p] \cdot \frac{\partial y_p}{\partial w_o} = 0 \qquad (2.11)$$

using equation (2.8) and (2.11) it is solved as,

$$\frac{\partial E}{\partial w_o} = -2\left[\frac{1}{N_v}\sum_{p=1}^{N_v} t_p \cdot x_p - w_o \cdot \sum_{p=1}^{N_v} x_p \cdot x_p\right] \quad (2.12)$$

this leads to the following linear equation,

$$\boldsymbol{R} \cdot \boldsymbol{W_o}^{\boldsymbol{T}} = \boldsymbol{C} \quad (2.13)$$

where $\boldsymbol{W_o}$ is the output weight matrix of size $M \times (\mathrm{N} + 1 + \mathrm{N_h})$, where M is the number of network outputs. R is the $(\mathrm{N} + 1 + \mathrm{N_h}) \times (\mathrm{N} + 1 + \mathrm{N_h})$ autocorrelation matrix which is defined in equation (2.12) as,

$$r(k,n) = \frac{1}{N_v}\sum_{p=1}^{N_v} X_p(k) \cdot X_p(n) \quad (2.14)$$

and C is the $(\mathrm{N} + 1 + \mathrm{N_h}) \times (M)$ cross-correlation matrix which is defined in equation (2.12) as,

$$c(k,i) = \frac{1}{N_v}\sum_{p=1}^{N_v} X_p(k) \cdot t_p(i) \quad (2.15)$$

The linear equation (2.12) can be solved using conjugate gradient method [28] or orthogonal least squares (OLS) method[26, 27]. After solving for all output weights, the error should be measured to determine the network's improvement.

2.3 Backpropagation for Input weights

Backpropagation [29-32] is a common method for updating the input weights. In this section steepest descent which is the first order optimization algorithm is used. The gradient for input weights are calculated as follows,

Elements of the negative gradient matrix $\boldsymbol{G}$ with respect to equation (2.6) are calculated as,

$$g(k,n) = \frac{-\partial E}{\partial w(k,n)} \tag{2.16}$$

For the p[th] pattern, the output and hidden layer delta functions $\delta_{po}(i)$, $\delta_p(k)$ [33] are respectively found as

$$\delta_{po}(i) = 2 \cdot \left(t_p(i) - y_p(i)\right) \tag{2.17}$$

$$\delta_p(k) = f'\left(n_p(k)\right) \cdot \sum_{i=1}^{M} \delta_{po}(i) \cdot w_{oi}(i,k) \tag{2.18}$$

where $f'\left(n_p(k)\right)$ is the first derivative of hidden unit activation and $\boldsymbol{\delta_p} = \left[\delta_p(1), \delta_p(2), \ldots, \delta_p(N_h)\right]^T$.

The matrix of the negative partial derivatives can be written as

$$\boldsymbol{G} = \frac{\boldsymbol{1}}{\boldsymbol{N_v}} \sum_{\boldsymbol{p=1}}^{\boldsymbol{N_v}} \boldsymbol{\delta_p} \left(\boldsymbol{x_p}\right)^{\boldsymbol{T}} \tag{2.19}$$

As steepest descent is used to modify the input weights, the optimal learning factor (OLF) z which is derived using a Taylor series expansion of the mean square error E expressed in terms of z is as follows,

Optimal Learning factor for input weights

Using the gradient ***G***, the optimal learning factor as follows, The first partial of E with respect to z is

$$\frac{\partial E}{\partial z} = \frac{-2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[t_p(i) - y_p(i) \right] \cdot \frac{\partial y_p(i)}{\partial z} \tag{2.20}$$

where,

$$\frac{\partial y_p(i)}{\partial z} = \sum_{k=1}^{N_h} w_{oh}(i) \cdot f'\left(n_p(k)\right) \sum_{n=1}^{N+1} g(k,n) \cdot x_p(n) \tag{2.21}$$

where, $f'\left(n_p(k)\right)$ is the first derivative of a hidden unit activation and $g(k,n)$ is the gradient matrix calculated in equation (2.19). Also the Gauss-Newton approximation of the second partial is

$$\frac{\partial^2 E(z)}{\partial z^2} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[\frac{\partial y_p(i)}{\partial z} \right]^2 \tag{2.22}$$

Thus the learning factor is calculated as

$$z = \frac{-\frac{\partial E}{\partial z}}{\frac{\partial^2 E}{\partial z^2}} \tag{2.23}$$

After finding the optimal learning factor the input weights and the threshold contained in $\boldsymbol{W}$ in a given iteration are updated as,

$$\boldsymbol{W = W + z \cdot G} \tag{2.24}$$

which can be written as,

$$\nabla \boldsymbol{W} = z \cdot \boldsymbol{G} \tag{2.25}$$

where z is a scalar optimal learning factor and $\boldsymbol{G}$ is the gradient matrix calculated in equation (2.19)

OWO-BP Algorithm

- Initialize $\boldsymbol{W}, \boldsymbol{W_{oi}}, \boldsymbol{W_{oh}}$
- Solve for linear equations using OLS and update the output weights.
- **for** it = N_{it}, solve for
- Find the input gradient matrix $\boldsymbol{G}$ from equation (2.19)
- Compute the optimal learning factor z using (2.18)
- Calculate the change in input weights and update as in equation (2.23)
- Solve again for output weights using linear equations of (2.13) using OLS or conjugate gradient
- **End for**

2.4 Hidden Weight Optimization (HWO)

In this section, the hidden weight optimization method [34, 35, 36] are described and also the OWO-HWO algorithm.

In HWO method, the hidden weights are updated by minimizing a separate error function for each hidden unit. These defined error functions use the difference between the desired and the actual net function. For the p^{th} pattern, the desired net function $n_{pd}(k)$ is

$$n_{pd}(k) = n_p(k) + z \cdot \Delta n_p(k) \cong n_p(k) + z \cdot \delta_p(k) \tag{2.26}$$

where z is the optimal learning factor and $\delta_p(k)$ is the delta function from equation (2.18)

The hidden weights are updated as

$$w(k,n) = w(k,n) + z \cdot d(k,n) \tag{2.27}$$

where $d(k,n)$ is the hidden weight change vector element, the weight changes are derived using

$$n_p(k) + z \cdot \delta_p(k) \cong \sum_{n=1}^{N+1} [w(k,n) + z \cdot d(k,n)] \cdot x_p(n) \tag{2.28}$$

Therefore,

$$\delta_p(k) \cong \sum_{n=1}^{N+1} d(k,n) \cdot x_p(n) \tag{2.29}$$

The error of (2.29) for the k[th] hidden unit is measured as

$$E_\delta(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} \left[\delta_p(k) - \sum_{n=1}^{N+1} d(k,n)\, x_p(n) \right]^2 \tag{2.30}$$

Setting to zero the derivative of $E_\delta(k)$ with respect to $d(k,n)$,

$$\boldsymbol{DR_i = G} \tag{2.31}$$

r_i is the autocorrelation matrix which is defined as,

$$r_i(k,n) = \frac{1}{N_v} \sum_{p=1}^{N_v} x_p(k) \cdot x_p(n) \tag{2.32}$$

So instead of directly using **G** to update the hidden weights as in equation (2.19), HWO minimizes a separate error function described in equation (2.30), it solves the linear equations in (2.31) and then updates the hidden weights using

$$\boldsymbol{W = W} + z \cdot \boldsymbol{D} \tag{2.33}$$

Equation (2.31) can be written as

$$\boldsymbol{D = G R_i^{-1}} \tag{2.34}$$

OWO-HWO Algorithm

- Initialize $\boldsymbol{W}, \boldsymbol{W_{oi}}, \boldsymbol{W_{oh}}$
- Solve for linear equations using OLS and update the output weights.
- **for** it = Number of iterations, solve for
- Find the input gradient matrix $\boldsymbol{G}$ from equation (2.19).
- Solve for the linear equations of (2.32) to find **D** matrix using Hidden weight optimization method.
- Compute the optimal learning factor z using (2.18) and **D** matrix.
- Calculate the change in input weights and update as in equation (2.23).
- Solve again for output weights using linear equations of (2.13) using OLS or conjugate gradient.
- **End for**

2.5 Multiple Optimal Learning factor for Input weights (MOLF)

Multiple Optimal Learning Factor (MOLF) [37, 38] is a second order training algorithm, where a different learning factor z_k is used to update weights feeding into the k^{th} hidden unit. The input weight connecting the n^{th} input to the k^{th} hidden unit is updated using

$$w(k,n) = w(k,n) + z_k \cdot d(k,n) \tag{2.35}$$

where, z_k denotes the learning factor that corresponds to the k^{th} hidden unit. The vector z_k can be found using OLS using the following relation

$$\boldsymbol{H_{molf}} \cdot \boldsymbol{z} = \boldsymbol{g_{molf}} \tag{2.36}$$

where,

$$g_{molf}(j) = \frac{-\partial E}{\partial z_j} \tag{2.37}$$

and

$$h_{molf}(l,j) = \frac{\partial^2 E}{\partial z_l z_j} \tag{2.38}$$

The net function vector $\boldsymbol{n_p}$ can be related to its input weights as

$$n_p(k) = \sum_{n=1}^{N+1} \big(w(k,n) + z(k) \cdot d(k,n)\big) \cdot x_p(n) \tag{2.39}$$

The total output vector $y_p(m)$ to be minimized is given as,

$$y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m,n)\, x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k) f\left(\sum_{i=1}^{N+1} \big(w(k,i) + z_k \cdot d(k,i)\big)\, x_p(i)\right) \tag{2.40}$$

The negative partial of E with respect to $\boldsymbol{z_j}$ is

$$-\frac{\partial E}{\partial z_j} = g_{molf}(j) = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \Big(t_p(i) - y_p(i)\Big) \cdot \frac{\partial y_p(i)}{\partial z_j} \tag{2.41}$$

where,

$$\frac{\partial y_p(i)}{\partial z_j} = w_{oh}(i,j) \,.\, f'\Big(n_p(j)\Big) \cdot x_p(i) \cdot d(j,i) \tag{2.42}$$

where, $f'\Big(n_p(j)\Big)$ is the first derivative of the hidden unit activation .Now Using the Gauss – Newton updates, the second partial derivate elements of the Hessian $\boldsymbol{h_{molf}(l,j)}$ of equation (2.36) are,

$$h_{molf}(l,j) = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} w_{oi}(m,l)\, w_{oi}(m,j) \sum_{p=1}^{N_v} \Big(f'\Big(n_p(j)\Big) \cdot x_p(m) \cdot d(j,m) \cdot f'\Big(n_p(l)\Big) \cdot x_p(m) \cdot d(l,m)\Big) \tag{2.43}$$

Thus the multiple optimal learning factor which is a vector of $(N_h \times 1)$ is found from the negative gradient vector and hessian matrix from the equation (2.34) using OLS, where $\boldsymbol{H_{molf}}$ is a $(N_h \times N_h)$ matrix and $\boldsymbol{g_{molf}}$ is a $(N_h \times 1)$ vector.

2.6 Regular HWO-MOLF algorithm description

Using the OWO, HWO and MOLF weight training methods, the Regular HWO-MOLF algorithm is constructed as follows,

1. Perform net control from section (2.2.1) and update the output weights using output weight optimization method by solving the linear equation from (2.13).
2. Start of the iterations, increment i_t by 1.
3. Perform backpropagation from section (2.3) and calculate the input gradient $\boldsymbol{G}$ using equation (2.19).
4. Solve the linear equations of (2.29) to find the D matrix using Hidden weight optimization method and update the G matrix as the D matrix.
5. Calculate $\boldsymbol{z}_k$ using OLS, and update the inputs weights using equation (2.34) and equation (2.33).
6. Again update the output weights using output weight optimization method from section (2.2.2).
7. Perform backtracking if necessary.
8. Go to step 3 and continue until $i_t = N_{it}$

Chapter 3

PIECEWISE LINEAR ACTIVATIONS

This chapter presents a new type of activation functions and gives a strong theoretical foundation to its training. Methods are given for training a network that uses the new activations. These methods are enhanced by the use of second order methods.

3.1 Initialization and Notations for New Piecewise Linear Activations

In this section, the initialization of a new network is discussed. For initialization, the random piecewise linear samples and its activation values are found, The activation function used is the sigmoidal activations. The random piecewise linear samples are chosen between '+4' and '-4' and the activations for these fixed samples are calculated using equation (2.3). The sigmoid is a continuous non-linear activation function whose outputs for all real input values fall within the range of 0 and 1 [43]. The basic idea behind choosing the random samples between '+4' and '-4' is that the sigmoid activation value of '+4' is '0.9820' and for '-4' is '0.0180' which is close to the 0 to 1 range of the property of the sigmoid function, A different range for the fixed sample values can be selected.

The piecewise linear samples can be calculated using various methods. The method used in this thesis is calculating samples that are equidistant from each (example: a fixed sample distance of 0.4) for a range of +4 to -4.

The following table shows the piecewise linear samples from +4 to -4 at a sample size of 0.4 and its respective activation values for 1 hidden unit. Here, only the piecewise linear activation values are trained,

so the size of the piecewise linear activation value is the total number of piecewise linear samples × N_h hidden units. Also the fixed piecewise linear samples are kept constant throughout the training, so the size is total number of piecewise linear samples × 1.

Sample	1	2	3	4	5	6	7	8	9	10
Fixed Piece-wise linear samples(ns)	-4.0	-3.6	-3.2	-2.8	-2.4	-2.0	-1.6	-1.2	-0.8	-0.4
Piecewise linear activations (a)	0.018	0.026	0.039	0.057	0.083	0.119	0.168	0.231	0.310	0.4013

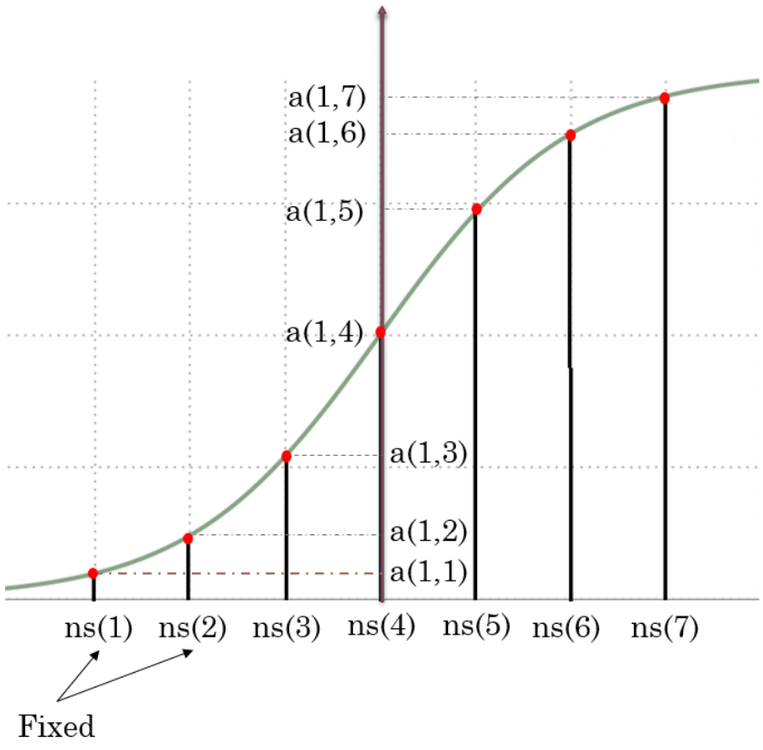
Table 3.1 - Fixed piecewise linear samples and its activation values (part 1)

Sample	11	12	13	14	15	16	17	18	19	20	21
Fixed Piece-wise linear samples(ns)	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
Piecewise linear activations (a)	0.500	0.598	0.690	0.768	0.832	0.880	0.916	0.942	0.960	0.973	0.982

Table 3.2 - Fixed piecewise linear samples and its activation values (part 2)

From the table, the first row is the sample number, where 21 samples are selected. The number of samples can vary according to preference. The second row are the fixed piecewise linear samples ***ns*** and the third row are it's piecewise linear activations ***A*** to be trained. Here the piecewise linear samples are same for all the hidden units during initialization and their activations for each hidden units are trained independently for every iteration. The algorithm for training these activations are explained in this chapter.

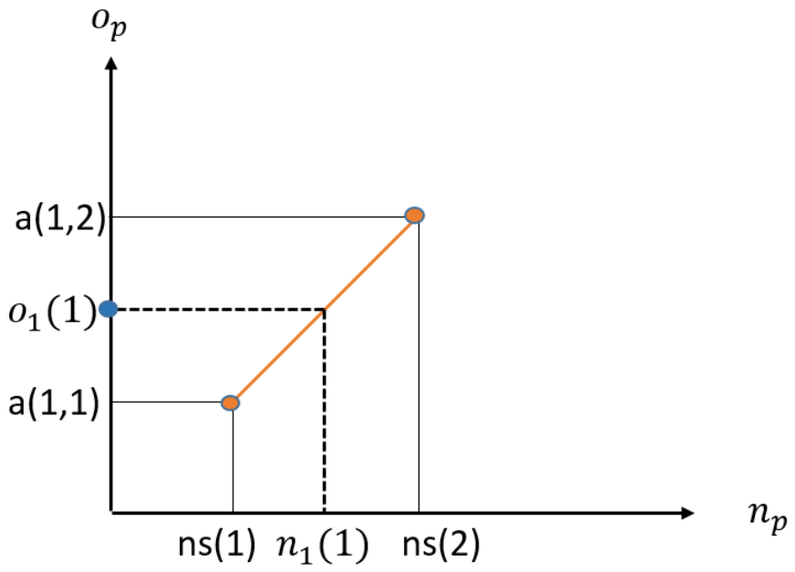
Fixed Piecewise sigmoid activations



3. 1 Fixed piecewise sigmoid activations

The above figure is the plot for a fixed piecewise sigmoid activation for net versus activations values where the sigmoid curve is sampled at a fixed rate on the sigmoid curve, the main idea is to use 2 of the fixed points using the interpolation method to adjust all the points between those fixed samples in a straight line between those 2 points.

Linear interpolation for sigmoid curve



3. 2 linear interpolation between 2 fixed known points

Linear interpolation involves estimating a new value of a function between two known fixed points [39]. Which means that if two known points are given by the coordinates $(ns(1), a(1,1))$ and $(ns(2), a(1,2))$, the linear interpolant is the straight line between these two points.

The above figure relates the use of linear interpolation between 2 fixed points for calculating the activation vector $\boldsymbol{o_p}$ from the sigmoid curve. Here ns(1) and ns(2) are the fixed samples from the net vector $\boldsymbol{n_p}$ on x axis and a(1,1) and a(1,2) are its respective sigmoid activation values on y axis. $n_1(1)$ from the figure is the 1st pattern of 1st hidden unit of the net vector and $o_1(1)$ is its activation, which has to be found using above concept.

For a value $n_1(1)$ in the interval $(ns(1), ns(2))$, the activation value $o_1(1)$ along the straight line is given from the equation.

$$o_1(1) = \frac{ns(2) - n_1(1)}{ns(2) - ns(1)} \cdot a(1,1) + \frac{n_p(1) - ns(1)}{ns(2) - ns(1)} \cdot a(1,2) \tag{3.1}$$

Using the same concept and different fixed samples, the activation vector $\boldsymbol{o_p}$ for all the hidden units can be calculated as,

The steps to calculate the output of each of the k^{th} hidden units $o_p(k)$ are,

- Find sample position number m_1 and m_2 using the search algorithm.
- Calculate $o_p(k)$ for the k^{th} hidden unit using linear interpolation.

The formula from the interpolation method can be re written for p patterns, different piecewise linear samples and N_h hidden units as,

$$o_p(k) = \frac{ns(m_2) - n_p(k)}{ns(m_2) - ns(m_1)} \cdot a(k, m_1) + \frac{n_p(k) - ns(m_1)}{ns(m_2) - ns(m_1)} \cdot a(k, m_2) \tag{3.2}$$

where for the gradient and learning factor calculations, the above equation can be substituted as,

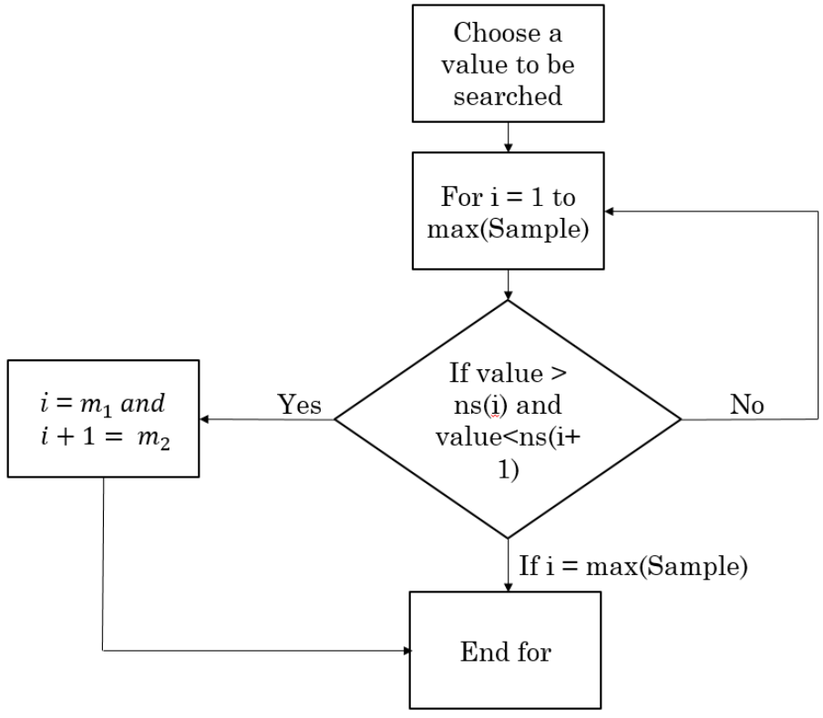
$$\frac{ns(m_2) - n_p(k)}{ns(m_2) - ns(m_1)} = w_{1p}(k) \tag{3.3}$$

and

$$\frac{n_p(k) - ns(m_1)}{ns(m_2) - ns(m_1)} = w_{2p}(k) \tag{3.4}$$

Search algorithm block diagram for position m_1 and m_2

In this block diagram the sample position m_1 and m_2 of the p^{th} element of net vector $\boldsymbol{n_p}$ from the fixed piecewise linear samples table are found, where m_1 and m_2 are the positions between which the p^{th} element of net vector $\boldsymbol{n_p}$ lies. .



For better understanding an example is provided for calculating the activation vector $\boldsymbol{o_p}$ as follows,

Suppose, the p^{th} pattern of the 1st hidden unit of net, $n_p(1)$ is '**1.425**', From table 3.1, 3.2 and search algorithm position the p^{th} element of net, $n_p(1)$, 1.425 lies between $m_1 = 14$ and $m_2 = 15$, thus the p^{th} pattern of the 1st hidden unit of activation, $o_p(1)$ using equation (3.2), (3.3), (3.4) is,

$$o_1(1) = \frac{ns(m_2) - n_1(1)}{ns(m_2) - ns(m_1)} \cdot a(1, m_1) + \frac{n_p(1) - ns(m_1)}{ns(m_2) - ns(m_1)} \cdot a(1, m_2) \tag{3.5}$$

which is,

$$o_1(1) = \frac{1.6 - 1.425}{1.6 - 1.2} \cdot 0.768 + \frac{1.425 - 1.2}{1.6 - 1.2} \cdot 0.832 \tag{3.6}$$

Thus

$$o_1(1) = 0.804 \tag{3.7}$$

From the $o_1(1)$ final value, comparing with figure 3.1, that it lies between $a(1, m_1)$ and $a(1, m_2)$, which proves the linear interpolation concept.

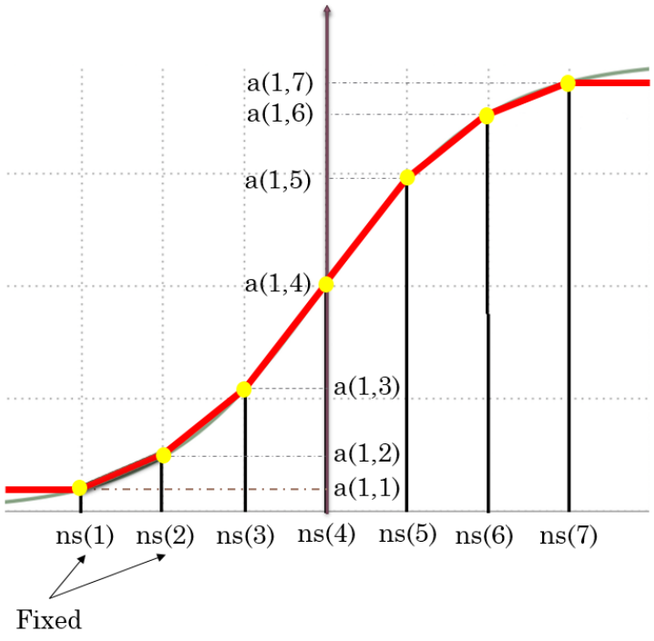
Also, for the p[th] pattern of k[th] hidden unit of the net vector $n_p(k)$ greater than +4 the $o_p(k)$ for the p[th] pattern of k[th] hidden unit is the piecewise linear activation value for n = +4. And for the p[th] pattern of k[th] hidden unit of the net vector $n_p(k)$ less than -4 the $o_p(k)$ for the p[th] pattern of k[th] hidden unit is the piecewise linear activation value for n = -4.

Which can also be written as,

$$o_p(k) = \begin{cases} a(k, \max(sample)) & for\ p > s \\ w_{1p}(k) \cdot a(k, m_1) + w_{2p}(k) \cdot a(k, m_2) & for\ s > p > r \\ a(k, \min(sample)) & for\ p < r \end{cases} \tag{3.8}$$

where r is the lower range i.e. -4 and s is the higher range i.e. +4 picked while initializing the random piecewise linear samples, p is the pattern and k is the hidden unit.

Piecewise linear activations



3. 3 Fixed piecewise linear activations

The above plot is the plot for 7 fixed piecewise linear activations for net versus activations values where all the points between the 2 fixed samples are made linear using the linear interpolation method.

3.3 Activations Training using steepest descent gradient method

Here the steepest descent gradient [40] method is similar to the back propagation used in chapter 2 for calculating a gradient. Using equations (2.6), (2.4), (2.1) and (3.2), (3.3), (3.4) the negative gradient of E with respect to the piecewise linear activations is,

$$g_o(k,m) = \frac{-\partial E}{\partial a(k,m)} \tag{3.9}$$

which is,

$$g_o(k,m) = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left(t_p(i) - y_p(i)\right) \cdot \frac{\partial y_p(i)}{\partial a(u,m)} \tag{3.10}$$

$$\frac{\partial y_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot \frac{\partial o_p(i)}{\partial a(u,m)} \tag{3.11}$$

$$\frac{\partial o_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot \begin{pmatrix} \left(\delta(m - m_1) \cdot w_1(p,u)\right) \\ +\left(\delta(m - m_2) \cdot w_2(p,u)\right) \end{pmatrix} \tag{3.12}$$

where, for the p[th] pattern and k[th] hidden unit of the net value the m_1 and m_2 sample positions are found, where the p[th] pattern of k[th] hidden unit of the net value lies between the two fixed piecewise linear sample values m_1 and m_2 of the u[th] hidden unit as described in the search algorithm. Also the $w_1(p,u)$ and $w_2(p,u)$ from equations (3.3) and (3.4) are also found. As search algorithm is used the correct *m* sample for a particular pattern's hidden unit is found. The equation (3.12) solves for the p[th] patterns u[th] hidden unit of the piecewise linear activations and accumulates the gradient for all the p[th] patterns of their respective u[th] hidden units Optimal

Learning factor for activations training(OLF-A)

Using the gradient $\boldsymbol{G_o}$, the optimal learning factor for activations training is calculated as,

The activation function vector $\boldsymbol{o_p}$ can be related to its gradient as,

$$o_p(k) = w_1(p,k) \cdot [a(k,m_1) + z \cdot g_o(k,m_1)] + w_2(p,k) \cdot [a(k,m_2) + z \cdot g_o(k,m_2)] \tag{3.13}$$

The first partial derivative of E with respect to z is

$$\frac{\partial E}{\partial z} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left(t_p(i) - y_p(i) \right) \cdot \frac{\partial y_p(i)}{\partial z} \tag{3.14}$$

where,

$$\frac{\partial y_p(i)}{\partial z} = \sum_{k=1}^{N_h} w_{oh}(i,k) \cdot \left(\left(w_1(p,k) \cdot g_o(k,m_1) \right) + \left(w_2(p,k) \cdot g_o(k,m_2) \right) \right) \tag{3.15}$$

where, m_1 and m_2 for the p[th] pattern and k[th] hidden unit of net vector $n_p(k)$ is again found, and find $g_o(k,m_1)$ and $g_o(k,m_2)$ from the gradient calculated from equations (3.10), (3.11) and (3.12).

also the Gauss-Newton approximation of the second partial is,

$$\frac{\partial^2 E(z)}{\partial z^2} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[\frac{\partial y_p(i)}{\partial z} \right]^2 \tag{3.16}$$

Thus the learning factor is calculated as

$$z = \frac{-\frac{\partial E}{\partial z}}{\frac{\partial^2 E}{\partial z^2}} \tag{3.17}$$

After finding the optimal learning factor the piecewise linear activations, **A**, are updated in a given iteration as

$$\boldsymbol{A = A + z \cdot G_0} \tag{3.18}$$

where $\boldsymbol{z}$ is a scalar optimal learning factor and $\boldsymbol{G}_0$ is the gradient matrix calculated in equation (3.10), (3.11) and (3.12).

Gradient descent Algorithm for activations training

- Initialize $\boldsymbol{W}, \boldsymbol{W_{oi}}, \boldsymbol{W_{oh}}$ and calculate the random piecewise linear samples and its activations.
- Calculate activation vector $\boldsymbol{o}_p(k)$ using equation (3.2).
- Solve for linear equations using OLS and update the output weights.
- **for** it = Number of iterations, solve for
- Find gradient $\boldsymbol{G_o}$ from equations (3.9), (3.10), (3.11), (3.12).
- Calculate scalar z from equation (3.17) and update $\boldsymbol{A}$ matrix as in equation (3.18).
- Calculate mean square error from equation (2.6) and check for the networks improvement.
- **End for**

3.4 Advanced learning algorithm (MOLF-A)

Multiple Optimal Learning Factor (MOLF) [37, 38] is a second order training algorithm, where a different learning factor $\mathbf{z}_k$ is used to update weights feeding into the kth hidden unit. The input weight connecting the nth input to the kth hidden unit is updated using

$$a(k,m) = w(k,m) + z_k \cdot g_o(k,m) \tag{3.19}$$

where, z_k denotes the learning factor that corresponds to the kth hidden unit.. The $g_{molf-A}(j)$ and Hessian matrix $h_{molf-A}(l,j)$ can be calculated as,

the activation function vector O_p can be related to its gradient as,

$$o_p(k) = w_1(p,k) \cdot [a(k,m_1) + z_k \cdot g_o(k,m_1)] + w_2(p,k) \cdot [a(k,m_2) + z_k \cdot g_o(k,m_2)] \tag{3.20}$$

so $\boldsymbol{g_{molf-A}(u)}$ is given by,

$$\frac{-\partial E}{\partial z_k} = g_{molf-A}(u) = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left(t_p(i) - y_p(i)\right) \cdot \frac{\partial y_p(i)}{\partial z_k} \tag{3.21}$$

where,

$$\frac{\partial y_p(i)}{\partial z_u} = w_{oh}(i,u) . \frac{\partial o_p(u)}{\partial z_u} \tag{3.22}$$

$$\frac{\partial o_p(u)}{\partial z_u} = f_1(p,u) = w_1(p,u) \cdot g_o(u,m_1) + w_2(p,u) \cdot g_o(u,m_2) \tag{3.23}$$

therefore,

$$g_{molf-A}(u) = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left(t_p(i) - y_p(i)\right) w_{oh}(i,u) . \begin{pmatrix} w_1(p,u) \cdot g_o(u,m_1) \\ + w_2(p,u) \cdot g_o(u,m_2) \end{pmatrix} \tag{3.24}$$

Now Using the Gauss – Newton updates, the second partial derivate elements of the Hessian $h_{molf-A}(l,j)$ using equations (3.22) and (3.24) are

$$h_{molf-A}(l,j) = \frac{\partial^2 E}{\partial z_l z_k} = \frac{2}{N_v} \sum_{i=1}^{M} w_{oh}(i,l) \cdot w_{oh}(i,j) \sum_{p=1}^{N_v} f_1(p,l) \cdot f_1(p,j) \tag{3.25}$$

The Gauss-Newton update guarantees that $\boldsymbol{H_{molf-A}}$ is non-negative definite. The vector $\boldsymbol{z}_k$ can be found using OLS algorithm using the following relation

$$\boldsymbol{H_{molf-A}} \cdot \boldsymbol{z} = \boldsymbol{g_{molf-A}} \tag{3.26}$$

Thus the multiple optimal learning factor which is a vector of size (N_h) is found from the negative gradient vector and the hessian matrix from the equation (2.33) using OLS algorithm, where $\boldsymbol{H_{molf-A}}$ is matrix of size $(N_h \times N_h)$ and $\boldsymbol{g_{molf-A}}$ is (N_h) vector.

MOLF Algorithm for activations training

- Initialize $\boldsymbol{W}, \boldsymbol{W_{oi}}, \boldsymbol{W_{oh}}$ and calculate the random piecewise linear samples and its activations.
- Calculate activation vector $\boldsymbol{o}_p(k)$ using equation (3.2).
- Solve for linear equations using OLS and update the output weights.
- **for** it = Number of iterations, solve for
- Find gradient $\boldsymbol{G_o}$ from equations (3.9), (3.10), (3.11), (3.12).
- Calculate multiple optimal learning factor (MOLF) $\mathbf{z}_k$ by solving linear equation (3.26) using OLS and update $\boldsymbol{A}$ matrix.
- Calculate mean square error from equation (2.6) and check for the networks improvement.
- **End for**

3.5 Final Piecewise Linear Training Algorithm Descriptions

In this section, final piecewise linear training algorithms are discussed. 2 piecewise linear activations training algorithms are described in the subsection. The Fixed piecewise linear activation training is the section (3.4.1) which uses the OWO method for output weights training, and HWO-MOLF for input weights training also, in this training algorithm the piecewise linear samples and activations both are fixed for all the iterations. Similarly in the piecewise linear activations training used in section (3.4.2) uses the same method for output weights training and the input weights training, but in this algorithm, an extra training algorithm for training the piecewise linear activations keeping piecewise linear samples constant for all the iterations is discussed, and the activations are trained using the gradient descent and MOLF algorithm. This algorithm can be learned two ways, the comparison of both the algorithm and the result for the best network is given in Chapter 5.A detailed algorithm description is as follows,

3.5.1 Fixed piecewise linear activations training algorithm description (HWO-MOLF - FPWLT)

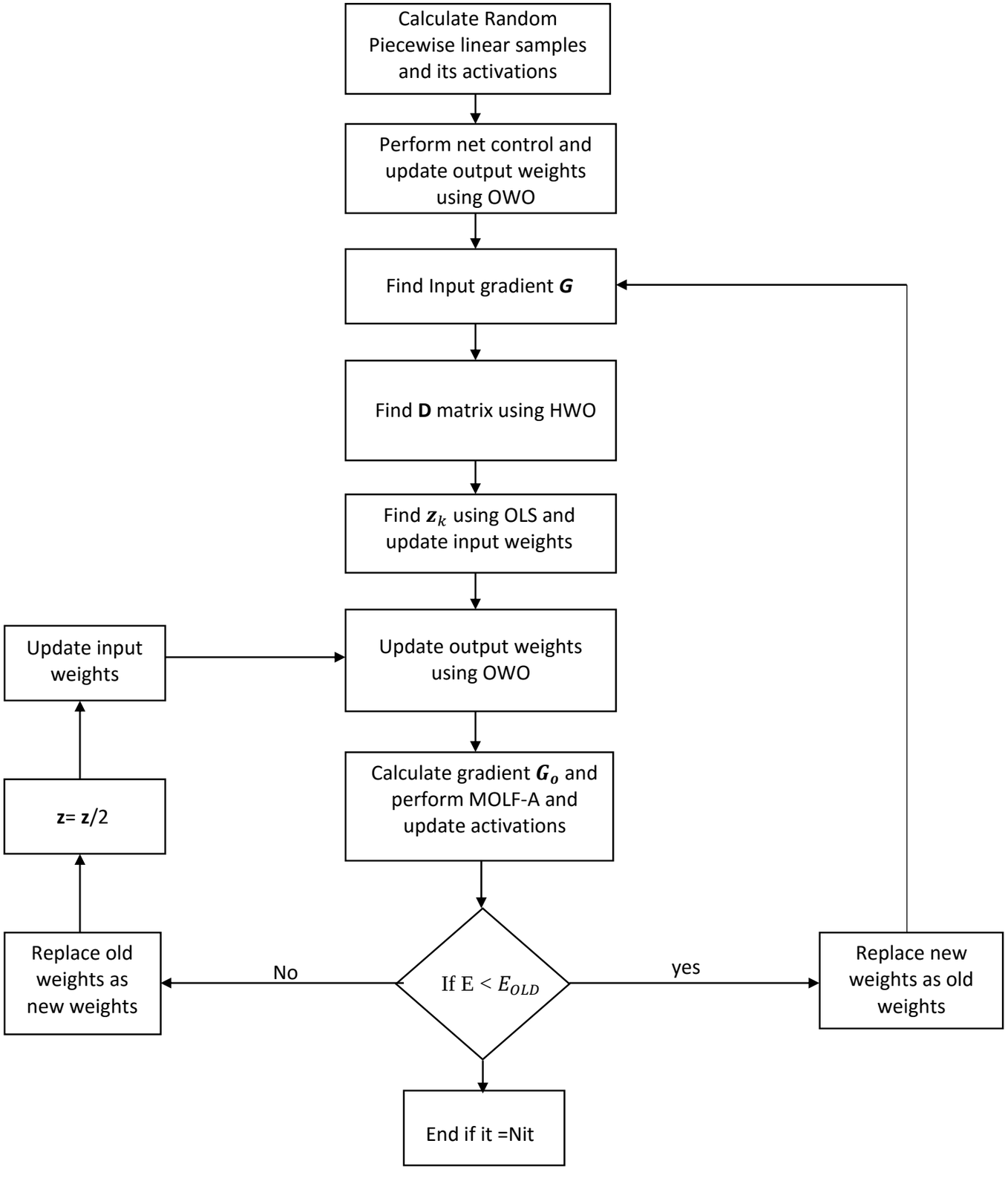
Using the OWO for output weights, A-MOLF for activation training, HWO-MOLF for input weights, the following piecewise linear activations training algorithm is constructed.

1. Calculate fixed random piecewise linear samples and its activations as shown in figure 3.1 and 3.2 and equation (3.2).
2. Perform net control from section (2.2.1) and update the output weights using output weight optimization method from Section (2.2.2).
3. Start of the iterations, increment i_t by 1.
4. Perform backpropagation from section (2.3) and calculate input gradient using equation (2.17).
5. Solve the linear equations of (2.23) to find D matrix using Hidden weight optimization method and update the G matrix as the D matrix.
6. Calculate $\boldsymbol{z}_k$ using the OLS algorithm, and update the inputs weights using equation (2.32).
7. Again update the output weights using output weight optimization method from section (2.2.2).
8. Perform Backtracking.
9. Go to step 4 and Continue until $i_t = N_{it}$

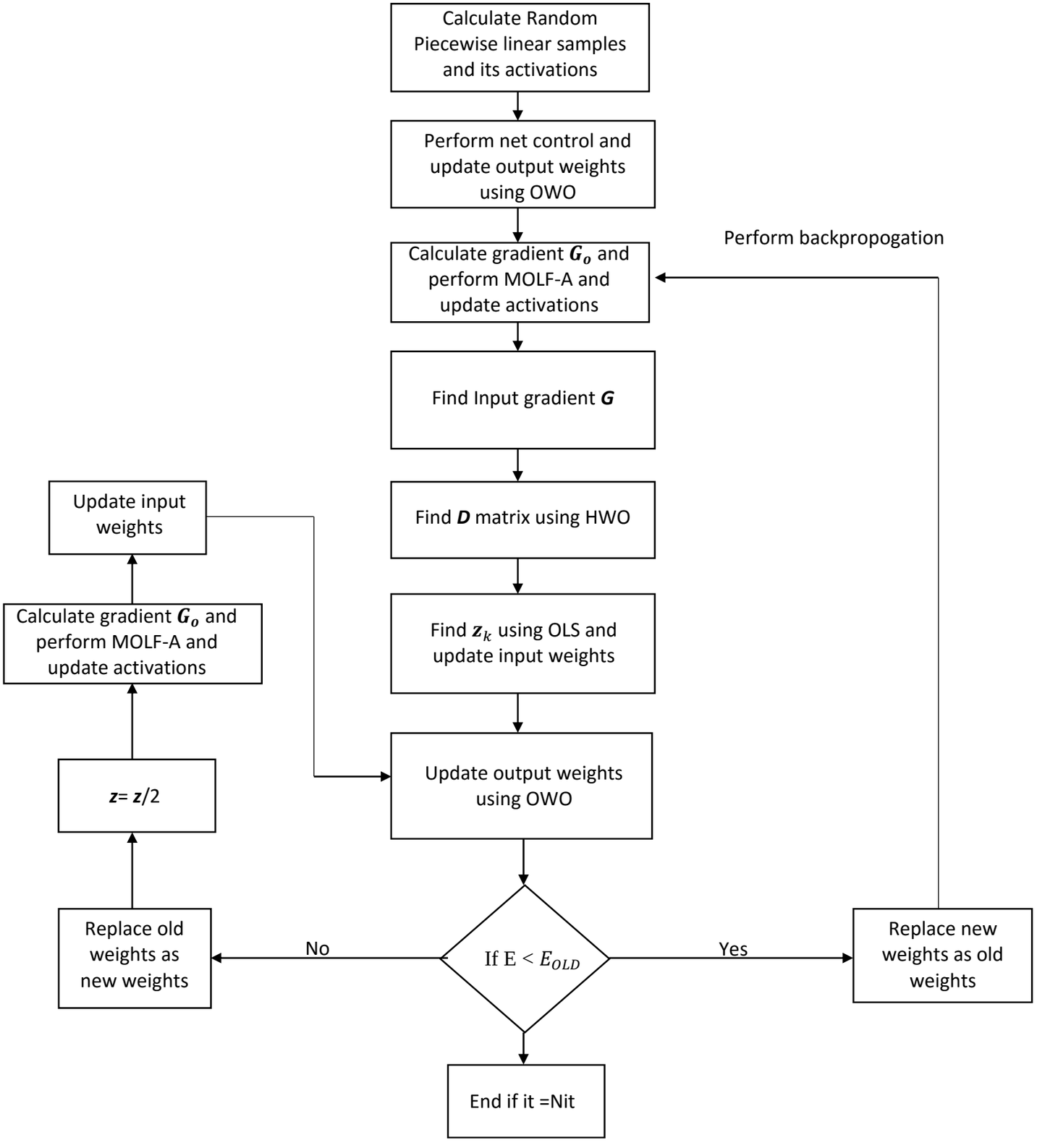
3.5.1 Piecewise linear activations training algorithm description (PWAT)

The piecewise linear activation algorithm can be learned two ways, can be called as PWAT version A and PWAT version B, the block diagram for each algorithm is given as Follows

PWAT version A block diagram



PWAT version B Algorithm block diagram

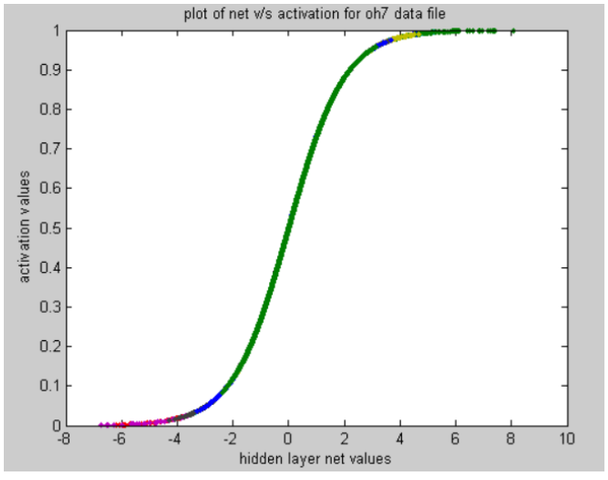


3.6 Activation function comparison

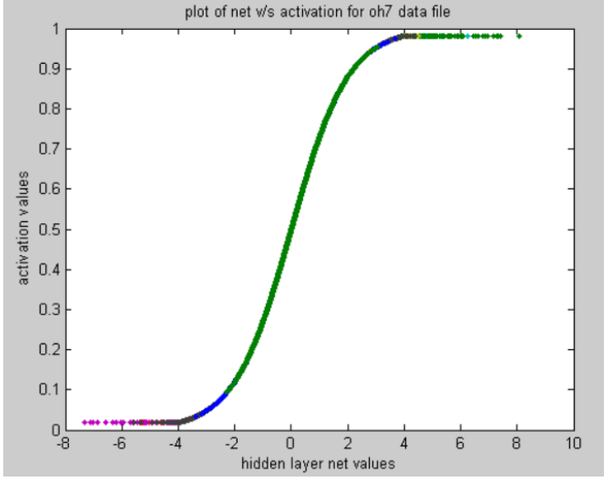
In this section, comparison of the sigmoid activations with the fixed piecewise linear activations and the activations after training are shown.

The activations are compared using the plot for the net values versus the activations.

Training data file used - oh7 , inputs =20, outputs = 3, patterns = 10453, number of hidden units = 30



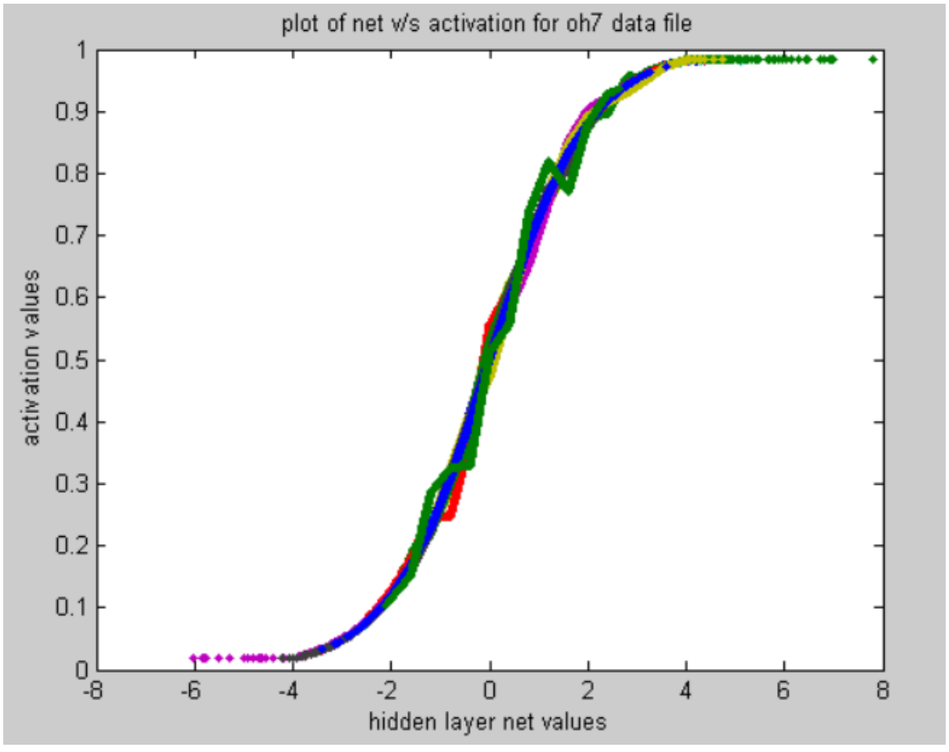
3.2 Sigmoid plot for all hidden units for oh7 data file



3.3 FPWL activations plot for all hidden units for oh7 data file

Figure 3.2 is the plot of the hidden layer net values v/s the activation values of the sigmoidal activation function for all 30 hidden units combined, where the values are between 0 and 1, which is the property of a sigmoid. Whereas figure 3.2 is the plot of hidden layer values v/s the activation values for a fixed piecewise linear activation for all 30 hidden units combined, which are defined in section 3.2, where the

values of net vector $\boldsymbol{n_p}$ that are beyond 4 and less than -4 have sigmoid activation values of +4 and -4 respectively.



3.4 Piecewise linear activation plot after training for all hidden units for oh7 data file

Figure 3.2 is the plot of hidden layer net values versus the activation values of the piecewise linear activations for all 30 hidden units combined after training for one iteration. Where each hidden unit are trained individually and some of the hidden units have different heights, which are the activation values.

Chapter 4

EXPERIMENTS AND RESULTS

In this chapter, training experiments and results for comparison between the regular HWO-MOLF algorithm and the piecewise linear activation algorithm described in chapter 2 and chapter 3 are presented. The algorithm was implemented in Matlab R2012b version. In section 4.1 comparison of the 2 piecewise linear activations training algorithms described in section 3.5 which uses both the OLF and MOLF-A training method individually for training the piecewise linear activations are done. In section 4.2 comparison of different training algorithm for various data files described in chapter 2 and chapter 3 are presented with final results.

The training files used for comparison in this chapter are

- Oh7.tra – Inputs = 20, Outputs = 3, Number of patterns = 10453
- 2spirals.txt – Inputs = 2, Outputs = 1, Number of patterns = 10000
- Inverse9_9.txt – Inputs = 9, Outputs = 9, Number of patterns = 10000
- Twod.tra – Input = 8, Output = 7, Number of patterns = 1768

The notation for different variables in the configuration is :

- Nh = number of hidden units.
- Nit = number of iterations.

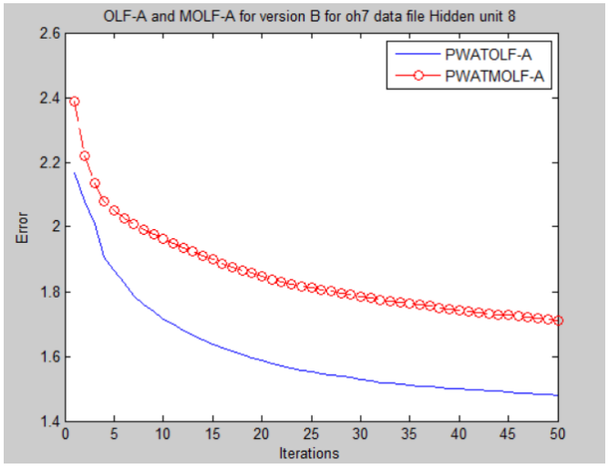
4.1 Comparison between PWAT version A and PWAT version B.

In this section, the goal is to compare and find the PWAT algorithm with the best result and use it for the comparison in section 4.2. In PWAT version A algorithm, for every iteration the algorithm trains the

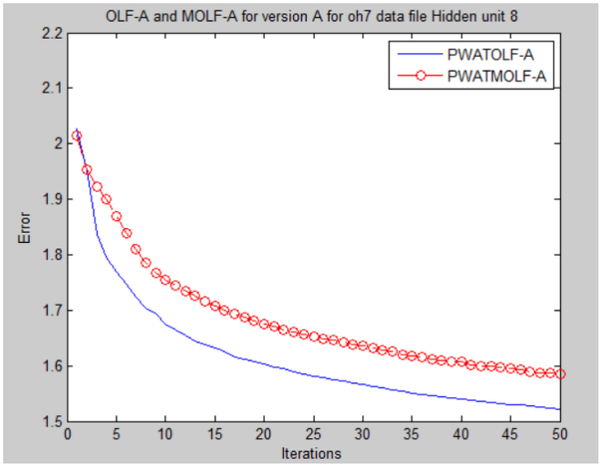
network with activations training first, then input weights followed by output weights, Similarly in PWAT version B, for every iteration this algorithm trains the network with the input weights first, then output weights followed by activations training.

The comparison of the PWAT algorithm is performed using different training data files and for various number of hidden units. Throughout the comparison the left half is the version B training algorithm and the right half is the version A training algorithm. Both OLF-A and MOLF-A learning methods for each PWAT algorithms are also used for comparison. The details can be understood from the legend used on the plots.

Experiments for oh7 data file with inputs =20, outputs = 3, patterns = 10453, Nit =50

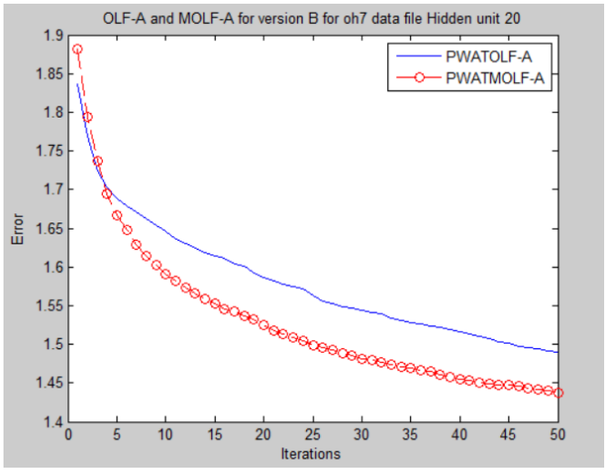


4.1 PWAT version B plot for Nh =8 and oh7 data file

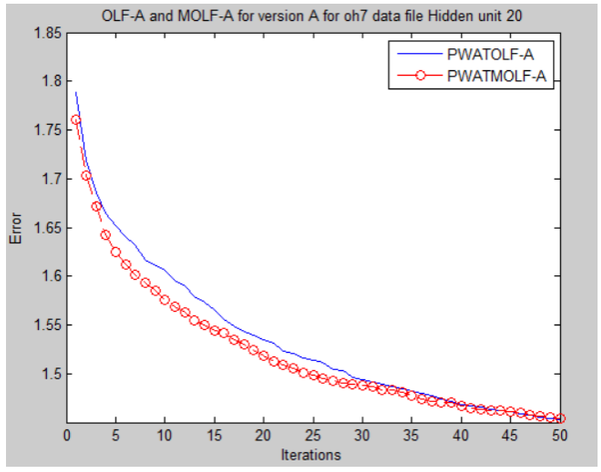


4.2 PWAT version A plot for Nh =20 and oh7 data file

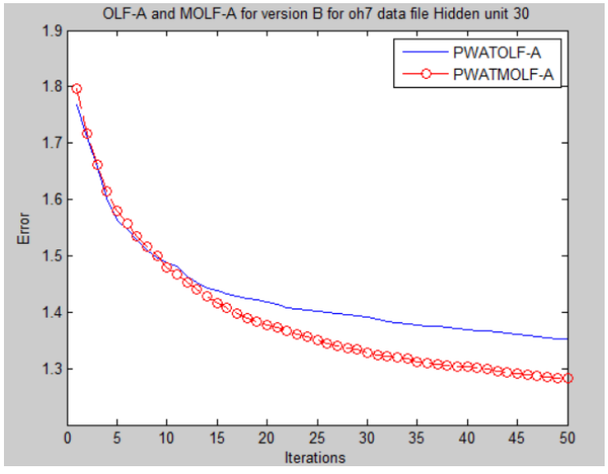
The left plot is the PWAT version B for 8 hidden units and the right plot is the PWAT version A for 8 hidden units. From these plots it can observe that for lower number of hidden units such as '8', both versions PWAT using OLF-A training method performs better than MOLF-A training method with monotonically decrease in error.



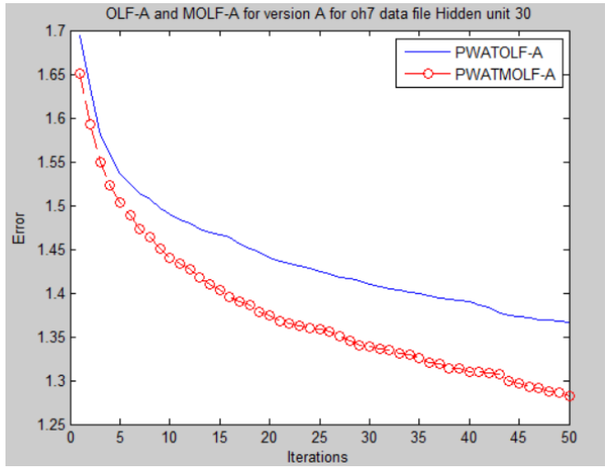
4. 1 PWAT version B plot for Nh-20 and oh7data file



4. 2 PWAT version A plot for Nh-20 and oh7data file

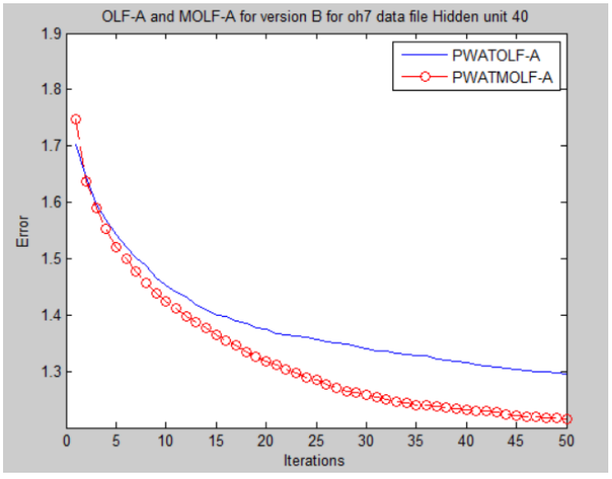


4. 3 PWAT version B plot for Nh-30 and oh7data file

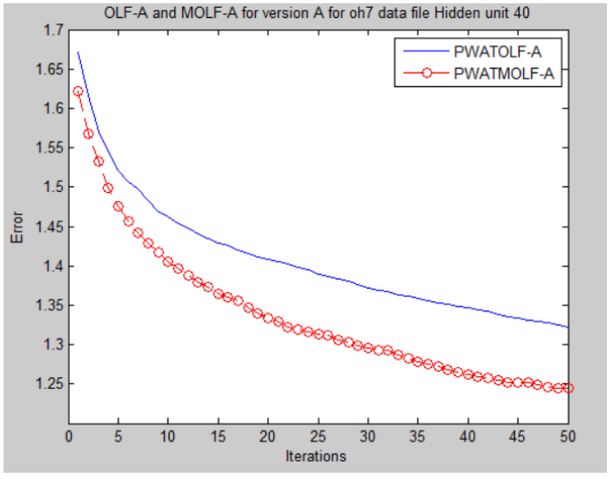


4. 4 PWAT version A plot for Nh-30 and oh7data file

From the figure 4.3, 4.4, 4.5, 4.6, it can be observed that as the number of hidden units are increased the PWAT algorithms with MOLF-A training method performs better than the OLF-A training method.



4. 5 PWAT version B plot for Nh-40 and oh7data file



4. 6 PWAT version A plot for Nh-40 and oh7data file

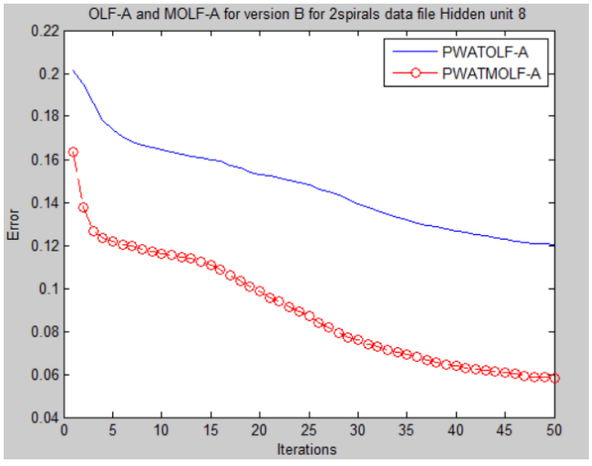
No of Hidden Units / Algorithm	8	20	30	40
PWAT-OLF Error version B	1.4803	1.4901	1.3518	1.2954
PWAT-OLF Error version A	1.5222	1.4530	1.3665	1.3227
PWAT-MOLF Error version B	1.7110	1.4376	1.2827	1.2151
PWAT-MOLF Error Version A	1.5848	1.4539	1.2827	1.2443

Table 4.1 - Error Analysis for PWAT training methods for oh7 training data file.

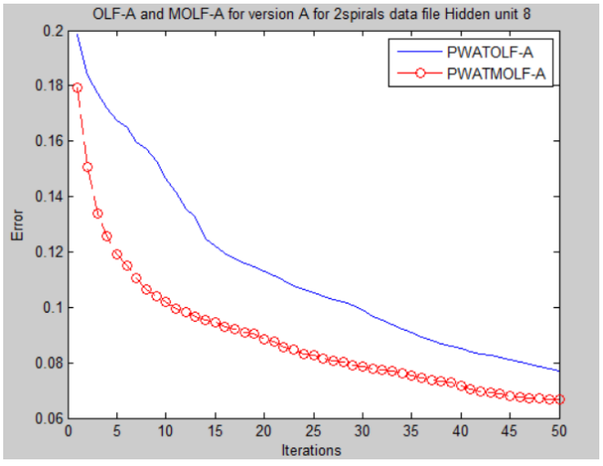
Table 4.1 is the summary of all the Error versus iterations for the PWAT algorithms with OLF-A and MOLF-A training methods for oh7 data file. From the table it can observe that the error monotonically

decreases at every iteration as backtracking method for every iteration is used, Also the PWAT training algorithms with MOLF-A training method shows minimal error after 50 iterations except for the PWAT version B algorithm for Nh=8, were the error is much higher than the other algorithms. From the plot it can also determine that the PWAT version B algorithm produces better results except for hidden unit 8.

Experiments for 2spirals data file with inputs =2, outputs = 1, patterns = 10000, Nit =50

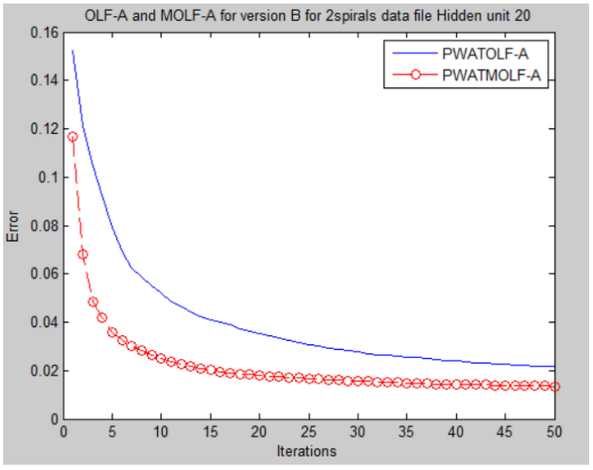


4. 7 PWAT version B plot for Nh-8 and 2spirals data file

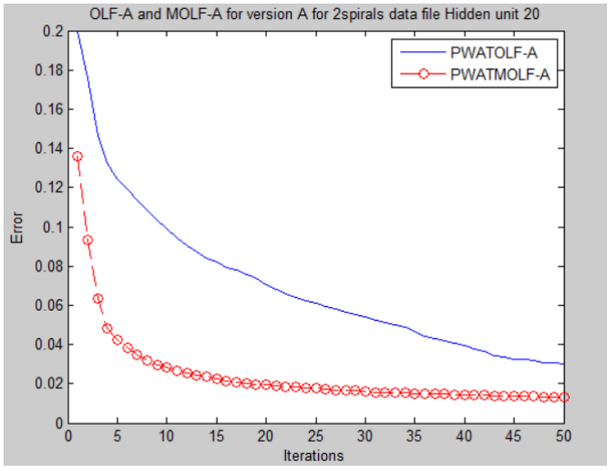


4. 8 PWAT version A plot for Nh-8 and 2spirals data file

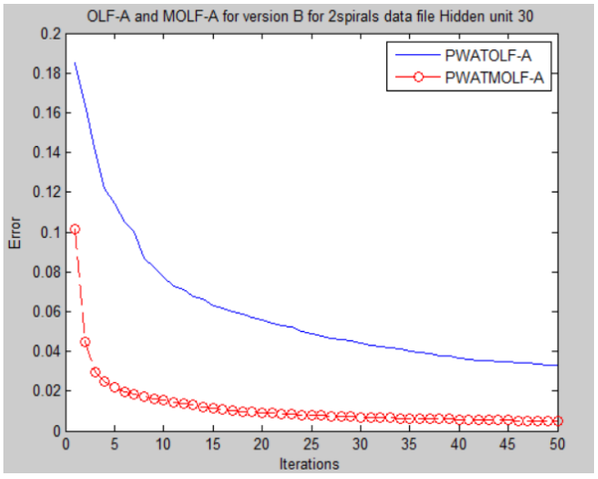
The left plot is the PWAT version B for 8 hidden units and the right plot is the PWAT version A for 8 hidden units. Comparing the plot for 8 hidden unit of 2 spirals data file with oh7 data file, the lower number of hidden units the PWAT algorithm with MOLF-A training method for 2spirals data file performs better than OLF-A algorithm with decrease in error at every iteration, Also at iteration 50 it can be also observed that the error remains almost constant than the previous error because the current gradient values are almost similar to the previous gradient values and the error is about to reach optimal minimum.



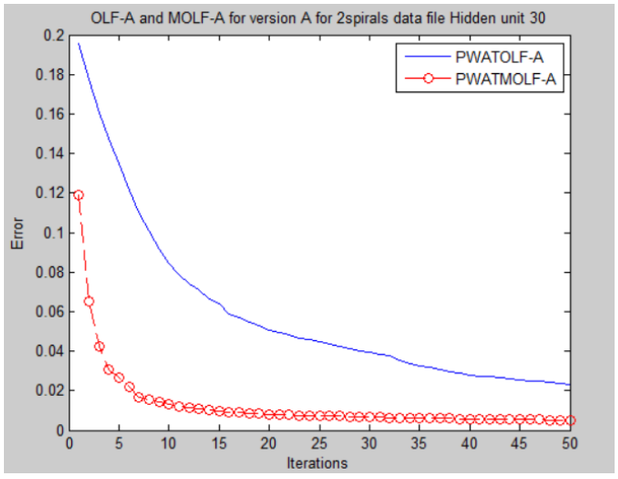
4. 9 PWAT version B plot for Nh20 and 2spirals data file



4. 10 PWAT version A plot for Nh-20 and 2spirals data file

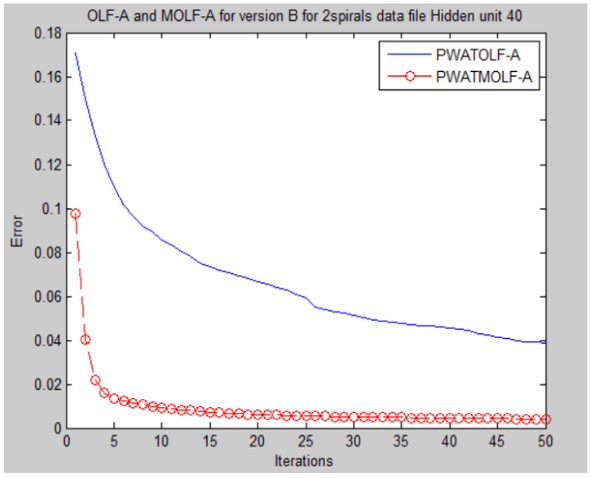


4. 11 PWAT version B plot for Nh-30 and 2spirals data file

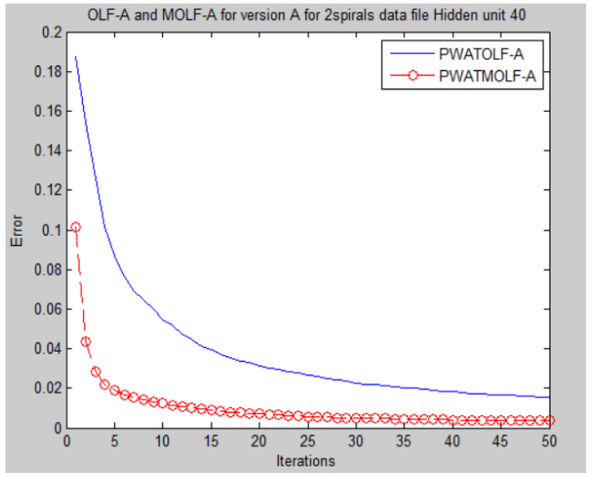


4. 12 PWAT version A plot for Nh-30 and 2spirals data file

For higher number of hidden units the results are similar to the oh7 data file with similar number of hidden units where the PWAT algorithms with MOLF-A training method shows better results. Also from the plot it can observe that the error values for these higher number of hidden units remain constant after a particular hidden unit, because the error is about to reach optimal minimum.



4. 13 PWAT version B plot for Nh-40 and 2spirals data file



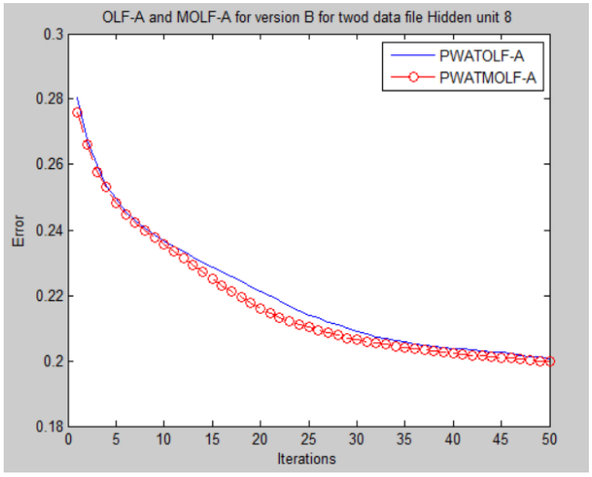
4. 14 PWAT version A plot for Nh-40 and 2spirals data file

No of Hidden Units / Algorithm	8	20	30	40
PWAT-OLF Error version B	0.1203	0.0216	0.0329	0.0388
PWAT-OLF Error version A	0.0770	0.0302	0.0229	0.0155
PWAT-MOLF Error version B	0.0583	0.0135	0.0050	0.0041
PWAT-MOLF Error Version A	0.0666	0.0132	0.0051	0.0035

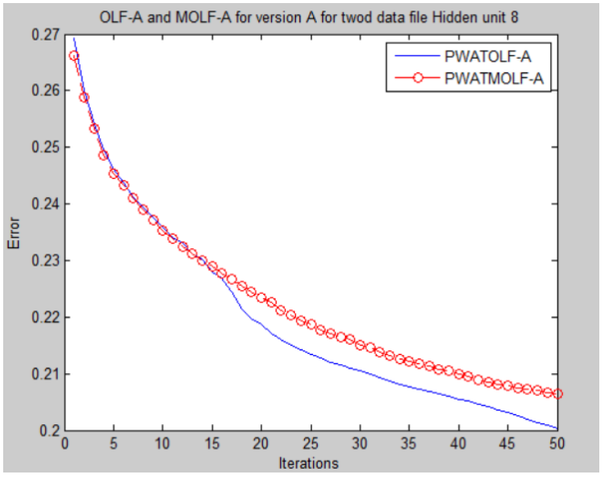
4.2 - Error Analysis for PWAT training methods for 2spirals training data file.

Table 4.1 is the summary of all the Error v/s iterations for the PWAT algorithms with OLF-A and MOLF-A training methods for 2spirals data file. From the table the PWAT algorithms with MOLF-A training method produces better results for every hidden unit can be observed. Also both the PWAT training algorithms perform equally.

Experiments for twod data file with inputs =8, outputs = 7, patterns = 1768, Nit =50

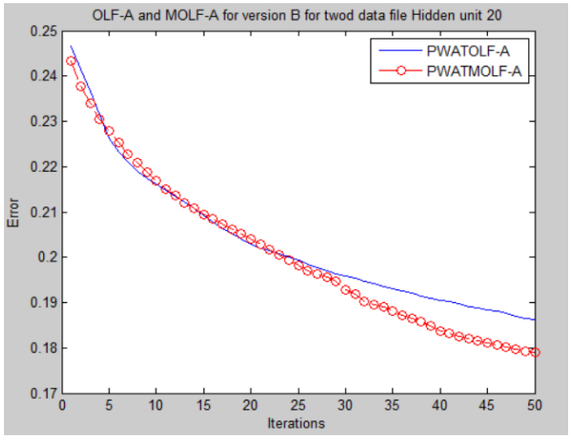


4. 15 PWAT version B plot for Nh-8 and Twod data file

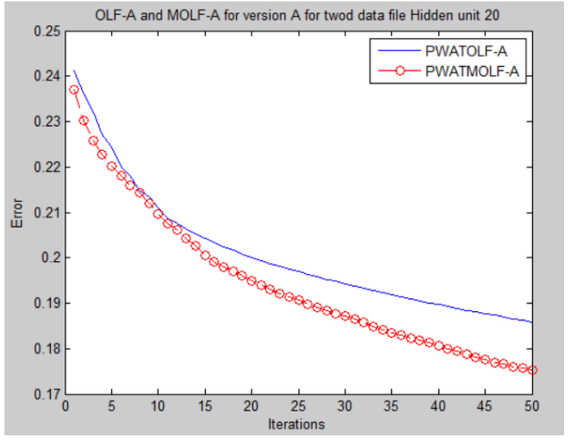


4. 16 PWAT version A plot for Nh-8 and Twod data file

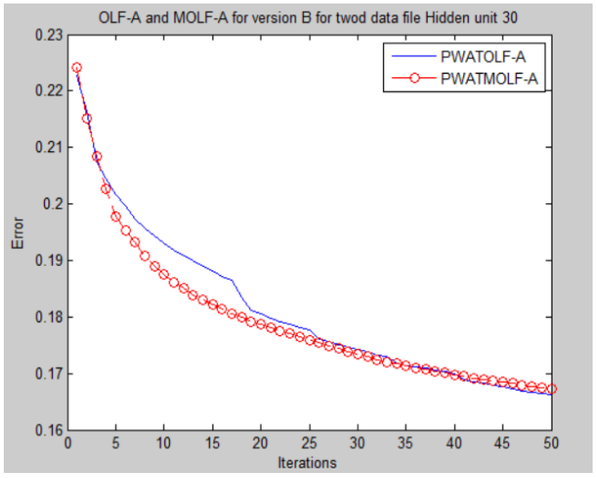
The left plot is the PWAT version B for 8 hidden units and the right plot is the PWAT version A for 8 hidden units. Comparing the plot, the results are found to be similar to the oh7 data file for 8 hidden units, where PWAT algorithms with OLF-A shows better results.



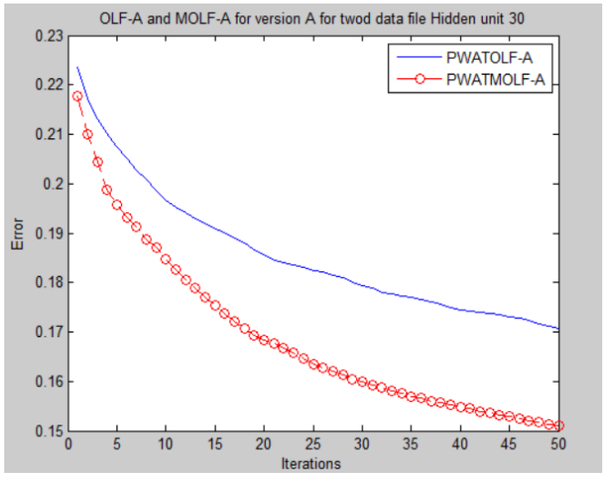
4. 17 PWAT version B plot for Nh-20 and Twod data file



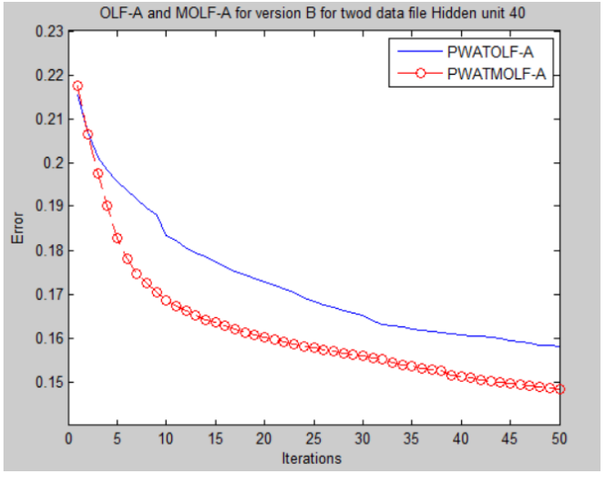
4. 18 PWAT version A plot for Nh-20 and Twod data file



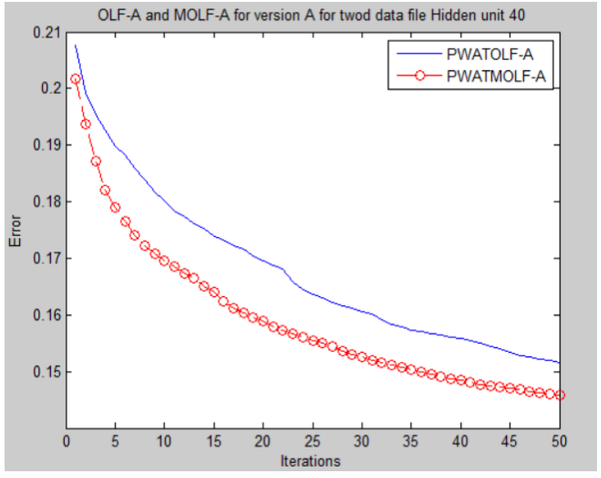
4. 19 PWAT version B plot for Nh-30 and Twod data file



4. 20PWAT version A plot for Nh-30 and Twod data file



4. 21 PWAT version B plot for Nh-40 and Twod data file



4. 22PWAT version A plot for Nh-40 and Twod data file

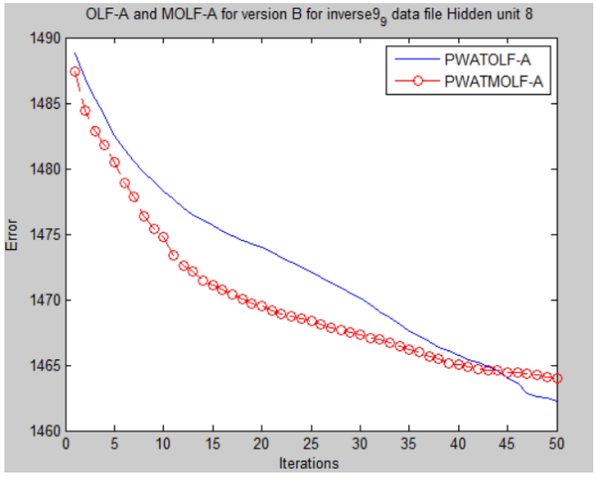
From all the above plots, as the number of hidden units increase MOLF-A training method improves the training error, which shows exactly similar results as of the oh7 data file.

No of Hidden Units / Algorithm	8	20	30	40
PWAT-OLF Error version B	0.2006	0.1863	0.1662	0.1579
PWAT-OLF Error version A	0.2002	0.1859	0.1708	0.1515
PWAT-MOLF Error version B	0.1998	0.1790	0.1674	0.1483
PWAT-MOLF Error Version A	0.2064	0.1753	0.1510	0.1457

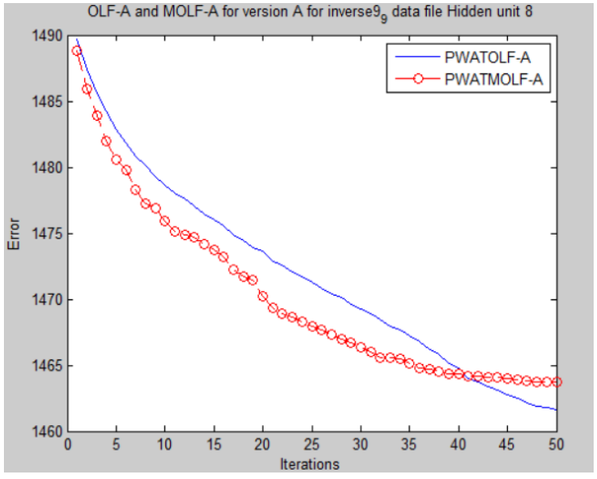
4.3 - Error Analysis for PWAT training methods for twod training data file.

Table 4.1 is the summary of all the Error v/s iterations for the PWAT algorithms with OLF-A and MOLF-A training methods for twod data file. From the table the PWAT algorithms with MOLF-A training method produces better results for every hidden unit can be observed. Also PWAT version A shows significant decrease in error as the number of hidden units increases.

Experiments for inverse9 data file with inputs = 9, outputs = 9, patterns = 10000, iterations =50

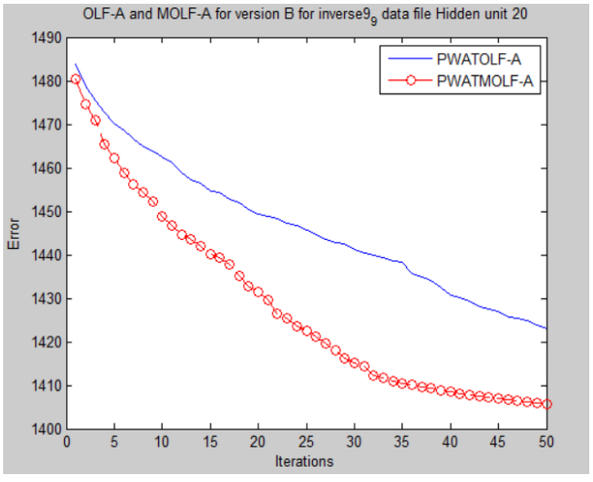


4. 23 PWAT version B plot for Nh-8 and inverse9 data file

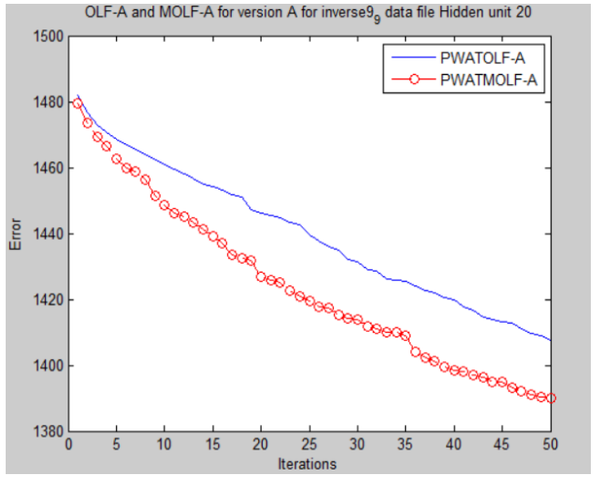


4. 24PWAT version A plot for Nh-8 and inverse9 data file

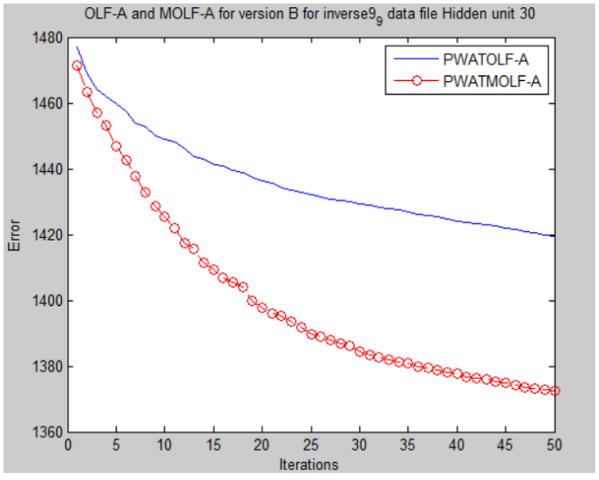
The left plot is the PWAT version B for 8 hidden units and the right plot is the PWAT version A for 8 hidden units. From the starting error for inverse9 data file in the plot it can observe that this file is difficult for training. For 8 hidden units it can observe that MOLF-A training method performs better with gradually decrease in error with the OLF-A training method, but after 50 iterations the OLF-A training method decreases the error while the MOLF-A method approaches optimal minimum.



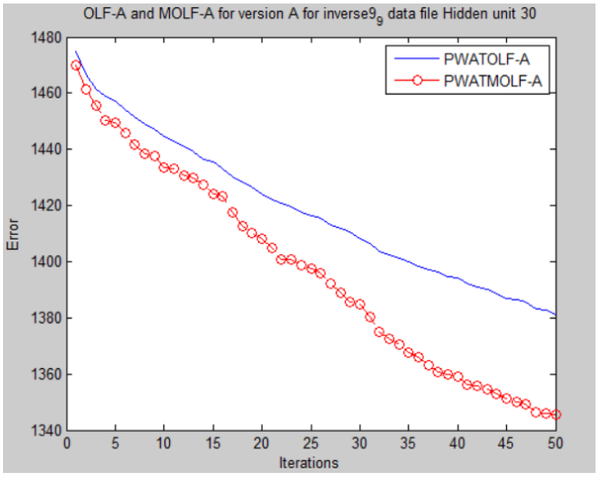
4. 25 PWAT version B plot for Nh-20 and inverse9 data file



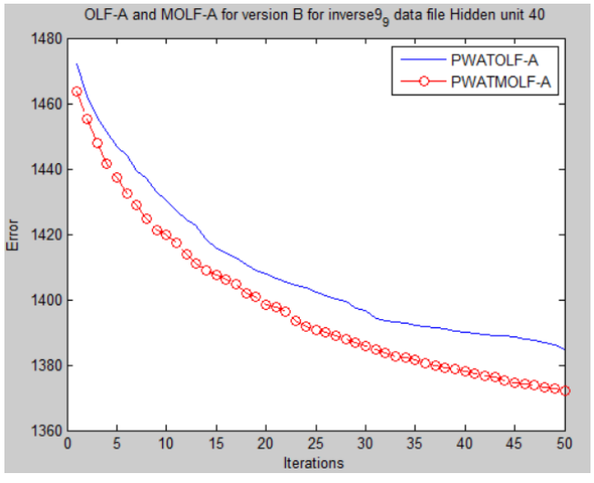
4. 26PWAT version A plot for Nh-20 and inverse9 data file



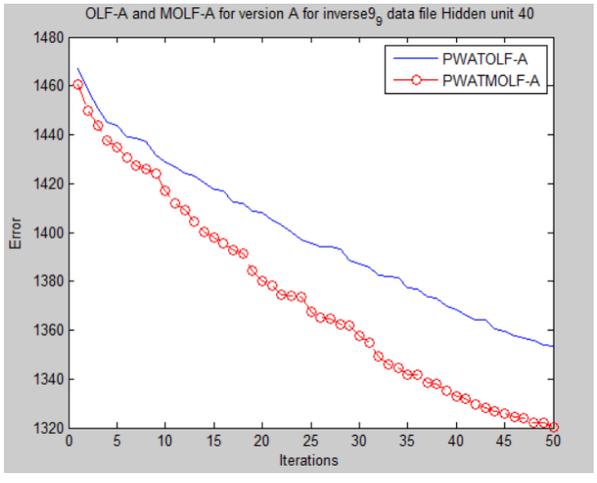
4. 27 PWAT version B plot for Nh-30 and inverse9 data file



4. 28 PWAT version A plot for Nh-30 and inverse9 data file



4. 29 PWAT version B plot for Nh-40 and inverse9 data file



4. 30PWAT version A plot for Nh-40 and inverse9 data file

From all the above plots, it can be said that the MOLF-A training method for PWAT algorithms portrays better final Error.

No of Hidden Units / Algorithm	8	20	30	40
PWAT-OLF Error version B	1.46220e+03	1.42305e+03	1.41936e+03	1.38474e+03
PWAT-OLF Error version A	1.46160e+03	1.40762e+03	1.38116e+03	1.35324e+03
PWAT-MOLF Error version B	1.46401e+03	1.40562e+03	1.37256e+03	1.37214e+03
PWAT-MOLF Error Version A	1.46372e+03	1.38995e+03	1.34557e+03	1.32027e+03

4.4 - Error Analysis for PWAT training methods for inverse9_9 training data file.

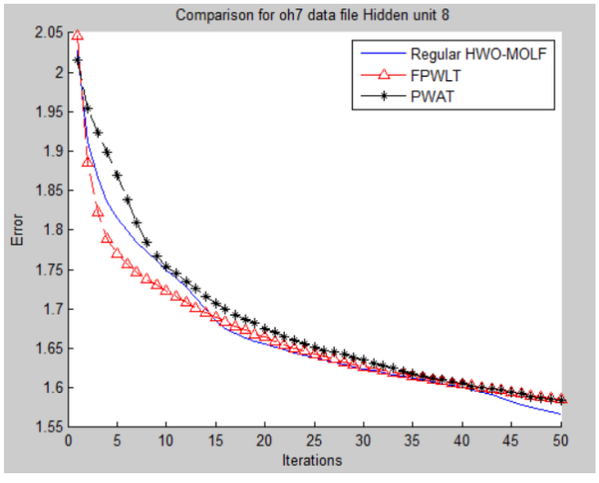
Table 4.1 is the summary of all the Error v/s iterations for the PWAT algorithms with OLF-A and MOLF-A training methods for inverse9_9 data file. Again for this data file the PWAT algorithms with MOLF-A training method produces better results for every hidden unit can be observed and PWAT version A shows significant decrease in error as the number of hidden units increases.

From all the experiments performed in section 4.1, it can concluded that the MOLF-A training method performs better than the OLF-A for both PWAT algorithms, except for lower number of hidden units, also the PWAT algorithms perform equally better, but the PWAT version A gives more promising result. So the final PWAT algorithm for comparison in section 4.2 is the PWAT version A, which can be called as PWAT in the following section.

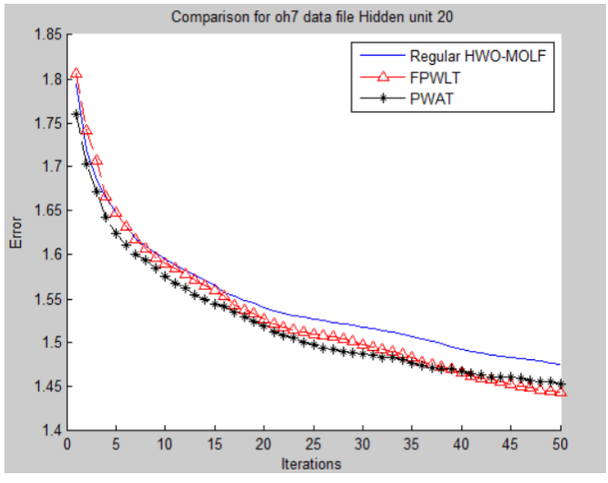
4.2 Regular HWO-MOLF v/s FPWLT v/s PWAT Experiments and results

The goal in this section is to compare the Piecewise linear activation training algorithm with Regular HWO-MOLF using sigmoidal activations. Also the comparison with Fixed piecewise linear activation training is also performed for better clarity. The final results for comparison with each data file are shown in tabular format after the experiments on each data file is done, and Error v/s iterations are plotted for all 3 training algorithms together to support the table.

Experiments for oh7 data file with inputs =20, outputs = 3, patterns = 10453, iterations =50

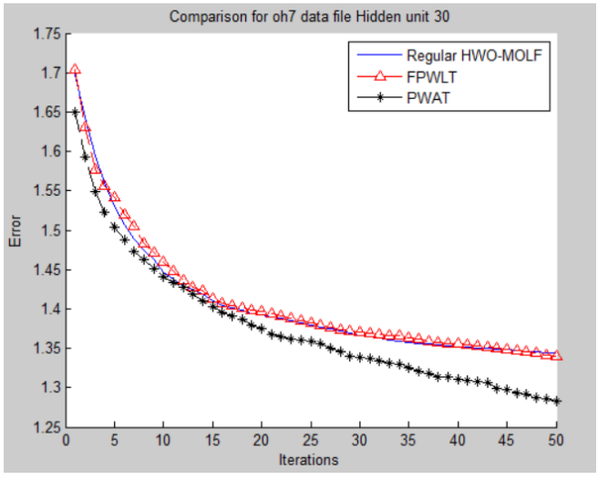


4. 31 Error Comparison for Nh= 8 and oh7 data file

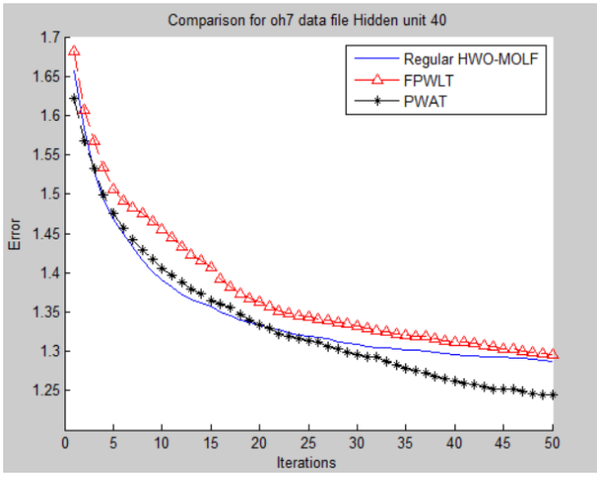


4. 32 Error Comparison for Nh= 20 and oh7 data file

The plot shows the error comparison for all training algorithms for hidden unit-8 and hidden unit 20 in figure 4.33 and 4.34 for 50 iterations and oh7 data file. Where, for lower number of hidden units the regular HWO-MOLF with sigmoid activations shows better results than FPWLT and PWAT, but as the number of hidden unit's increases FPWLT and PWAT displays better performance than the regular HWO-MOLF.



4. 33 Error Comparison for Nh= 30 and oh7 data file



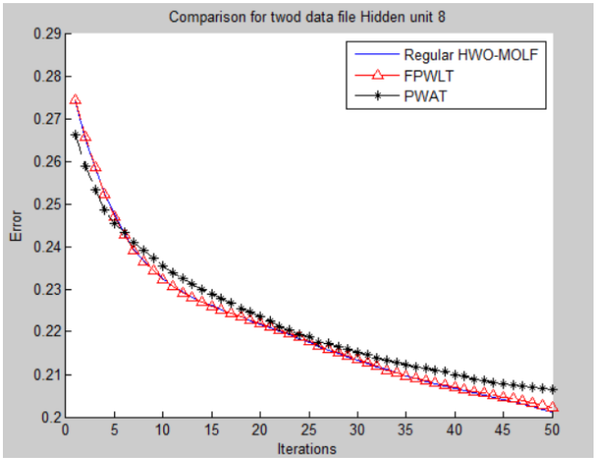
4. 34 Error Comparison for Nh= 40 and oh7 data file

No of Hidden units / Error	8	20	30	40
Regular HWO-MOLF	1.5674	1.4761	1.3443	1.2877
FPWLT	1.5852	1.4437	1.3403	1.2961
PWAT	1.5848	1.4539	1.2827	1.2443

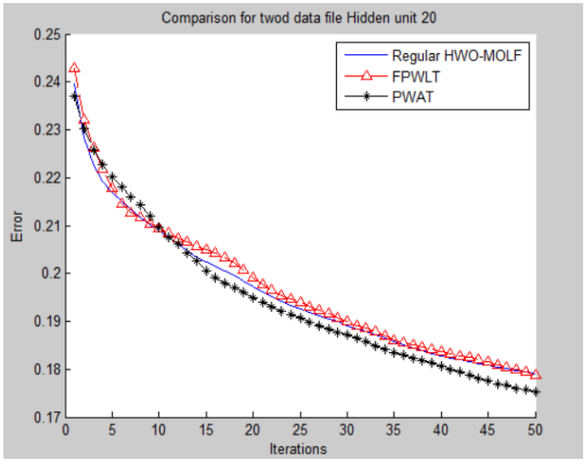
4.5 - Error comparison for Regular HWO-MOLF, FPWLT and PWAT for oh7 data file

Table 4.1 is the summary of the Error v/s iterations comparison for all the training algorithms described for oh7 data file. From the table it can observed that as the number of hidden units increases the performance of PWAT algorithms increase, which means for oh7 data file for larger number of hidden units PWAT algorithm shows better results.

Experiments for twod data file with inputs =8, outputs = 7, patterns = 1768 iterations =50

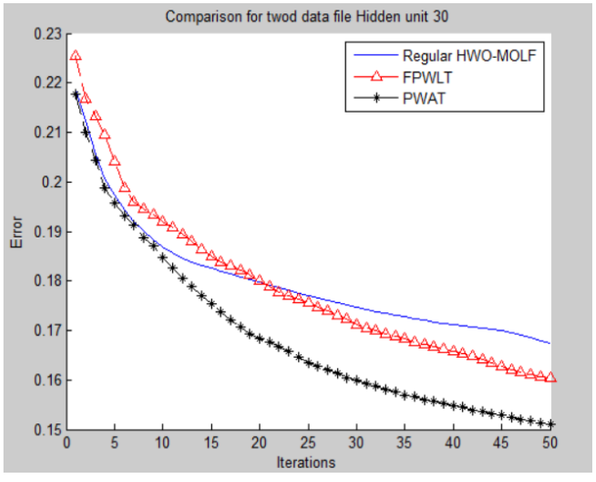


4. 35 Error Comparison for Nh= 8 and Twod data file

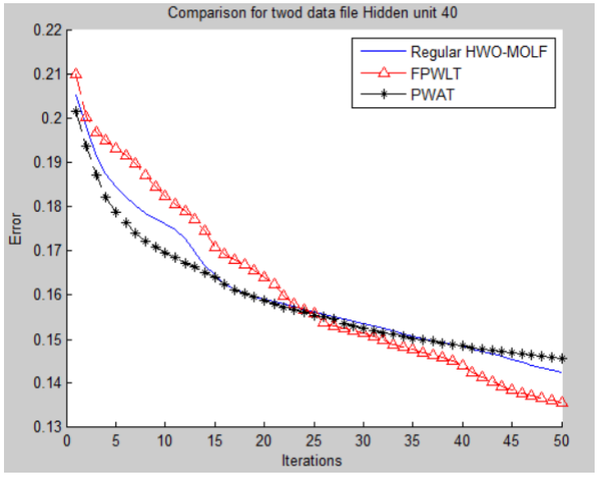


4. 36 Error Comparison for Nh= 20 and Twod data file

Here similar results can be seen for 8 hidden units and 20 hidden units, where for lower number of hidden units the regular HWO-MOLF performs better but as the number of hidden unit's increase the performance of PWAT also increases.



4. 37 Error Comparison for Nh= 30 and Twod data file



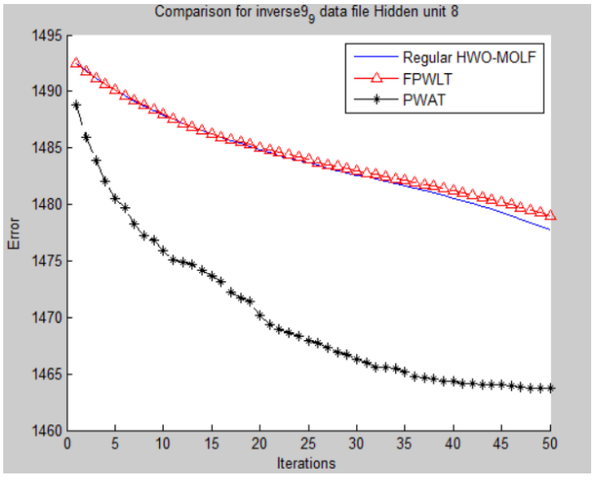
4. 38 Error Comparison for Nh= 40 and Twod data file

No of Hidden units / Error	8	20	30	40
Regular HWO-MOLF	0.2012	0.1791	0.1673	0.1425
FPWLT	0.2023	0.1787	0.1603	0.1358
PWAT	0.2064	0.1753	0.1510	0.1457

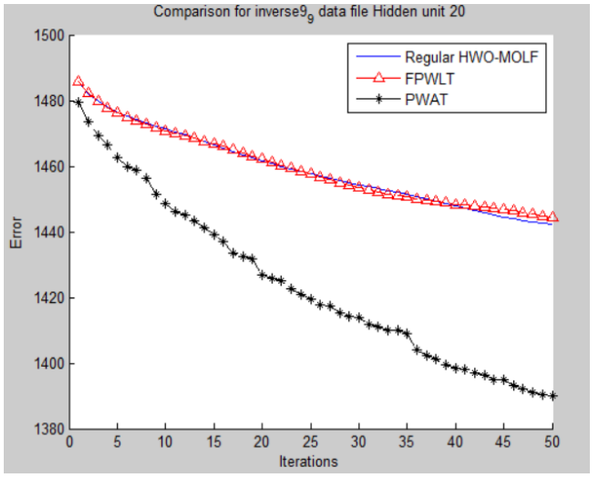
4.6 - Error comparison for Regular HWO-MOLF, FPWLT and PWAT for twod data file

Table 4.1 is the summary of the Error v/s iterations comparison for all the training algorithms described for twod data file. The results are similar to that of the oh7 data file but for the hidden unit 40 the regular HWO-MOLF again shows better results than the PWAT algorithm, but the FPWLT shows even more better performance than the other 2.

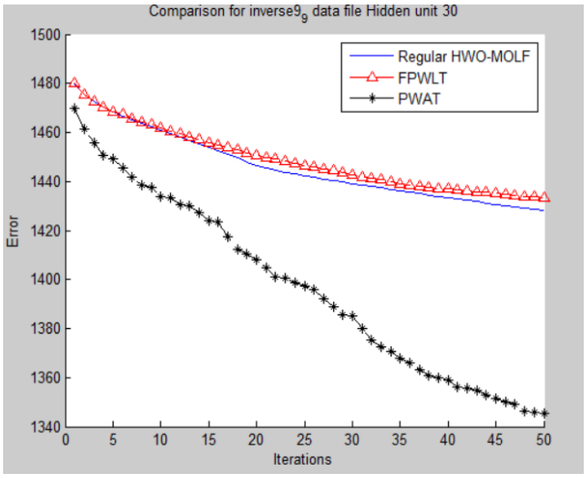
Experiments for inverse9_ data file, inputs =9, outputs = 9, patterns = 10000 iterations =50

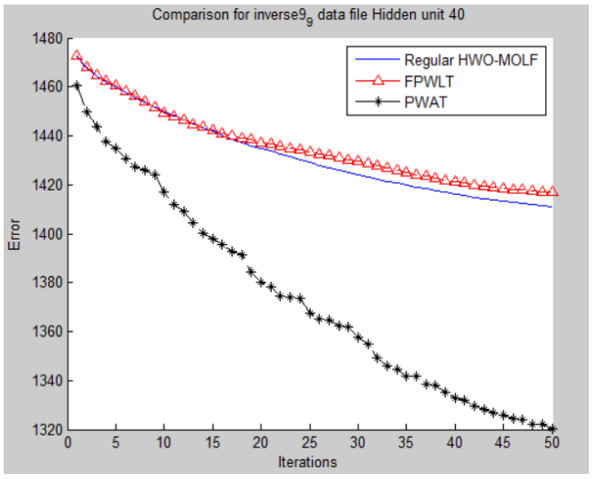


4. 39 Error Comparison for Nh= 8 and inverse9_9 data file



4. 40 Error Comparison for Nh= 20 and inverse9_9 data file





4. 41 Error Comparison for Nh= 30 and inverse9_9 data file

4. 42 Error Comparison for Nh= 40 and inverse9_9 data file

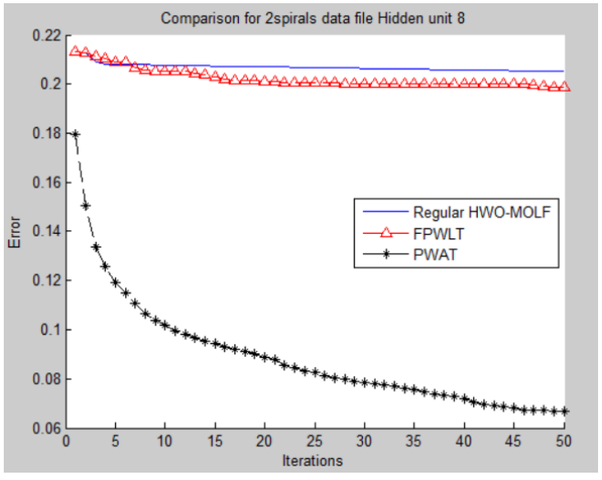
No of Hidden units / Error	8	20	30	40
Regular HWO-MOLF	1.4778e+03	1.4422e+03	1.4284e+03	1.4109e+03
FPWLT	1.4790e+03	1.4444e+03	1.4333e+03	1.4169e+03
PWAT	1.4637e+03	1.3900e+03	1.3456e+03	1.3203e+03

4.7 - Error comparison for Regular HWO-MOLF, FPWLT and PWAT for inverse9_9 data file

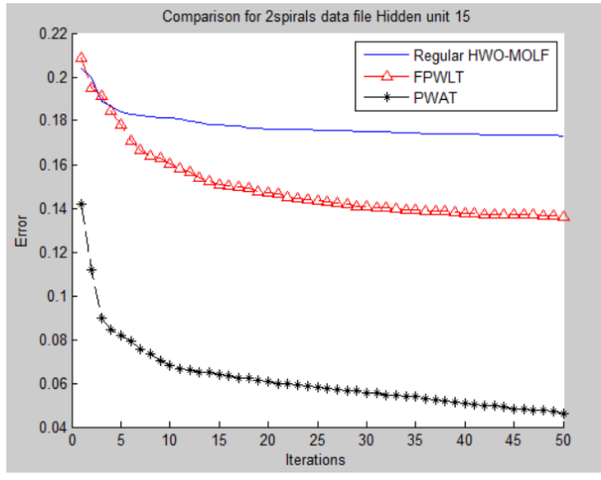
Table 4.1 is the summary of the Error v/s iterations comparison for all the training algorithms described for inverse9_9 data file. The inverse9_9 is considered difficult to train. The results for PWAT algorithm for this data shows astonishing results as from the plots for all hidden units the error gradually decreases at a rate higher than the decrease in error for the other 2 data file.

Experiments for 2spirals data file with inputs =2, outputs = 1, patterns = 10000 iterations =50

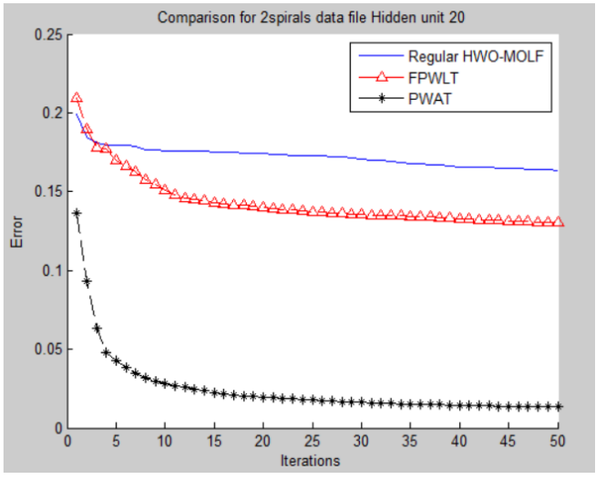
As the number of inputs are 2,which is more than twice as small as the inputs used for other data files so experiments are performed for lower number of hidden units



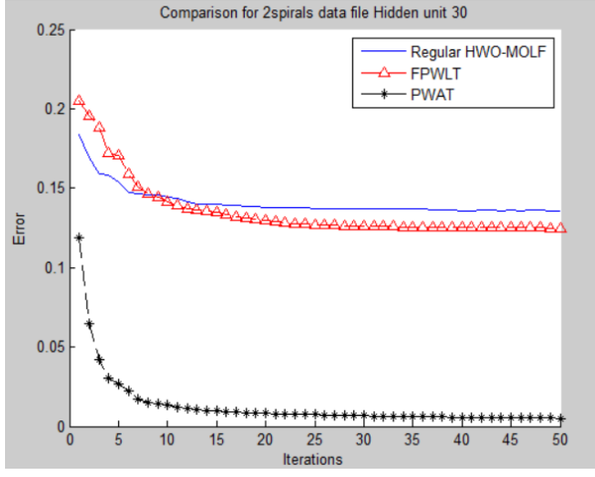
4. 43 Error Comparison for Nh= 8 and 2spirals data file



4. 44 Error Comparison for Nh= 15 and 2spirals data file



4. 45 Error Comparison for Nh= 20 and 2spirals data file



4. 46 Error Comparison for Nh= 30 and 2spirals data file

No of Hidden units / Error	8	15	20	30
Regular HWO-MOLF	0.2051	0.1731	0.1631	0.1359
FPWLT	0.1988	0.1362	0.1304	0.1249
PWAT	0.0666	0.0461	0.0132	0.0051

4.8 - Error comparison for Regular HWO-MOLF, FPWLT and PWAT for 2spirals data file

Table 4.1 is the summary of the Error v/s iterations comparison for all the training algorithms described for 2spirals data file. Similar to the inverse9_9 data file the PWAT algorithm shows better results than the other 2 algorithms and from the figure the error graph remains flat after some iterations as the error reaches optimal minimum by the time the network is being trained.

From all the experiments performed in this section, the final results can be stated that for higher number of hidden units the piecewise linear activations training shows promising results than the regular HWO-MOLF, whereas for lower number of hidden units Regular HWO-MOLF performs.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, a piecewise linear activation for hidden units in a multilayer perceptron is discussed. These piecewise linear activation are trained using the gradient descent and multiple optimal learning factors algorithms along with training of the input and output weights. The experiments performed in Chapter 4 demonstrates that the new piecewise linear activation learning algorithm produces significant results than the regular sigmoidal activations for higher number of hidden units. The activations that are learned in the network at every iteration can overcome the problem of using same activation functions for all the applications. Also, different sets of piecewise linear samples show different final results. Even initialization of the input weights can affect the network's performance.

The experiments in section 4.1 for 2 different versions of piecewise linear activation training proved that both the PWAT versions showed better results than the regular HWO – MOLF, but PWAT version A shows more promising result.

Also, from various experiments performed in section 4.2, for higher number of hidden units the PWAT algorithm performed better than the regular HWO-MOLF than for lower number of hidden units.

5.2 Future work

Improving the piecewise linear activations by adjusting the irregularities of the spacing of the fixed piecewise linear samples. Also improve the network for learning for lower number of hidden units by using right pair of piecewise linear samples during initialization. Initialization of input weights in accordance to the piecewise linear activations.

Deep learning is the main area for research nowadays, and as it consists of more than one hidden layer's this method can be implemented to learn each of the hidden layers hidden units independently depending upon the data pass through the network. The final results are expected to improve the deep neural networks efficiency as in multilayer perceptron with one hidden layer the network performed significantly better for large size of hidden units than the regular HWO-MOLF. Also after improving for lower number of hidden units the new activation training method can be used for any number of hidden units for a deep neural networks.

References

1. AN EFFICIENT PIECEWISE LINEAR NETWORK by Rohit Rawat, University of Texas at Arlington.
2. S. Haykin. *(1994, Neural Networks a Comprehensive Foundation).*
3. H. White. Economic prediction using neural networks: The case of IBM daily stock returns. Presented at *Proceedings of the IEEE International Conference on Neural Networks.*
4. Currency Exchange Rate Prediction and Neural Network Design Strategies A.N. Refenes 1, M. Azema-Barac 1, L. Chen I and S.A. Karoussos.
5. M. W. Craven and J. W. Shavlik. (1997, Using neural networks for data mining. *FGCS.Future Generations Computer Systems 13(2-3),* pp. 211-229.
6. H. Lu, R. Setiono and H. Liu. (1996, Effective data mining using neural networks. *IEEE Trans. Knowled. Data Eng. 8(6),* pp. 957-961.
7. F. L. Lewis, S. Jagannathan and A. Yesildirek. (1998, *Neural Network Control of Robot Manipulators and Nonlinear Systems.*
8. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang. (1989, Phoneme recognition using time-delay neural networks. *Readings in Speech Recognition* pp. 393–404.
9. C. L. Wilson, G. T. Candela and C. I. Watson. (1994, Neural network fingerprint classification. *J. Artif. Neural Networks 1(2),* pp. 203-228.
10. Guyon. (1991, Applications of neural networks to character recognition. *INT.J.PATTERN RECOG.ARTIF.INTELL. 5(1),* pp. 353-382.

11. S. Lawrence, C. L. Giles, A. C. Tsoi and A. D. Back. (1997, Face recognition: A convolutional neural-network approach. *IEEE Trans. Neural Networks 8(1),* pp. 98-113.
12. “Multilayer feedforward networks are universal approximators" Kur Hornik, Maxwell Stinchcombe and Halber White(1989).
13. Statistical Pattern Recognition, Summer 2015 By Dr Michael Manry, University of Texas at Arlington.
14. M. T. Manry, S. J. Apollo, L. S. Allen, W. D. Lyle, W. Gong, M.S. Dawson, and A. K. Fung, "Fast Training of Neural Networks for Remote Sensing", Remote Sensing Reviews, vol. 9, pp. 77-96, 1994.
15. Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture Saurabh Karsoliya. *International Journal of Engineering Trends and Technology- Volume3Issue6-2012*
16. Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961
17. Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations. MIT Press, 1986.
18. Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function *Mathematics of Control, Signals, and Systems,* 2(4), 303–314.
19. Neural networks. II. What are they and why is everybody so interested in them now?Wasserman, P.D.; Schwartz, T.; Page(s): 10-15; IEEE Expert, 1988, Volume 3, Issue 1
20. Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014).Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems* (pp. 2933-2941).

21. Hornik, K., Stinchcombe, M., White, H.: Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural networks 3(5), 551{560(1990)

22. Suitable MLP Network Activation Functions for Breast Cancer and Thyroid Disease Detection, I.S.Isa, Z.Saad, S.Omar, M.K.Osman,K.A.Ahmad, H.A.Mat Sakim, *Second International Conference on Computational Intelligence, Modelling and Simulation.*

23. J. Olvera, X. Guan and M. T. Manry, "Theory of Monomial Networks," *Proceedings of SINS'92,* Automation and Robotics Research Institute, Fort Worth, Texas, pp. 96-101, December 1992

24. Barton, S.: A matrix method for optimizing a neural network. Neural Computation 3(3), 450{459 (1991)

25. Kim, T., Manry, M., Maldonado, J.: New learning factor and testing methods for conjugate gradient training algorithm. In: Neural Networks, 2003. Proceedings of the International Joint Conference on, vol. 3, pp. 2011{2016. IEEE (2003).

26. Maldonado, F. J., and M. T. Manry. "Optimal pruning of feedforward neural networks based upon the Schmidt procedure." *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*. Vol. 2. IEEE, 2002.

27. Rawat, Rohit, Jignesh K. Patel, and Michael T. Manry. "Minimizing validation error with respect to network size and number of training epochs." *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013.

28. J.P. Fitch, S.K. Lehman, F.U. Dowla, S.Y. Lu, E.M. Johansson, and D.M. Goodman, "Ship Wake-Detection Procedure Using Conjugate Gradient Trained Artificial Neural Networks," IEEE Trans. on Geoscience and Remote Sensing, Vol. 29, No. 5, September 1991, pp. 718-726.

29. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “Learning internal representations by error propagation,” in D.E. Rumelhart and J.L.McClelland (Eds.), Parallel Distributed Processing, vol. I, Cambridge,Massachusetts: The MIT Press, 1986.
30. W. Kaminski, P. Strumillo, “Kernel orthonormalization in radial basis function neural networks,” IEEE Transactions on Neural Networks, vol. 8, Issue 5, pp. 1177 - 1183, 1997
31. R.P Lippman, “An introduction to computing with Neural Nets,” IEEE ASSP Magazine,April 1987.
32. Pramod L. Narasimha, Walter H. Delashmit, Michael T. Manry, Jiang Li, Francisco Maldonado, “An integrated growing-pruning method for feedforward network training,” Neurocomputing vol. 71, pp. 2831–2847, 2008.
33. Chen, Sheng, Colin FN Cowan, and Peter M. Grant. "Orthogonal least squares learning algorithm for radial basis function networks." *Neural Networks, IEEE Transactions on* 2.2 (1991): 302-309.
34. Yu, C., Manry, M.T., Jiang, L.: E_ects of nonsingular preprocessing on feedforward network training. International Journal of Pattern Recognition and Arti_cial Intelligence 19(02),217{247 (2005).DOI10.1142/S0218001405004022.URL:http://www.worldscienti_c.com/doi/abs/10.1142/S0218001405004022
35. A Neural Network Training Algorithm Utilizing Multiple Sets of Linear Equations Hung-Han Chen a, Michael T. Manry b, and Hema Chandrasekaran b
36. Hidden Layer Training via Hessian Matrix Information Changhua Yu, Michael T. Manry, Jiang Li
37. S. S. Malalur, M. T. Manry, "Multiple optimal learning factors for feed-forward networks," accepted by The SPIE Defense, Security and Sensing (DSS) Conference, Orlando, FL, April 2010
38. Analysis and Improvement of Multiple Optimal Learning Factors for Feed-Forward Networks Praveen Jesudhas, Michael T. Manry, Rohit Rawat, and Sanjeev Malalur.

39. Hazewinkel, Michiel, ed. (2001), "Linear interpolation", Encyclopedia of Mathematics,Springer, *ISBN 978-1-55608-010-4*

40. 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.

41. A Novel Weight Initialization Method for the Random Neural Network by Stelios Timotheou Intelligent Systems and Networks Group Department of Electrical and Electronic Engineering Imperial College London SW7 2BT, UK