OPTIMIZING A NEURAL NETWORK OVER SIZE

AND ITERATION NUMBER

by

JIGNESH K PATEL

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 UNIVERSITY OF TEXAS AT ARLINGTON

May 2012

ACKNOWLEDGEMENTS

I am heartily thankful to my advisor, Dr. Michael T. Manry, for supervising me on this thesis and guiding me in each step of work.

I wish to thank Dr. Victoria Chen and Dr. Alan Davis for taking time to serve on my thesis committee.

I wish to thank my parents, my sister, my brother-in-law and all my friends for their support.

May 01, 2012

ABSTRACT

OPTIMIZING A NEURAL NETWORK OVER SIZE AND ITERATION NUMBER

Jignesh K. Patel, M.S.

The University of Texas at Arlington, 2012

Supervising Professor: Michael T. Manry

A unique algorithm has been developed for training multilayer perceptron neural networks. First the training algorithm has been used with hidden weight optimization (HWO) and multiple optimal learning factors (MOLF) to get the best performance. In each training iteration, this method first optimally orders the inputs and then optimally orders the hidden basis functions. At the end of each training iteration, the method calculates the validation error versus number of basis function curve in one pass through the validation data. Since, pruning is done at each iteration, we optimize validation error over number of basis functions and number of iterations simultaneously. The number of required multiplies for the algorithm has been analyzed. The method has been compared to others in simulations and been found to work very well.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1 Applications of Neural Networks

Neural networks consist of highly interconnected processing elements working to solve specific problems. A neural network processing element (a neuron) consists of a number of inputs that are multiplied by gains, a threshold that is added to it, and an activation function that performs a nonlinear transformation on the result [56]. Neural networks have an ability to learn [54, 55]. Given an arbitrary data set, a neural network can approximate a mapping from the inputs to the outputs [1]. It has been proved that a multilayer perceptron with sufficiently many nonlinear units (neurons) in a single hidden layer can work as a universal function approximator [34]. Other favorable features include the MLP's ability to mimic Bayes discriminant [35] and MAP estimates [36].

Neural networks have emerged as powerful statistical tools capable of learning and generalization. They have been used in the in the areas of parameter estimation [2] [3], document analysis and recognition [4], finance and manufacturing [5] and data mining [6]. Specific applications of neural networks include target recognition [7, 8], power load forecasting [9, 10], ZIP code recognition [11, 12], prognostics [13], face recognition [14], image retrieval [15] and speaker recognition [16] among others.

1.2 Neural Network Properties

While initially developed to mimic brain functions, neural networks can be thought of as another way of developing approximations. A function of many variables needs to be approximated given only values of the function (often perturbed by noise) at various points in the input space. Typically, a feed-forward network with a single hidden layer, with enough hidden units to approximate the continuous function is selected. The optimum size and topology of the network has been studied in the literature [17, 49, 50, 57]. Neural networks are being increasingly preferred over polynomial approximations. Unlike polynomial approximations, neural networks can be trained for ill-posed patterns, but the mappings they yield for such cases may not be globally accurate [18].

A classifier groups items with similar feature vectors into labeled groups. A nearest neighbor [48] classifier can be designed by using clustering algorithms, and assigning classes to clusters or groups of clusters. When neural network classifiers are designed, there is usually one output per class, and each output functions as the discriminant for the corresponding class.

1.3 Current Research

A major problem in applying neural networks is specifying the size of the network [51] and a lot of research has been done to determine the best network size. The network designer's dilemma occurs because both large and small networks exhibit a number of advantages. Even for moderately sized networks the number of parameters may become large compared to the number of patterns [21]. When a network has too many free parameters (i.e., weights and/or units) not only is learning fast [22]–[25], but local minima are more easily avoided [26]. In particular, a theoretical study [27] has shown that when the number of hidden units equals the number of training examples (minus one), the back propagation error surface is guaranteed to have no local minima only for output weights.

Both theory [28] and experience [29]–[31] show that networks with few free parameters exhibit a better generalization performance, and this is explained by recalling the analogy between neural network learning and curve fitting. Moreover, knowledge embedded in small trained networks is presumably easier to interpret and thus the extraction of simple rules can hopefully be facilitated [32]. Lastly, from an implementation standpoint, small networks only require limited resources in any physical computational environment.

When optimizing the structural complexity of feed-forward neural networks, different size networks are designed which minimize the mean-square training error. Typically, the machine with the smallest validation error is saved. When different size networks are independently initialized and trained, however, the training error versus network size curve is not monotonic. Similarly, the resulting validation error versus size curve is not smooth, making it difficult to find a proper minimum. [33] So, monotonic training curves and smooth validation curves are desirable.

For validation error minimization two approaches can be used. In pruning[50] we start with a large multilayer perceptron with an adequate performance for the problem at hand and then prune it by eliminating certain weights or nodes in a selective and ordered fashion. This approach combines the advantages of training large networks (i.e., accuracy and avoidance of local minima) and those of using small ones (i.e., improved generalization). However it requires advance knowledge of what size is “large” for the problem at hand, but this is not a serious concern as upper bounds on the number of hidden units have been established[49]. By pruning, validation error as a function of network size (number of basis functions) is obtained, which can be non-monotonic. The network with the smallest validation error will be chosen.

In the second approach, the neural network is trained for several iterations and for each iteration validation error is obtained. So, finally we will get a validation error versus number of iterations curve. This curve can also be non-monotonic. A neural network can be trained for the iteration which has minimum validation error.

These approaches only optimize the network over either number of iterations or number of hidden units. In general it is not known how many iterations and hidden units result in the minimum validation error.

1.4 Thesis Goals

In this thesis, an algorithm is developed which optimizes a neural network over size and iteration number simultaneously. In chapter 2, batch training algorithms are reviewed. The multiple optimal learning factor (MOLF) is discussed in chapter 3. Classification is discussed in chapter 4. In chapter 5, first we discuss algorithm to get optimally ordered orthonormalized neural network. The method is discussed to get validation error versus basis functions curve in one pass through validation data using the trained orthonormalized network. So, here we have first pruned the network and then early stopping is applied on validation data set, thus we will get best network size for each iteration. Results for both approximation and classification case are shown in chapter 6.

CHAPTER 2

MLP NOTATION AND TRAINING

2.1 Multilayer Perceptron

The multilayer perceptron (MLP) forms a very important class of neural networks. A typical MLP structure consists of a set of input units that constitute the input layer, one or more layers of neurons forming the hidden layers and an output layer for producing the actual output. All the layers are connected by weights and the signal travels from the input through the hidden layers, to the output layer. Such a network is shown below.

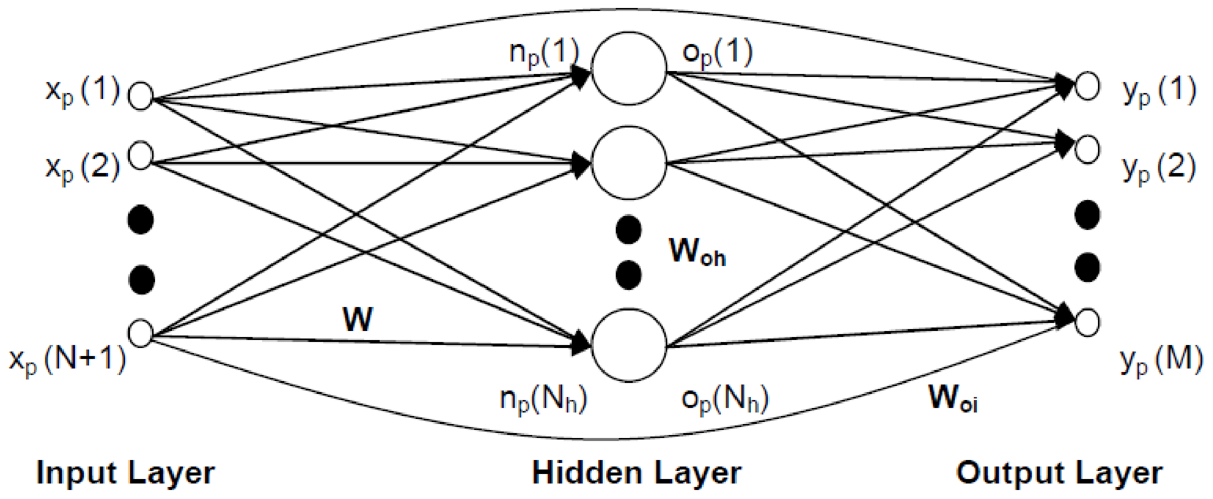

Figure 1 Fully connected Multilayer perceptron

As shown in Figure 1, input weights w(k,n) connect the n^th input $x_p(n)$ to the k^th hidden unit. Output weights $w_{oh}(m,k)$ connect the k^th hidden unit's activation $o_p(k)$ to the m^th output $y_p(m)$, which has a linear activation. The bypass weight $w_{oi}(m,n)$ connects the n^th input to the m^th output. The training data, described by the set ($\mathbf{x}_p$,$\mathbf{t}_p$) consists of N-dimensional input vectors $\mathbf{x}_p$ and M-dimensional desired output vectors*,* **t**$_p$. The pattern number p varies from 1 to N_v where N_v denotes the number of training vectors present in the data set.

In order to handle thresholds in the hidden and output layers, the input vectors are augmented by an extra element $x_p(N+1)$ where, $x_p(N+1) = 1$, so $\mathbf{x}_p = [x_p(1), x_p(2),\ldots., x_p(N+1)]^T$. Let N_h denote the number of hidden units. The dimensions of the weight matrices **W**, $\mathbf{W_{oh}}$ and $\mathbf{W_{oi}}$ are respectively N_h by (N+1), M by N_h and M by (N+1) .The kth element of net vector $\mathbf{n}_p$ can be written as,

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

(2.1)

In matrix notation it can be summarized as,

$$\mathbf{n_p} = \mathbf{W} \cdot \mathbf{x_p}$$

(2.2)

where the kth element of the hidden unit activation vector $\mathbf{o_p}$ is calculated as $o_p(k) = f(n_p(k))$ and f(.) denotes the hidden layer activation function. For the sigmoid activation case, the output of the *k*th hidden unit is given by

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

(2.3)

The actual output of the network, $\mathbf{y}_p$ can be written as,

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

(2.4)

In matrix notation it can be summarized as,

$$\mathbf{y_p} = \mathbf{W_{oi}} \cdot \mathbf{x_p} + \mathbf{W_{oh}} \cdot \mathbf{o_p}$$

(2.5)

The MLP computes its outputs as a weighted sum of the inputs and the hidden unit outputs. The weights form the unknowns, which are typically found by minimizing the mean squared error between the actual and desired outputs,

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

2.2 Output Weight Optimization

Output weight optimization (OWO) is a technique to solve for weights connected to the actual outputs of the network (this would be the output weights, $\mathbf{W_{oh}}$ and by-pass weights, $\mathbf{W_{oi}}$). Since the outputs have linear activation, finding the weights connected to the outputs is equivalent to solving a system of linear equations. The expression for the actual outputs given in (2.6) can be re-written as

$$\mathbf{y_p} = \mathbf{W_o} \cdot \mathbf{X_p} \tag{2.7}$$

where $\mathbf{X_p}$=[$\mathbf{x_p}$, $\mathbf{o_p}$] is the augmented input vector of size $N_u = N + N_h + 1$, $\mathbf{W}_o$ is formed as [$\mathbf{W}_{oi}$: $\mathbf{W}_{oh}$] and has a size of M by N_u. The output weights can be solved by setting $\partial E / \partial \mathbf{W}_o = \mathbf{0}$ which leads to a set of linear equations given by,

$$\mathbf{C} = \mathbf{R} \cdot \mathbf{W_o^T} \tag{2.8}$$

where,

$$\mathbf{C} = \frac{\mathbf{1}}{\mathbf{N_v}} \sum_{\mathbf{p=1}}^{\mathbf{N_v}} \mathbf{X_p} \cdot \mathbf{t_p}^{\mathbf{T}} \tag{2.9}$$

$$\mathbf{R} = \frac{\mathbf{1}}{\mathbf{N_v}} \sum_{\mathbf{p=1}}^{\mathbf{N_v}} \mathbf{X_p} \cdot \mathbf{X_p}^{\mathbf{T}}$$

Popular candidates for solving equation (2.8) are conjugate gradient[38] and the orthogonal least squares methods[39].

2.3 Back Propagation Algorithm

The popular backpropagation (BP) algorithm[52] is a first order method that uses gradient information to update the weights in the network. In full batch mode, the (BP) algorithm updates the input weights and thresholds, **W**, as

$$w(k,n) = w(k,n) + z\left(\frac{-\partial E}{\partial w(k,n)} \right) \tag{2.10}$$

for $1 \le k \le N_h$ and $1 \le n \le (N + 1)$. Here z is the learning factor. For the pth pattern, we use BP to get the partial derivative of E_p (error for the pth pattern) as

$$-\frac{\partial E_p}{\partial w(k,n)} = -\frac{\partial E_p}{\partial n_p(k)} \cdot \frac{\partial n_p(k)}{\partial w(k,n)} \tag{2.11}$$

For the pth pattern, output layer delta and hidden layer delta are,

$$\delta_p(k) = o'_p(k) \sum_{m=1}^{M} \delta_{po}(m) w_{oh}(m,k)$$

$$\delta_{po}(m) = 2\left(t_p(m) - y_p(m)\right) \tag{2.12}$$

Now, the negative gradient of E is

$$-\frac{\partial E}{\partial w(k,n)} = g(k,n) = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p(k) x_p(n) \tag{2.13}$$

$$\mathbf{G} = \frac{1}{N_v} \sum_{p=1}^{Nv} \boldsymbol{\delta}_p \mathbf{x}_p^T$$

where $\boldsymbol{\delta}_p = \left[\delta_p(1), \delta_p(2), \ldots, \delta_p(N_h)\right]^T$. If steepest descent is used to modify the hidden weights, **W** is updated in a given iteration as

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{z} \cdot \mathbf{G} \tag{2.14}$$

$$\mathbf{\Delta W} = \mathbf{z} \cdot \mathbf{G} \tag{2.15}$$

First order methods are generally easier to implement and also require the least computation per training iteration. They are also guaranteed to converge to a global or local minimum. However, these benefits are largely outweighed by the method's rate of slow convergence [40].

2.4 Output Weight Optimization-Backpropagation

One option to train an MLP would be to divide the weight adaptation into two separate stages: (i) train all weights, $\mathbf{W_{oh}}$, $\mathbf{W_{oi}}$ connected to the actual network outputs[53] and (ii) train all the input weights,**W**. During either stage, the other weights are not updated. This approach combines the two previously described approaches and is called output weight optimization-backpropagation (OWO-BP).

This method is attractive for several reasons. First, the training is faster, since training weights connected to the outputs is equivalent to solving for linear equations. Second, it helps us avoid some local minima. Third, the method exhibits improved training performance.

A brief description of OWO-BP is given below. For every training iteration,

1) Find the negative Jacobian matrix **G** described in equation (2.13)
2) Find the optimal learning factor $z = \frac{-\partial E / \partial z}{\partial^2 E / \partial z^2}$ and update the input weights, **W**, using equation (2.14)
3) Solve the system of linear equations described in section 2.2, using OLS and update the output weights, $\mathbf{W_o}$.

2.5 Hidden Weight Optimization

In HWO, the hidden weights are updated by minimizing separate error functions for each hidden unit. The error functions measure the difference between the desired and the actual net function. For the kth hidden unit and pth pattern, the desired net function is constructed as[42]

$$n_{pd}(k) \cong n_p(k) + z \cdot \delta_p(k) \qquad (2.16)$$

where $n_{pd}(k)$ is the desired net function and $n_p(k)$ is the actual one in (2.1). z is the learning factor and $\delta_{po}(m)$ and $\delta_p(k)$ are given in equation (2.12).

The hidden weights are to be updated as

$$w_{hi}(k,n) \leftarrow w_{hi}(k,n) + z \cdot g_{hwo}(k,n) \qquad (2.17)$$

where $g_{hwo}(k, n)$ is the weight change. The weight changes are derived using

$$n_p(k) + z\delta_p(k) \cong \sum_{n=1}^{N+1}\left[w_{hi}(k,n) + z \cdot g_{hwo}(k,n)\right]x_p(n) \qquad (2.18)$$

Therefore,

$$\delta_p(k) \cong \sum_{n=1}^{N+1}\left[g_{hwo}(k,n)\right]x_p(n) \qquad (2.19)$$

The error of (2.18) and (2.19) for the kth hidden unit is measured as

$$E_{\delta}(k) = \frac{1}{N_v} \sum_{n=1}^{N_v} \left[\delta_p(k) - \sum_{n=1}^{N_v} g_{hwo}(k,n) \cdot x_p(n) \right]^2 \tag{2.20}$$

Equating the gradient of $E_d(k)$ with respect to the hidden weight change to zero, we have

$$\sum_{n=1}^{N+1} g_{hwo}(k,n) \cdot r_i(n,m) = g_{bp}(k,m) \tag{2.21}$$

where

$$r_i(n,m) = \frac{1}{N_v} \sum_{p=1}^{N_v} x_p(n) x_p(m) \qquad \begin{array}{l} \text{for } 1 \le n \le N+1 \\ \text{for } 1 \le m \le N+1 \end{array}$$

$$g_{bp}(k,m) = \frac{1}{N_v} \sum_{p=1}^{N_v} \left[\delta_p(k) - x_p(m) \right] = \frac{-\partial E}{\partial w_{hi}(k,m)} \tag{2.22}$$

During the first half of the training epoch, OWO updates the output weights. During the second half of the training epoch, the hidden weights are updated using HWO as (2.17), where $\mathbf{G}_{hwo}$ is a matrix of hidden weight changes. Thus OWO-HWO is similar to OWO-BP with the BP component replaced by HWO.

Setting to zero the derivative of $E_{\delta}(k)$ with respect to $g_{hwo}(j,m)$, we get,

$$\mathbf{G_{hwo} R_i = G_{bp}} \tag{2.23}$$

Instead of directly using the negative Jacobian $\mathbf{G}_{bp}$ to update the hidden weights, HWO minimizes a separate error function, given by (2.20) and solves the system of linear equations in (2.21) and updates the hidden weights using (2.17). Equation (2.23) can be re-written as

$$\mathbf{G_{hwo} = G_{bp} R_i^{-1}} \tag{2.24}$$

where $\mathbf{R}_i$ is the autocorrelation matrix defined in (2.22). OWO-HWO is summarized below. For every training epoch

- Solve the linear equations and update the output weights
- Find the negative Jacobian matrix $\mathbf{G}_{bp}$ described in equation (2.22)
- Solve the linear equations in (2.21) to find the HWO and update $\mathbf{G}_{hwo}$
- Compute the optimal learning factor.
- Update the hidden weights using (2.17)

CHAPTER 3

MULTIPLE OPTIMAL LEARNING FACTOR TRAINING

In this chapter, a new learning algorithm called the multiple optimal learning factor (MOLF) algorithm [44] is reviewed. MOLF calculates an optimal learning factor for every hidden unit, in order to increase the speed of learning and overall convergence. In MOLF Newton's method is used in order to calculate a separate optimal learning factor for each hidden unit's input weights in a feedfoward network.

In the past, researchers have used multiple learning rates and/or momentum terms in order to speed up the learning process. Unfortunately, these methods are mostly heuristic, and their performance relies on the settings of some user chosen parameters. Also, using standard gradients makes them slow to converge. The Newton's method can be viewed as a second order method to assign a learning rate to every weight in the network. Elements of the new method's Hessian matrix are shown to be weighted sums of elements from the total network's Hessian. First we will discuss basic algorithm to generate one optimal learning factor.

3.1 Basic MOLF Algorithm

A MLP is assumed to be trained using OWO-BP. However, let **z** be a vector whose kth element z_k represent the OLF used to update the input weights, w(k,n).The error function to be minimized is given by (2.6). The predicted output $y_p(m)$ is given by,

$$y_p(m) = \sum_{n=1}^{N+1} w_{oi}(m,n) x_p(n) + \sum_{k=1}^{N_h} w_{oh}(m,k) f\left(\sum_{i=1}^{N+1} \left(w(k,i) + z_k g(k,i) x_p(i) \right) \right) \qquad (3.1)$$

where, g(k,i) is again an element of the negative Jacobian matrix **G** and f() denotes the hidden layer activation function. The negative first partial derivative of E with respect to an OLF z_j is,

$$g_{molf}(j) = \frac{-\partial E}{\partial z_j} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} \left[\bar{t}_p(m) - \sum_{k=1}^{N_h} w_{oh}(m,k) o_p(z_k) \right] w_{oh}(m,j) o'_p(j) \Delta n_p(j) \tag{3.2}$$

where $o_p^{'}(j)$ indicates the first partial derivative of $o_p(j)$ with respect to its net function and,

$$\bar{t}_p(m) = t_p(m) - \sum_{n=1}^{N+1} w_{oi}(m,n) x_p(n)$$

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

$$o_p(z_k) = f\left(\sum_{n=1}^{N+1} \left(w(k,n) z_k g(k,n) \right) x_p(n) \right) \tag{3.3}$$

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

$$h_{molf}(k,j) \approx \frac{\partial^2 E}{\partial z_k \partial z_j} = \frac{2}{N_v} \sum_{m=1}^{M} w_{oh}(m,k) w_{oh}(m,j) \sum_{p=1}^{N_v} o'_p(k) o'_p(j) \Delta n_p(k) \Delta n_p(j)$$

$$= \sum_{i=1}^{N+1} \sum_{n=1}^{N+1} \left[\frac{2}{N_v} u(k,j) \sum_{p=1}^{N_v} x_p(i) x_p(n) o'_p(k) o'_p(j) \right] g(k,i) g(j,n) \tag{3.4}$$

where,

$$u(j,k) = \sum_{m=1}^{M} w_{oh}(m,j) w_{oh}(m,k)$$

The Gauss-Newton update guarantees that $\mathbf{H}_{molf}$ is non-negative definite. Given the negative gradient vector and the Hessian $\mathbf{H}_{molf}$, the error E is minimized with respect to the vector **z** using Newton's method. For every iteration in the training algorithm, the steps are as follows:

i. Find the negative Jacobian matrix **G** described in equation (2.13)

ii. Solve **z** using $\mathbf{H}_{molf} \cdot \mathbf{z} = \mathbf{g}_{molf}$ and update the input weights as $w(k,n) \leftarrow w(k,n) + z_k \cdot g(k,n)$

iii. Solve linear equations for all output weights.

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

3.2 MOLF-HWO

The hidden weight optimization (HWO) technique was introduced in section 2.5. In this section we are showing that HWO is immune to the presence of linearly dependent inputs during training. We then replace the BP component in MOLF with HWO. The resulting improved versions of MOLF is shown to be unaffected by linearly dependent inputs and better than using BP.

3.2.1 Effect of Dependence on HWO

HWO finds the input weight update by solving for a system of linear equations as given by equation (2.21), reproduced below for convenience.

$$\mathbf{G_{hwo}} \cdot \mathbf{R_i} = \mathbf{G_{bp}} \tag{3.5}$$

If one of the inputs to the network is linearly dependent, clearly it would cause the input auto-correlation matrix, $\mathbf{R_i}$ to be singular. In such a situation, using the conjugate gradient (C-G) algorithm to solve (3.5) would lead to poor training, since the convergence of C-G is affected by the presence of linearly dependent inputs. However, using OLS or inversion methods could prove useful in detecting and eliminating the linearly dependent input, as analyzed in the following subsections.

Orthogonal Least Squares:

Using OLS to solve for $\mathbf{G_{hwo}}$ in (3.12) involves computing the orthonormal weight update, $\mathbf{G'_{hwo}}$, as

$$\mathbf{G'_{hwo}} = \mathbf{G_{bp}} \cdot \mathbf{Ci}^{\mathbf{T}} \tag{3.6}$$

where $\mathbf{C_i}$ is a lower triangular matrix of orthonormal coefficients of dimension (N+1). The orthonormal weight update can be mapped to the original weight update as

$$\begin{aligned} \mathbf{G'_{hwo}} &= \mathbf{G_{hwo}} \cdot \mathbf{C_i} \\ &= \mathbf{G_{gp}} \cdot \mathbf{C_i}^{\mathbf{T}} \mathbf{C_i} \end{aligned} \tag{3.7}$$

Assume $x_p(N+2)$ was linearly dependent. This would cause the $(N+2)^{th}$ row and column of $\mathbf{R_i}$ to be linearly dependent. During OLS, a singular auto-correlation matrix transforms to the $(N+2)^{th}$ row of $\mathbf{C_i}$ to be zero. The expression for $\mathbf{G_{hwo}}$ contains $\mathbf{C_i}^T\mathbf{C_i}$, which will be a square, symmetric matrix with zeros for the $(N+2)^{th}$ row and column. This would reflect in $\mathbf{G_{hwo}}$ having zeros for the $(N+2)^{th}$ column. The implication is that the weight update vector computed for all input weights connected to the dependent input (N+2) is zero. These weights are not updated during training, effectively freezing them. This is highly desirable, as the dependent input is not contributing any new information. Thus HWO-type update using OLS is perfectly capable of picking up linearly dependent inputs, leading to a very robust training algorithm.

CHAPTER 4

CLASSIFICATION

Neural network classifiers trained using the mean squared error (MSE) objective function have been shown to approximate the optimal Bayesian classifier. Although the expectation value of the classifier error rate is considered to be the ideal training criteria, training algorithms that are based on the minimization of the expected squared error criteria are often easier to mechanize and better understood. MLP, commonly used neural network classifier, is designed through iterative regression - minimizing the mean squared error (MSE) between the target class and a network mapping function.

The MLP approximates the Bayes discriminant, when designed by minimizing the standard training error as in equation (2.6), where M is the number of classes, N_v denotes the total number of training patterns, E(i) is mean-squared error for the ith class. $t_p(i)$ denotes the ith desired output and $y_p(i)$ denotes the ith observed output. The ith desired output for the pth pattern is defined as

$$t_p(i) = \begin{cases} 1 & \text{for } i = i_c \\ 0 & \text{for } i = i_d \end{cases} \tag{4.1}$$

where i_c denotes the correct class number for the current training pattern and i_d denotes any incorrect class number for that pattern. The output can be written as in equation (2.4).

Output activations $y_p(i_c)$ are used as class discriminants. The classifier is said to have correctly classified the pth pattern when $y_p(i_c)$ is the largest observed output, otherwise we indicate that the network has misrecognized the pth pattern by incrementing the classification error count. In either case, squared residuals accumulate in the standard training error as we step through the set of N_v training patterns.

4.1 Algorithm for Calculating Percentage Classification Error

Algorithm for calculating the probability of error and percentage classification error:

(1) Initially N_{er} = 0, P_e = 0 and Percentage Classification Error = 0

(2) For $1 \le p \le N_v$, read the input vector ($\mathbf{X}_p$) and the correct class (i_c)

(3) Find the output vector ($\mathbf{y}_p$) using input vector ($\mathbf{X}_p$) and output weight matrix ($\mathbf{W_o}$)

(4) Find the index i for which $y_p(i)$ is maximum, that index i is the estimated class

(5) If estimated class = correct class then go to step(7)

(6) Increment N_{er}

(7) If the entire data file is read then go to step(8) else go to step(2)

(8) $P_e = N_{er}/N_v$, Percentage Classification error= $(100 \times N_{er})/ N_v$

4.2 Problems with Regression Based Classifiers

Let $N_v(i)$ denote the number of patterns belonging to class i, S(i) denote the set of patterns that correspond to class i, $f_x(\mathbf{x}_p)$ denote the probability density of feature vector $\mathbf{x}_p$, $f(\mathbf{x}_p|i)$ the conditional density of feature vector $\mathbf{x}_p$ given that it belongs to class i, and P(i) the probability that the feature vector comes from class i.

The optimal Bayesian or minimum probability of error discriminant $d_p(i)$ for feature vector $\mathbf{x}_p$ is given as

$$d_p(i) = P(i|\mathbf{x}_p) \tag{4.2}$$

where $P(i|\mathbf{x}_p)$ is the aposteriori probability given that vector $\mathbf{x}_p$ belongs to class i. Let the desired output $t_p(i)$ for correct class i_c in this case be defined as

$$t_p(i)= \delta(i-ic) \tag{4.3}$$

Then the expected squared error between network output y(i) and optimal Bayes discriminant is given by

$$E_B = \sum_{i=1}^{M} E[(d(i) - y(i))^2] \tag{4.4}$$

where E[.] is the expectation operator and y(i) is a random variable storing the ith actual network output. Note that $y_p(i)$ is the actual ith output when the input vector is $\mathbf{x}_p$.

Theorem 2: As the training patterns N_v increase, E approaches the (E_B + K), where K is a constant.

Thus E = E_B + K, implies that network with estimated output $\mathbf{y}_p$, which minimizes the MSE yields the optimal Bayes discriminant function in the minimum mean squared error sense.

Unfortunately; classifiers designed using regression have the following problems:

1) Theorem 1 does not mention anything about the network structure.

2) Minimizing E_B is not the same as minimizing P_e even though y(i) = d(i) would minimize P_e .For example, if the actual outputs are better than the desired outputs as $y_p(i_c) > t_p(i_c)$ or $y_p(i_d) < t_p(i_d)$ then the MSE(E) increases but the probability of classification error decreases.

3) The weight matrix $\mathbf{W_o}$ obtained by minimizing the MSE is not optimal; the weight matrix for the SVM or Bayes classifier is not obtained by minimizing the MSE.

4.3 The potential for MSE-Based Optimal Classifiers

Let $\mathbf{W_o}^{\mathbf{opt}}$ denote an output weight matrix for a classifier that is optimal in some sense, such as the minimum probability of error or Bayes classifier, or the SVM.

Theorem 2: $\mathbf{W_o}^{\mathbf{opt}}$ is the solution to a least squares problem

Proof: Given $\mathbf{W_o}^{\mathbf{opt}}$, the optimal cross-correlation matrix $\mathbf{C}^{opt}$ is,

$$\mathbf{C}^{opt} = \mathbf{R}\cdot(\ \mathbf{W_o}^{\mathbf{opt}}\ )^T \tag{4.5}$$

The optimal cross-correlation matrix $\mathbf{C}^{opt}$ can be expressed as

$$\mathbf{C^{opt}} = \frac{\mathbf{1}}{\mathbf{N_v}} \sum_{\mathbf{p=1}}^{\mathbf{N_v}} \mathbf{X_p}\left(\mathbf{t^{opt}}_{\mathbf{p}}\right)^{\mathbf{T}} \tag{4.6}$$

If $\mathbf{C}^{opt}$ and $\mathbf{R}$ are known in equation (4.6), we can solve for $\mathbf{W_o}^{\mathbf{opt}}$. In designing the classifier, we assume that the basis vectors $\mathbf{X}_p$ are given and unchangeable. The only component that is under user control is the desired output vector $\mathbf{t}_p$. $\mathbf{C}^{opt}$ can be found from the above equations by using good desired outputs $\mathbf{t}_p^{opt}$ since the basis vectors $\mathbf{X}_p$ are unchangeable. So in order to get $\mathbf{W_o}^{\mathbf{opt}}$ we need to use good desired outputs $\mathbf{t}_p^{opt}$. We can get good desired outputs $\mathbf{t}_p^{opt}$ by changing the desired outputs $\mathbf{t}_p$ appropriately.

Various classifiers with desirable properties can be designed through regression, merely through the proper choice of the desired output $\mathbf{t}_p^{opt}$.

Assume that $\mathbf{C}^{opt}$ is available and that we want to generate the desired output vectors $\mathbf{t}_p$.

$$\begin{bmatrix} c(i,1) \\ c(i,2) \\ \vdots \\ c(i,L) \end{bmatrix} = \begin{bmatrix} t(i,1) \\ t(i,2) \\ \vdots \\ t(i,N_v) \end{bmatrix} \begin{bmatrix} a(1,1) & a(1,2) & \cdots & a(1,N_v) \\ a(2,1) & a(2,2) & \ldots & a(2,N_v) \\ \ldots & & \ldots & \\ a(L,1) & a(L,2) & \ldots & a(L,N_v) \end{bmatrix}$$

where c(i,n) is an element of $\mathbf{C}^{opt}$. The nth row pth column element of the L by N_v matrix $\mathbf{A}$ is $X_p(n)/N_v$, and t(i,p) denotes the ith element of $\mathbf{t}_p$. Assuming that $L < N_v$ and that $\mathbf{A}$ has rank L, above equation is an underdetermined set of equations for the vector $t_p(i)$ and has uncountably many exact solutions. Assume that $\mathbf{C}^{opt}$ is available and that we want to generate the desired output vectors $\mathbf{t}_p$. It is easy to generate the desired output vectors that produce $\mathbf{C}^{opt}$. The catch is that this must be done without pre-knowledge of $\mathbf{W}_o^{opt}$ or $\mathbf{C}^{opt}$. Iterative algorithms designed to find the desired outputs $\mathbf{t}_p^{opt}$ are called Output Reset (OR) algorithms and the iterations are called OR iterations.

4.4 Output Reset Algorithm

We have used an improved output reset (OR) algorithm which, through manipulation of local error biases, relaxes the MSE objective function which is then applied to training of neural network classifiers. Compared to conventionally-designed MLPs, the OR-designed networks have proved superior for complicated classification problems. The OR algorithm, combined with enhanced MOLF-HWO training, greatly improves classifier performance. Details for the OR algorithm are described in [37].

CHAPTER 5

PRUNING IN EACH ITERATION

5.1 Introduction

The unique one pass pruning method is presented which uses orthogonal least squares. Our aim is to get a monotonically non-increasing graph of validation minimum square error versus basis functions. Here, OLS is used to eliminate less useful inputs and hidden units and also to detect linear dependent basis function. This algorithm first optimally orders the inputs and then hidden units. The Schmidt procedure is used to get orthonormal basis functions.

In section 5.2 we are getting the optimal ordering of basis function, according to their usefulness to get the output layer. In 5.3, using that orthonormalized network we are getting monotonically non-increasing graph of $E_v(N_u)$. We are repeating this procedure for every iteration and at the end of each iteration network with the least validation error is saved. Our algorithm is described is section 5.4 for both approximation and classification case. The computational burden of algorithm is calculated in section 5.5. The MOLF-HWO described in section 3.2 is used as a training algorithm.

5.2 Ordered pruning

The purpose of pruning is to eliminate less useful inputs and hidden units which have no information relevant for estimating outputs or are linearly dependent on inputs or hidden units that have already been orthonormalized. In section 4.2, we have only discussed OLS type OWO, optimal order of the basis functions has not been taken into account. In this section we modify the Schmidt procedure so that during pruning useless basis functions **x,** are eliminated.

Let o(m) be the new optimal order of the neural network, so that o(m) specifies the order in which raw basis function $\mathbf{x_k}$ will be processed into orthonormal basis function $\mathbf{x'_i}$. Then **x** is to be calculated from $x_{o(m)}, x_{o(m-1)}, \dots$ and so on. This function also defines the structure of the new hidden layer where 1 ≤ m

≤ N_u and 1 ≤ o(m) ≤ N_u. If o(m) = *k* then the m^th unit of the new structure comes from the k^th unit of the original structure. Given the function o(m), the m^th orthonormal basis function is described as,[45]

$$x'_m = \sum_{k=1}^{m} a_{mk} x_{o(k)} \tag{5.1}$$

Initially, x_1^i is found as $a_{11}x_{j(1)}$where,

$$a_{11} = \frac{1}{\left\| x_{o,(1)} \right\|} = \frac{1}{r(o(1),o(1))^{1/2}} \tag{5.2}$$

For 2 ≤ m ≤ N_u, we first perform,

$$c_i = \sum_{q=1}^{i} a_{iq} r(o(q),o(m)) \tag{5.3}$$

For 1 ≤ i ≤ m-1, Second, we set b_m=1 and get

$$b_k = -\sum_{i=k}^{m-1} c_i a_{ik} \tag{5.4}$$

Lastly for 1 ≤ k ≤ m-1, we get coefficient a_{mk} as,

$$a_{mk} = \frac{b_k}{\left[r(o(m),o(m)) - \sum_{i=1}^{m-1} c_i^2 \right]^{1/2}} \tag{5.5}$$

For 1 ≤ k ≤ m, $w'_0(i,m)$ is found as,

$$w'_0(i,m) = \sum_{k=1}^{m} a_{mk} c(i,o(k)) \tag{5.6}$$

The goal of ordered pruning is to find the function o(m) which defines the structure of the basis function. Here it is assumed that the original basis functions are linearly independent i.e. the denominator of equation (5.5) is not zero.

Since we want the effects of inputs and the constant ‘1’ to be removed from the orthonormal basis functions, the first N+1 basis functions are picked as,

$$o(m) = m, \qquad \text{for } 1 \le m \le N+1$$

5.2.1 Ordering Inputs

First the selection process will be used to optimally order the inputs N+1. We now define notation that helps us specify the set of candidate basis function to choose in a given iteration. First, define S(m) as the set of indices of chosen basis functions where m is the number of inputs. Then S(m) is given by

$$S(m) = \begin{cases} \{\varphi\} & \text{for } m = 0 \\ \{o(1), o(2), \ldots o(m)\} & \text{for } 0 < m \le N+1 \end{cases} \tag{5.7}$$

Let's take o(1) = 1, putting the threshold as a first input. the set of candidate basis functions is clearly $S^c\{m\}$ = {1,2,…,N+1}-S(1)*,* which is {1,2,…,N+1}. For 2 < m ≤ N+1, we obtain S^c(m-1). For each trial value of o(m) € S^c {m-1} perform operations (5.3), (5.4), (5.5), and (5.6). Then P(m) is

$$P(m) = \sum_{i=1}^{M} [w'_o(i,m)]^2 \tag{5.8}$$

The trial value of o(m) that maximizes P(m) is found. Assuming that P(m) is maximum when testing the k^th element, then o(m) = k. S(m) is updated as

$$S(m) = S(m-1) \cup \{o(m)\} \tag{5.9}$$

Then for the general case the candidate basis functions are S^c (m-1) = {1,2,…,N+1}-{o(1),o(2),…,o(m-1)}, with N-m candidate basis function. By using equation (4.25) after testing all the candidate basis function, *o(m)* takes its value and *S(m)* is updated according to equation (4.26). The process is repeated until m = N+1.

5.2.2 Ordering Hidden Units

Same process as for inputs will be repeated for hidden units. Now S(m) is,

$$S(m) = \begin{cases} \{\varphi\} & \text{for } m = 0 \\ \{j(1), j(2), \ldots j(m)\} & \text{for } 0 < m \le N_u \end{cases} \tag{5.10}$$

Starting with an initial network which has zeros hidden unit, m = N+1, the set of candidate basis functions will be $S^c\{m\}$ = {1,2,…,N_u}-S(m), which is {N+2,N+3,…,N_u}. For N+2 ≤ m ≤ N_u, we obtain S^c(m-1). For each trial value of o(m) € S^c {m-1} perform operations (5.3), (5.4), (5.5), and (5.6). Then P(m) is found as in (5.8). The trial value of o(m) that maximizes P(m) is found. Then o(m) will get its value and S(m) is updated as in (5.9).

Then for the general case the candidate basis functions are S^c (m-1) = {1,2,…,N_u}-{o(1),o(2),…,o(m-1)}, with N_u-m+1 candidate basis function. After testing all the candidate basis function, o(m) takes its value and S(m) is updated according to equation (5.9). Defining N_{hd} as the desired number of units in the hidden layer, the process is repeated until m = N+1+N_{hd}.

After the o(m) function is complete, both the original basis functions and the orthonormal ones are ordered. Then the orthonormal weights are mapped to normal weights. Considering the final value of o(m), row reordering of the original input weights matrix is performed for generating the right $o_p(j)$, values. After reordering the rows, as only the N_{hd} units are kept, the remaining units ($o_p(j)$, with $N_{hd} < j \le N_h$) are pruned by deleting the last N_h - N_{hd} rows.

5.2.3 Linear Dependency Condition

Assume that a raw basis function, $x_{j(m)}$ is linearly dependent on previously chosen basis functions, where j(m) denotes an input ($1 \le m \le N$) and j(m) has taken on a trial value. Then

$$x_{j(m)} = \sum_{k=1}^{m-1} d'_k \, x'_k \tag{5.11}$$

Now the denominator of a_{mk} in (5.5) can be rewritten as

$$g = \langle z_m, z_m \rangle^{1/2} \tag{5.12}$$

where,

$$z_m = x_{j(m)} - \sum_{i=1}^{m-1} \langle x'_i, x'_{j(m)} \rangle x'_i \tag{5.13}$$

Substituting (5.11) into (5.13), however, we get

$$\langle x'_i, x_{j(m)} \rangle = d_1 \tag{5.14}$$

and z_m and *g* are both zero.

If o(m) denotes an input, and *g* satisfies

$$g < \varepsilon$$

for ε = 10^{-27}**,** then we perform

$$j(k) = k$$

for 1 ≤ k < m, and

$$j(k) = k + 1$$

for m ≤ k ≤ N.

In effect we decrease N by one and let the j(k) function skip over the linearly dependent input. If j(m) denotes a hidden unit, the same procedure is used to determine whether or not $x_{j(m)}$, is useful. If $x_{j(m)}$ is found to be linearly dependent, the current, bad value of j(m) is discarded before a_{mk} is calculated.

5.3 One pass validation

In section 5.2, we have got optimal ordered neural network. Now the idea is to combine it with the early stopping. For that we will need validation error versus basis functions $E_v(N_u)$ curve.

Given the matrix **A** and the MLP network with ordered basis functions, we wish to generate the validation error versus N_u curve $E_v(N_u)$ from the validation data. The vector $\mathbf{x_k}$ is converted into orthonormal basis function by transformation,

$$x'_m = \sum_{k=1}^{m} a_{mk} x_k \quad \text{for } 1 \le k \le m-1 \tag{5.15}$$

In order to get the validation error for all size networks in a single pass through the data, we use the following strategy. Let $y_p(i,m)$ represent the ith output of the network having m hidden units for the pth pattern, let $E_v(m)$ represent the mean square error of the network for validation with m hidden units. First, the linear network output is obtained and the corresponding error is calculated as follows [46]:

$$y_p(i,0) = \sum_{k=1}^{N+1} w'(i,k) \cdot x'_p(k) \qquad \text{for } 1 \le i \le M$$

$$E_v(0) \leftarrow E_v(0) + \sum_{i=1}^{M} \left[t_p(i) - y_p(i,0)\right]^2 \tag{5.16}$$

Then for 1 ≤ m ≤ N_h, the following two steps are performed. For 1 ≤ i ≤ M,

$$y_p(i,m) = y_p(i,m-1) + w'(i,N+1+m) \cdot x'_p(N+1+m) \tag{5.17}$$

$$E_v(m) \leftarrow E_v(m) + \sum_{i=1}^{M}\left[t_p(i) - y_p(i,m)\right]^2 \tag{5.18}$$

Apply equations (5.16)-(5.18) for 1 ≤ p ≤ N_v and get the total validation error over all the patterns for each size network. Then these error values should be normalized as

$$E_v(m) \leftarrow \frac{E_v(m)}{N_v} \qquad \text{for } 0 \le m \le N_h \tag{5.19}$$

Thus we generate the validation error versus the network size curve in one pass through the validation data set.

5.4 Combining Pruning with Training at each iteration

5.4.1 Approximation case

In section 5.2, we got network with optimal order of basis functions and in section 5.3 we generated $E_v(N_u)$ curve for validation data in one data pass. Now, we want to optimize validation error not only over N_u, but also over N_{it} . Hence, we are optimizing E_v over N_v and N_{it} simultaneously.

At each iteration, we will first train the network using MOLF-HWO algorithm, then we will use OLS type OWO to optimally order basis functions. Using that we will do one pass validation and get the desired $E_v(N_u)$ curve. The detailed steps for the approximation are given below.

Firstly, take Gaussian random variables as initial input weights and perform net control to get desired net function ($\mathbf{n_p}$) mean is equal to 0.5 and desired variance is equal to 1. Use OWO-OLS to get initial output weights. And then for each iteration we are doing following steps,

1) Calculate $\mathbf{R}_i$ and $\mathbf{G}_{bp}$ using equation (2.22), using those solve equation (2.21) using OLS method to get $\mathbf{G}_{hwo}$.
2) Calculate $\mathbf{G}_{molf}$ and $\mathbf{H}_{molf}$ using equations (3.4) and (3.6) respectively. Then calculate multiple learning factor **z** using OLS method.
3) Update the input weights using, $w(k,n) = w(k,n) + z(k) \cdot g_{hwo}(k,n)$

4) Using changed input weights calculate auto-correlation and cross-correlation matrix using equation (2.9).
5) Generate an optimally ordered orthonormal network using discussed OWO-OLS algorithm.
6) Using validation data get the $E_v(N_u)$ curve using the one pass validation technique.
7) Store the original network weights for each network size, if the validation error for that network size is decreased than that for the previous iteration. There will be total N_h+1 networks, one network with N+1 inputs.
8) Get the original network from the orthonormal one and calculate outputs of the network.
9) If the maximum number of iterations is not reached, then go to step (1).

5.4.2 Classification case

In this section, we will discuss the algorithm for the classification case. Our goal is the same as for the approximation case to optimize the validation error over number of iterations and network size simultaneously. We have used improved OR algorithm discussed in section 4.4, to get the best results.

Firstly, generate desired output for each pattern using equation (4.1). Then take the Gaussian random variable as initial input weights, and do net control. Generate initial output weights using OWO-OLS algorithm. Here, the classification validation error is being calculated as described in section 4.1. Follow the below procedure for each iteration.

1) Calculate the optimal desired output $\mathbf{t}_p^{opt}$, using improved OR algorithm as described in section 4.2
2) Calculate $\mathbf{R}_i$ and $\mathbf{G}_{bp}$ using equation (2.22), using those solve equation (2.21) using OLS method to get $\mathbf{G}_{hwo}$. During calculation of output deltas consider the optimal desired output.
3) Follow step (2) and (3) as described for approximation case.
4) Using the changed input weights calculate the outputs of the network.
5) Calculate the optimal desired output using improved OR algorithm.
6) Using changed input weights and changed desired outputs calculate auto-correlation and cross-correlation matrix using equation (2.9).
7) Follow step (5), (6), (7), (8) and (9) as described for approximation case.

After repeating the above procedure (in both the approximation and the classification case) for N_{it} iterations and capturing the best size of the network, we can generate a unique number of basis functions versus number of iteration $N_u(N_{it})$ curve. From looking at the curve one can easily decide the best size of the network and it can be very useful to compromise between performance and network complexity.

5.5 Computational cost

In this section the computational burden on the algorithm is calculated. The number of multiplies required to solve output weights using OLS is,

$$M_{ols}=N_u[M(N_u+2)+\frac{3}{4}N_u(N_u+1)] \tag{5.20}$$

The gradient $\mathbf{G_{bp}}$ and auto-correlation for inputs $\mathbf{R_i}$, are calculated using equation (2.22). To calculate HWO gradient $\mathbf{G_{hwo}}$ as in equation (2.21) using OLS the number of multiplies required is,

$$M_{hwo}=N_v[N_h(M+N+3)]+M_{ols\text{-}hwo} \tag{5.21}$$

where,

$$M_{ols\text{-}hwo}=(N+1)[(N+2)(N_h+\frac{N}{3})+(N+1)] \tag{5.22}$$

The multiplies required to calculate multiple optimal learning factors **z** using OLS is,

$$M_{molf}=N_vN_h[(N+2)+M(N_h+3)]+M_{ols\text{-}molf} \tag{5.23}$$

where,

$$M_{ols\text{-}molf}=(N_h+1)[\frac{N_h(N_h+2)}{3}+(N_h-1)] \tag{5.24}$$

To optimally order the basis functions using the method described is section 5.2 multiplications required is,

$$M_{ols\text{-}order}=\frac{N_u+1}{6}[12M+N_h^2(7N_u+6M+6)+3MN_u]-\frac{N_h(N+1)}{6}[(N+2)(2N+3M+6)+6M]-M^2N \tag{5.25}$$

The number of multiplies required for one pass validation error calculation is,

$$M_{val}=N_{vv}[N_h(N+2)+\frac{N_u}{2}(N_u+4M+1)] \tag{5.26}$$

where, N_{vv} is number of patterns in validation data set. So, for MOLF-HWO required multiplication is given as,

$$M_{\text{molf-hwo}} = M_{\text{molf}} + M_{\text{hwo}} \tag{5.27}$$

Using all the above equations, we will now compare the number of multiplies required for three different methods: Optimization alone, Pruning alone and our method (described in above section).

The total number of multiplies required for optimization alone is,

$$M_{\text{opti-alone}}=M_{\text{molf-hwo}}+M_{\text{ols}}+N_v[2N_h(N+2)+2M(N_u+1)+N_u(M+N_u)] \tag{5.28}$$

The validation error versus N_h curve is not obtained and basic OLS is used not optimally ordered OLS.

Total number of multiplies required for pruning alone is,

$$M_{\text{pruning-alone}}=N_{it}(M_{\text{molf-hwo}}+M_{\text{ols}}+N_v[N_h(N+2)+N_u(2M+N_u)])+M_{\text{ols-order}}+M_{\text{val}} \tag{5.29}$$

Here, pruning is only done for one time after the MOLF-HWO algorithm, so $M_{\text{ols-order}}$ and M_{val} is required only one time. N_{it} is number of iterations used.

The total number of multiplies required for algorithm described is section 5.5.1 for approximation case is,

$$M_{\text{our-app}}=M_{\text{molf-hwo}}+N_v[N_h(N+2)+N_u(2M+N_u)]+M_{\text{ols-order}}+M_{\text{val}} \tag{5.30}$$

Now, for classification case, as we are using percentage classification error, the multiplies required for one pass validation is,

$$M_{\text{val-class}}=N_{vv}[N_h(N+2)+\frac{N_u}{2}(N_u+2M+1)] \tag{5.29}$$

As described in section 5.5.2, we are updating the optimal output using the improved OR algorithm that does not require any multiplication. Since we are calculating output and output energy again before optimal ordering using OLS, there will be $N_vM(N_u+1)$ extra multiplications. So, the total multiplication required for classification case is,

$$M_{\text{our-class}}=M_{\text{molf-hwo}}+N_v[N_h(N+2)+N_u(2M+N_u)+M(N_u+1)]+M_{\text{ols-order}}+M_{\text{val-class}} \tag{5.30}$$

All the equations given in this section contains the multiplications required for each iteration except equation (5.29). The number of multiplies required for the different data sets for the three methods discussed are compared. It shows that the difference between the number of multiplies required for our method and other methods is nominal.

Table 1 Multiplies comparison for approximation data sets

Data set	N_h and N_{it}	Optimization alone	Pruning alone	Our method
twod.tra N_v=1179, N_{vv}=589	30,70	889997418	844420410	1038341838
	20,25	167127204	155940191	181807292
mattrn N_v=1333, N_{vv}=667	8,15	18382848	16402750	18875505
	20,10	44885773	42630526	49537582
single2 N_v=6667, N_{vv}=3333	30,15	790464561	730732164	855030201
	40,10	799807719	755503467	893477904
oh7 N_v=10000, N_{vv}=5000	50,30	5416831071	5120699091	6120338781
	20,15	773617115	715074610	824905932

Table 2 Multiplies comparison for classification data sets

Data set	N_h and N_{it}	Optimization alone	Pruning alone	Our method
gongtrn N_v=2000, N_{vv}=1000	10,15	102699853	97913213	110353168
	30,50	1561123122	1511006622	1745170422
speech_class N_v=1456, N_{vv}=728	10,15	272950893	265673947	296793768
	30,50	3616546838	3534898032	4021433088

CHAPTER 6

RESULTS

In this chapter, results for the algorithms shown in chapter 5 are presented. The algorithms are implemented in Matlab and Microsoft Visual C. We have compared our algorithm with pruning alone and optimizing alone. In pruning alone we are doing pruning only one time after the network is trained completely using MOLF algorithm. And 10-fold validation is used to generate the results.

We will have the validation error of the network with all different sizes at each iteration. So, we could plot 3-D of validation error versus network size and number of iteration. The local minima of this plot will be the network with the least validation error and that will be the best network.

We are also saving the best network for each size. So, if one wants the network with the particular size, he will know how much training is needed. So that it does not get over-trained. This saves a lot of computational cost.

6.1 Approximation Results

6.1.1 twod data set

The validation error versus the number of iterations is plotted for optimizing E_v over N_{it} alone using MOLF-HWO algorithm and for our method as described in previous chapter. So, the first method considers all the hidden units for each iteration. Since our method does pruning at each iteration it considers the best network at each iteration. Number of hidden units used, N_h = 30.

In figure 3, a 3-D contour of validation error versus hidden units and iterations is shown. The minimum of that curve is the best network for given number of hidden units and iterations.

In figure 4, the validation error versus number of hidden units is plotted for pruning alone and for our method. For pruning order OWO-OLS is used. As we are doing pruning at each iteration our method has a better result.

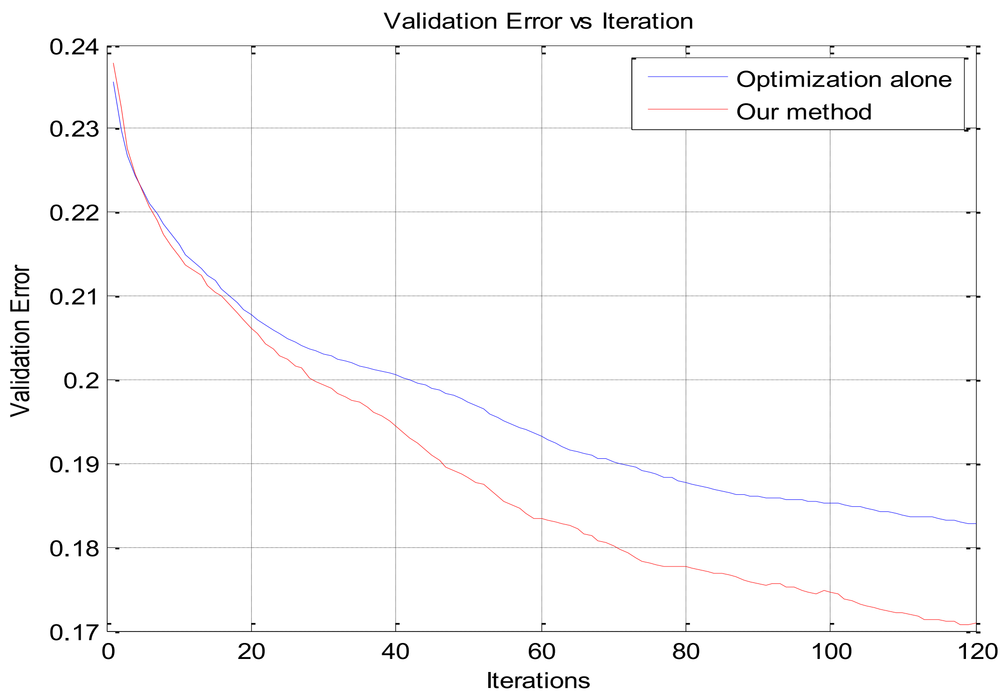

Figure 2 E_v vs N_{it} for twod.tra, N_h=30, N_{it}=120

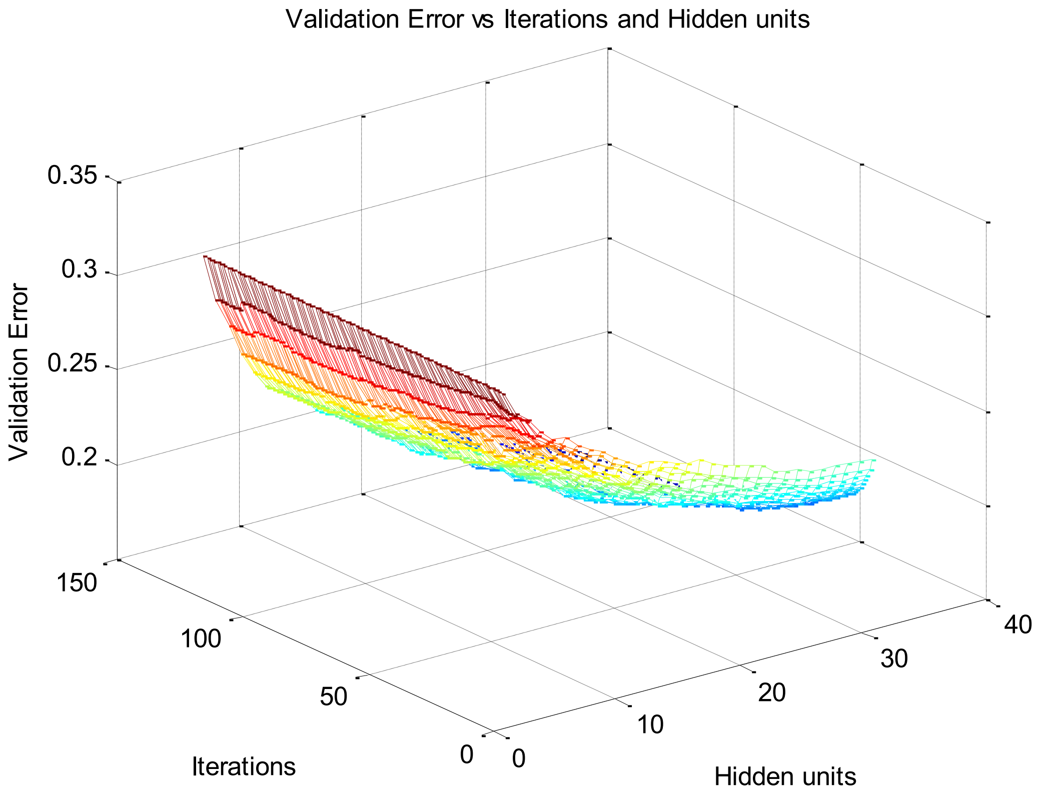

Figure 3 E_v vs N_{it} and N_h for twod.tra, N_h=30, N_{it}=120

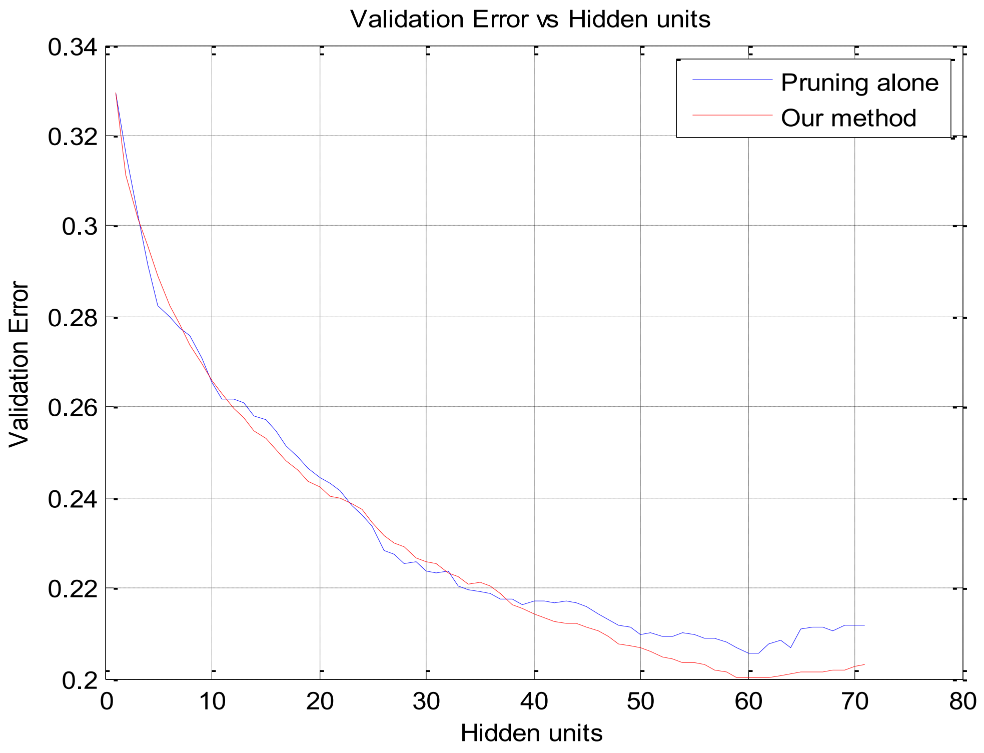

Figure 4 E_v vs N_h, for twod.tra, N_{it}=10,N_h=70

6.1.2 mattrn data set

Here, the validation error decreases very fast for even a small number of hidden units and iterations. For comparison with optimization alone over N_{it}, N_h=70 and N_{it}=20 is taken. In figure 7, our method is compared with pruning alone, N_h=30 and N_{it}=15 is taken.

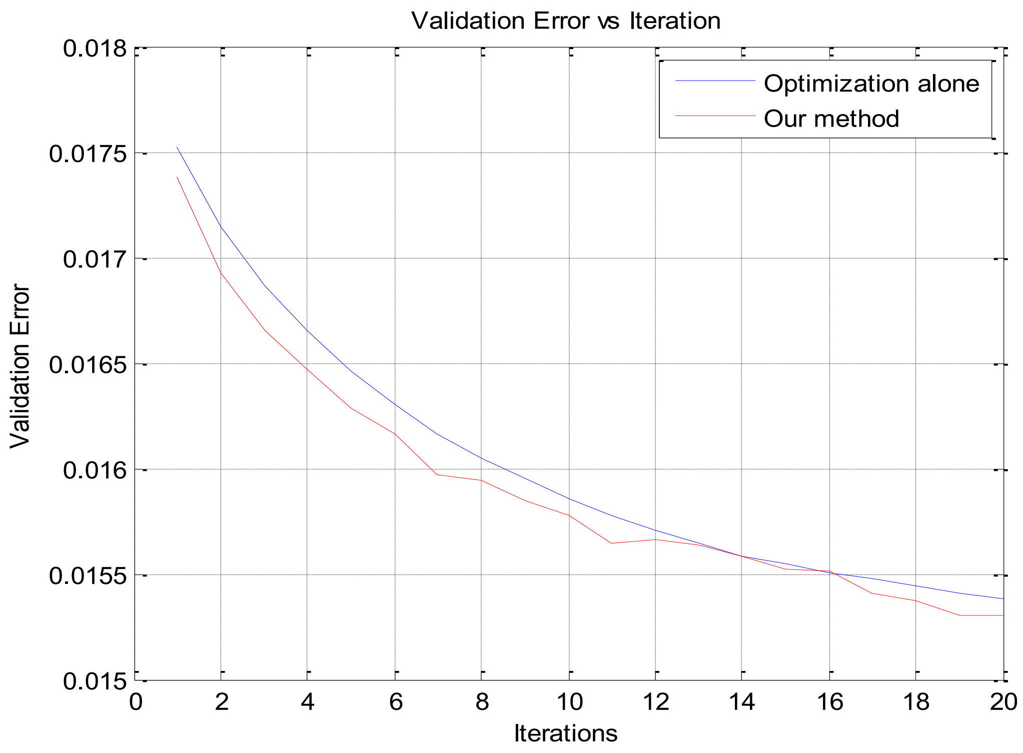

Figure 5 E_v vs N_{it}, for mattrn, N_{it}=20, N_h=70

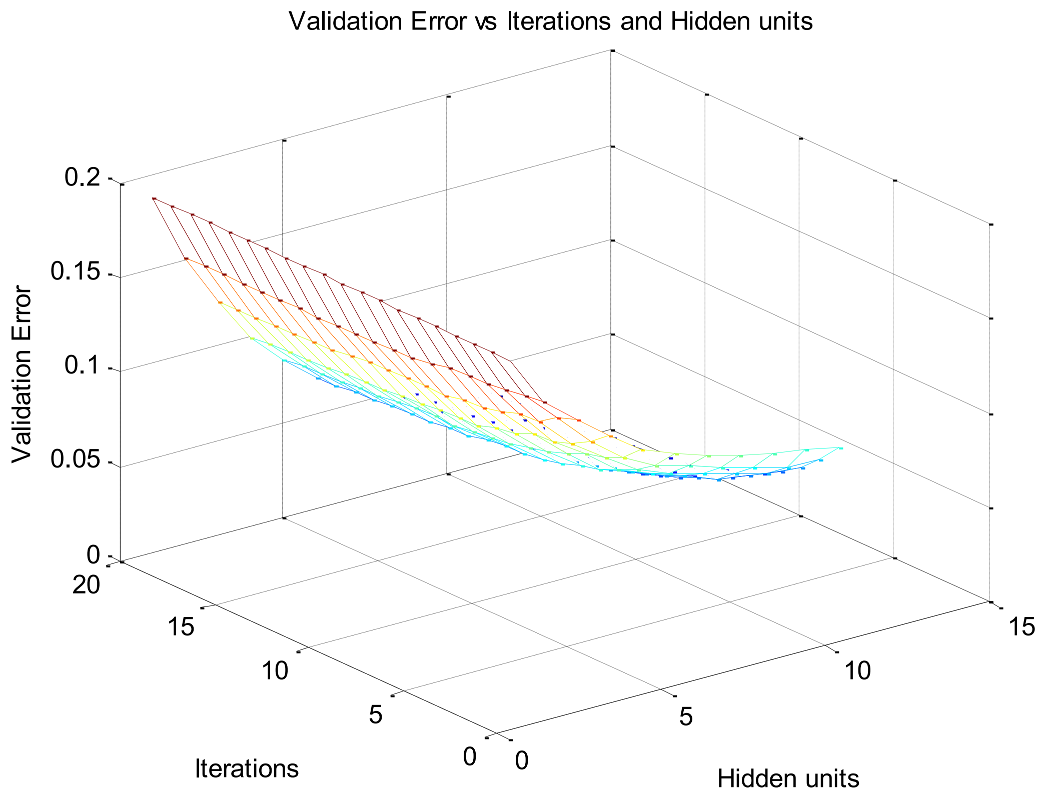

Figure 6 E_v vs N_{it} and N_h, for mattrn, N_{it}=20, N_h=10

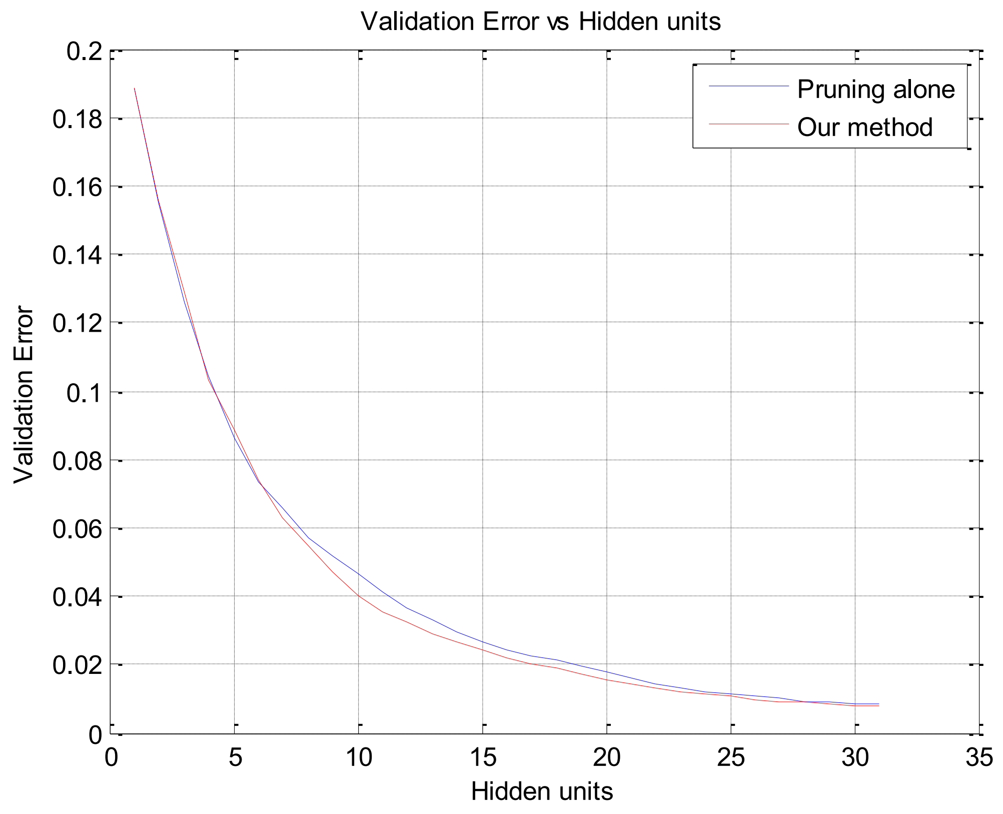

Figure 7 E_v vs N_h, for mattrn, N_{it}=15, N_h=30

6.1.3 single2 data set

Figure 8 clearly shows that at each iteration, pruning has huge effect on validation error, here N_{it}=18 and N_h=15. While comparing, our method with pruning alone, network with less hidden units has no effect. But as the hidden unit increases the validation error decreases faster and we are getting best result at N_h=26 then error increases again, N_h=30 and N_{it}=15 is used.

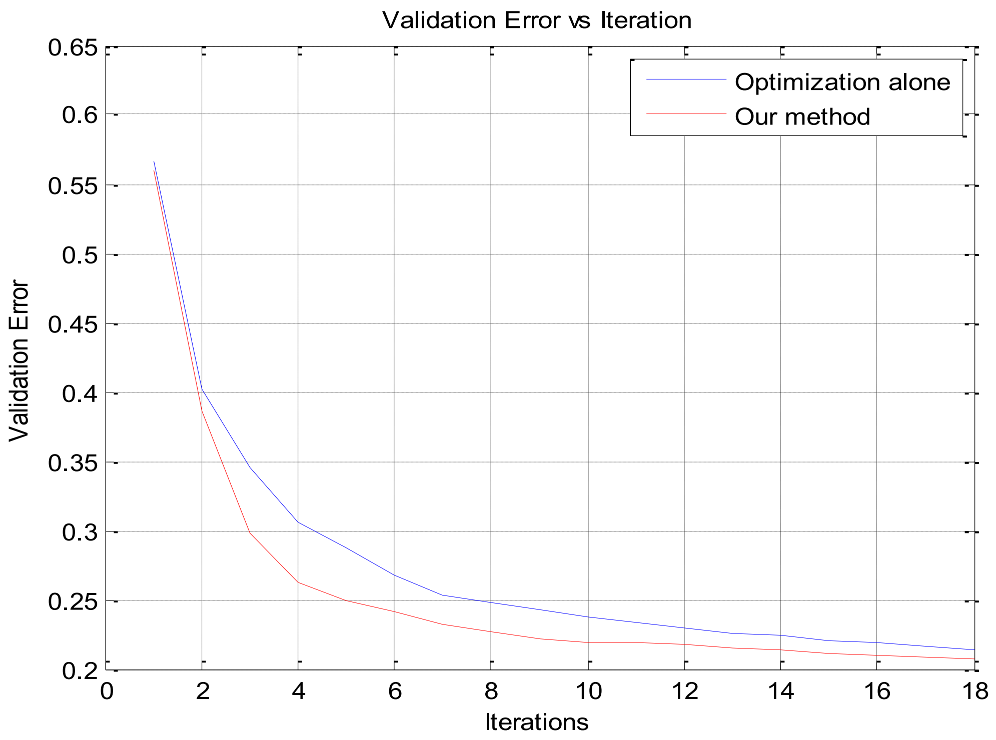

Figure 8 E_v vs N_{it}, for single2, N_h=15, N_{it}=18

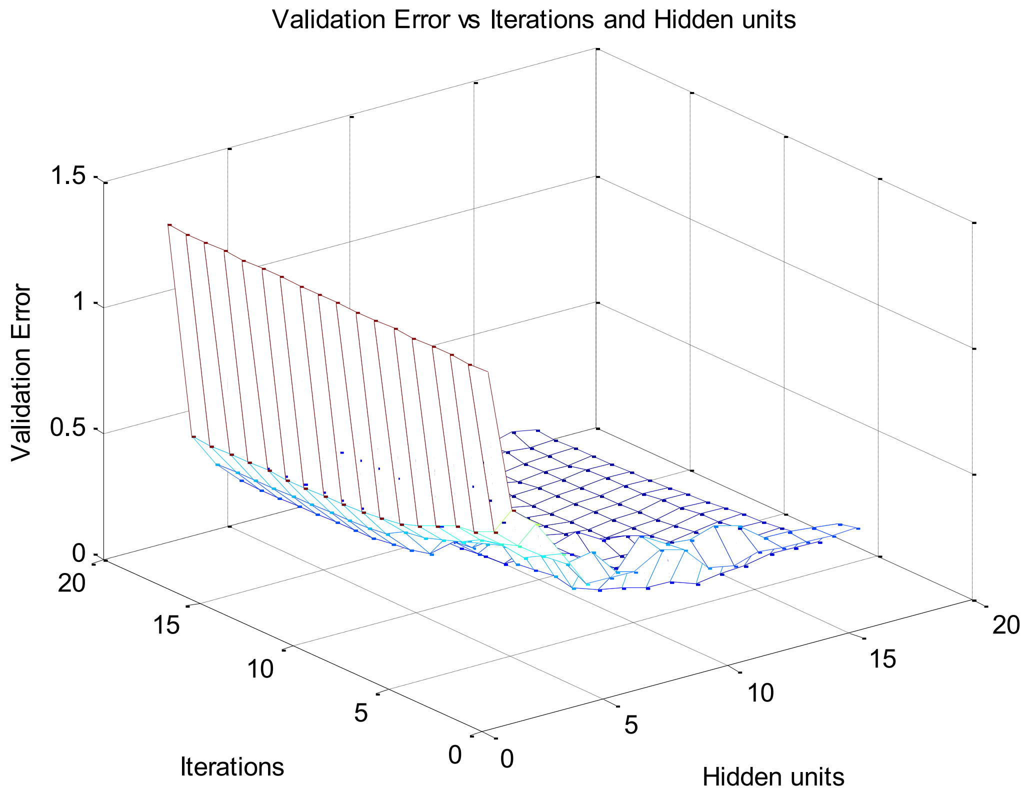

Figure 9 E_v vs N_{it}, for single2, N_h=15, N_{it}=18

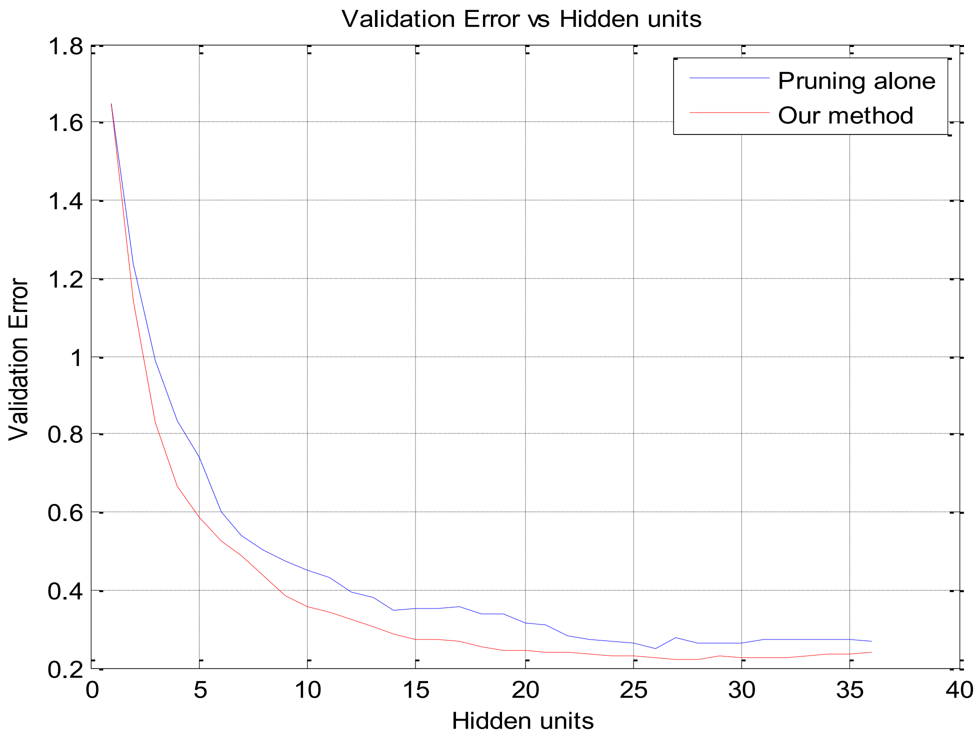

Figure 10 E_v vs N_h, for single2, N_h=35, N_{it}=15

6.1.4 oh7 Data set

For a low number of iterations, pruning has not affected the error very much, but after the 4th iteration pruning affects greatly improves the error, here N_{it}=10 and N_h=25. A 3-D contour of validation error versus hidden units and iterations is shown. In figure 12, the validation error versus number of hidden units is plotted for both methods. N_h=3 and N_{it}=18 is taken.

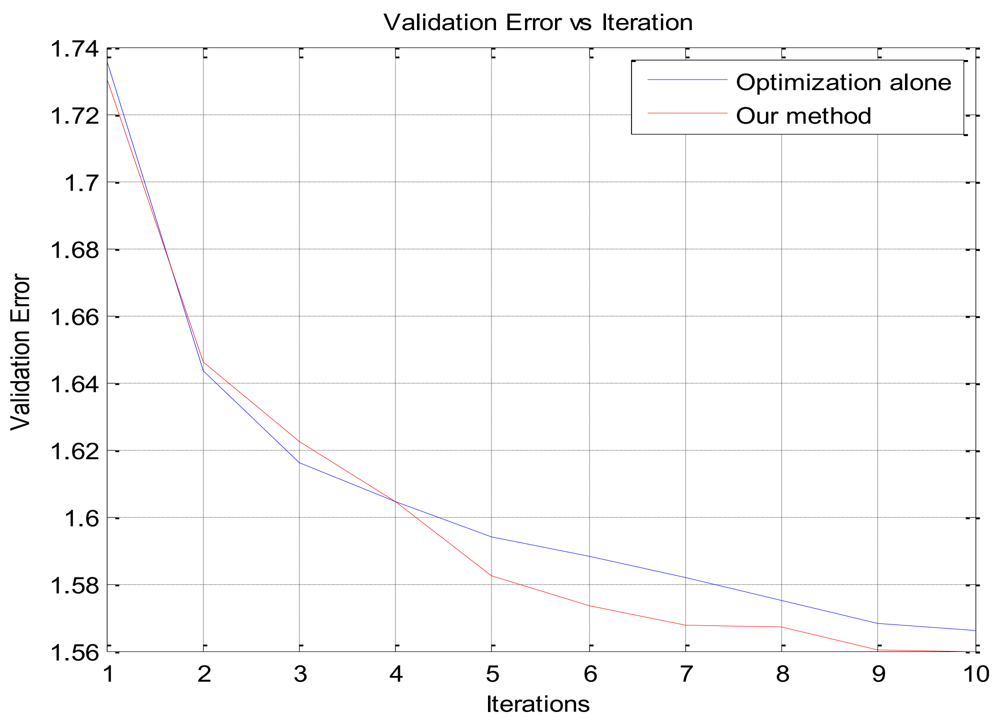

Figure 11 E_v vs N_{it}, for oh7, N_h=25, N_{it}=10

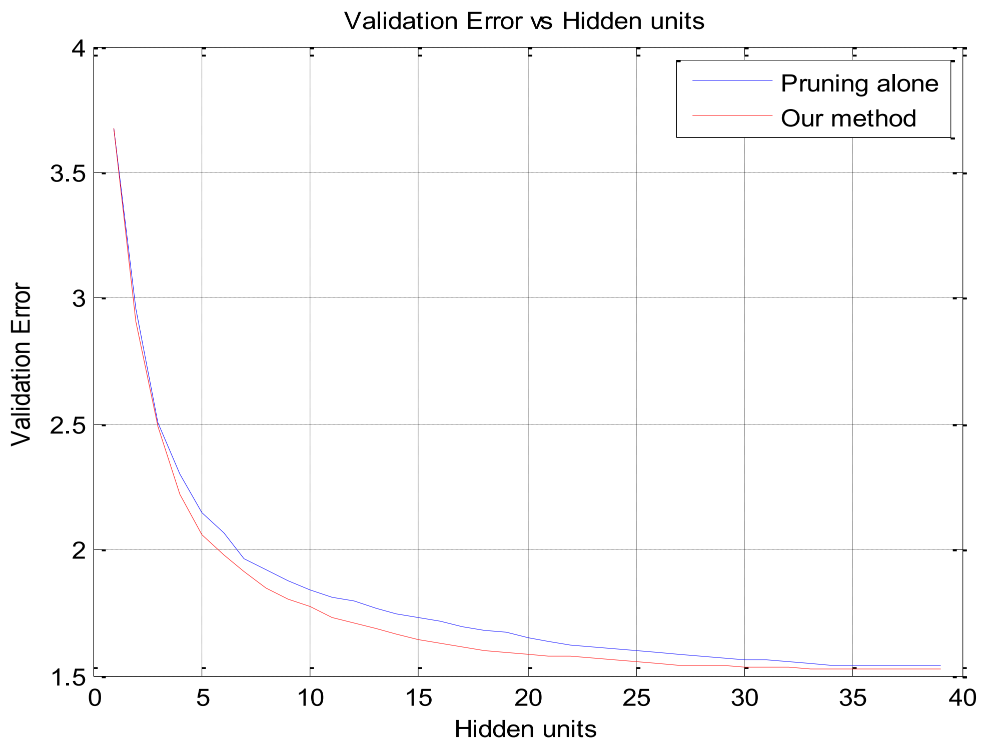

Figure 12 E_v vs N_h, for oh7, N_h=38, N_{it}=18

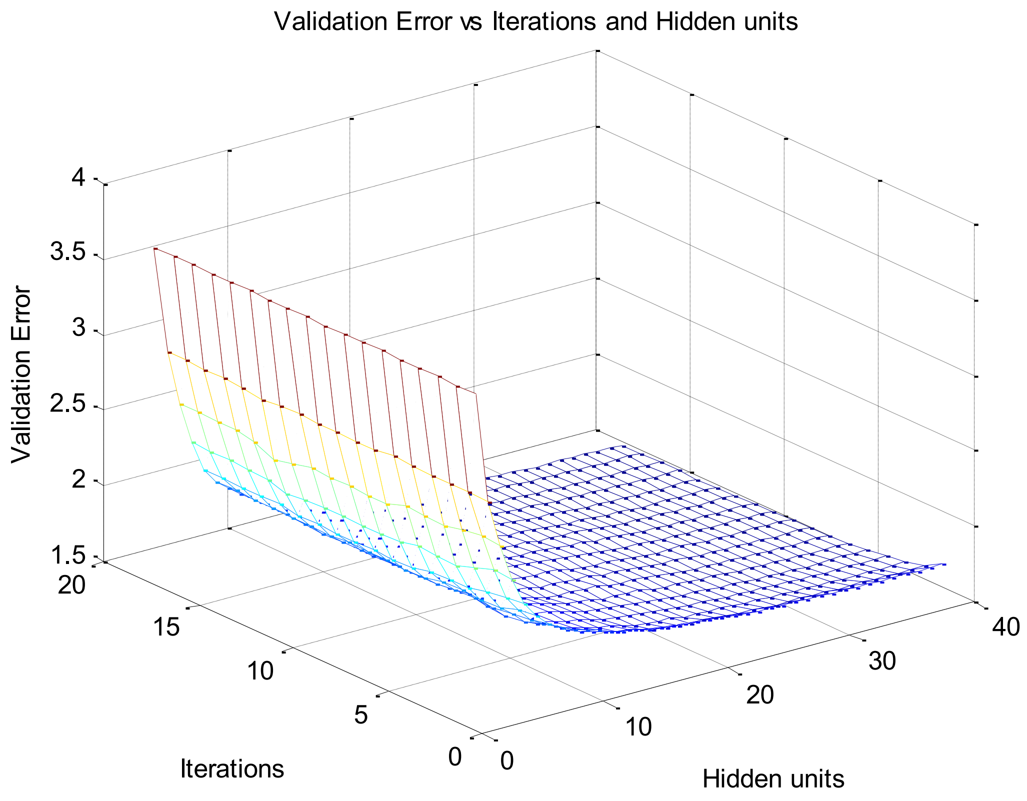

Figure 13 E_v vs N_{it} and N_h, for oh7, N_h=38, N_{it}=18

6.2 Classification results

6.2.1 gongtrn data set

The validation error here is a percentage classification error calculated as discussed in 4.1. For the gongtrn data, both methods are tested for validation error versus iterations. It is clear from figure 14 that our method gives much better classification result than optimization alone. Figure 16 shows that despite in our method starting error is more, but it decreases very fast until the 23th hidden units and then it increases and never decreases again, here, N_h=30 and N_{it}=15. A 3-D contour of validation error versus hidden units and iterations is shown, here, N_h=15 and N_{it}=50.

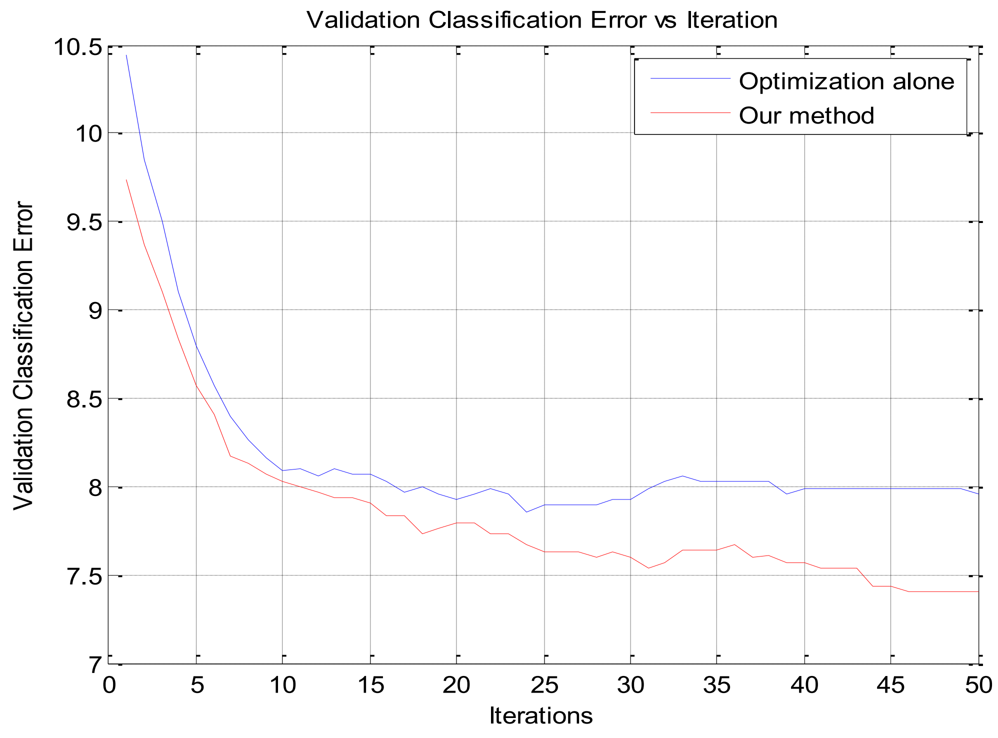

Figure 14 E_v vs N_{it}, for gongtrn, N_{it}=50, N_h=15

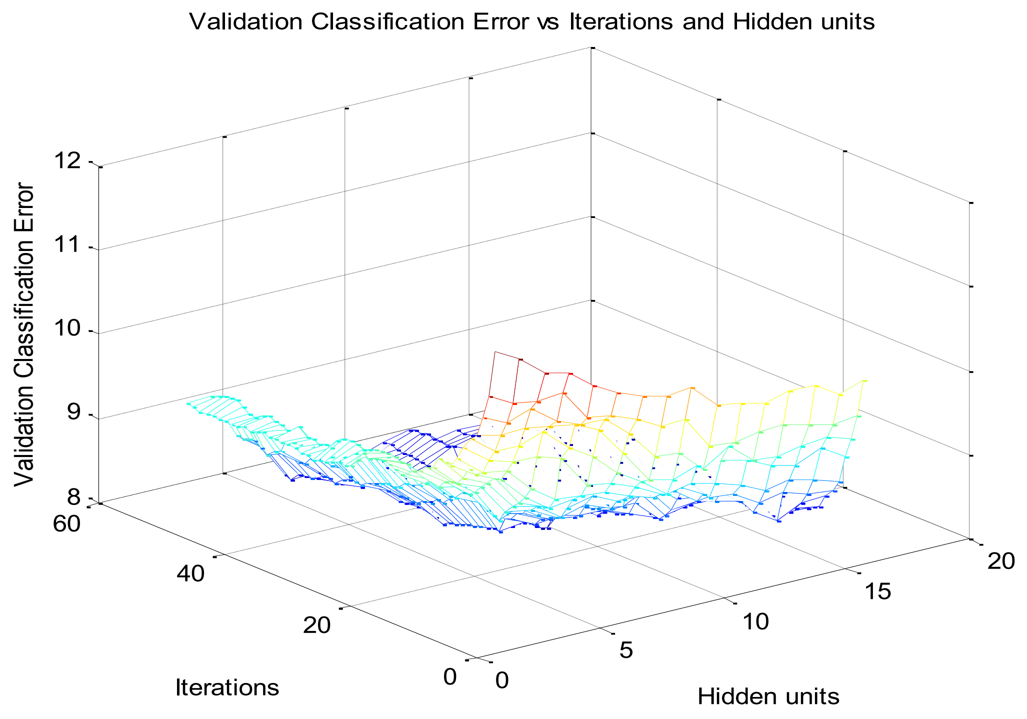

Figure 15 E_v vs N_h and N_{it}, for gongtrn, N_h=15, N_{it}=50

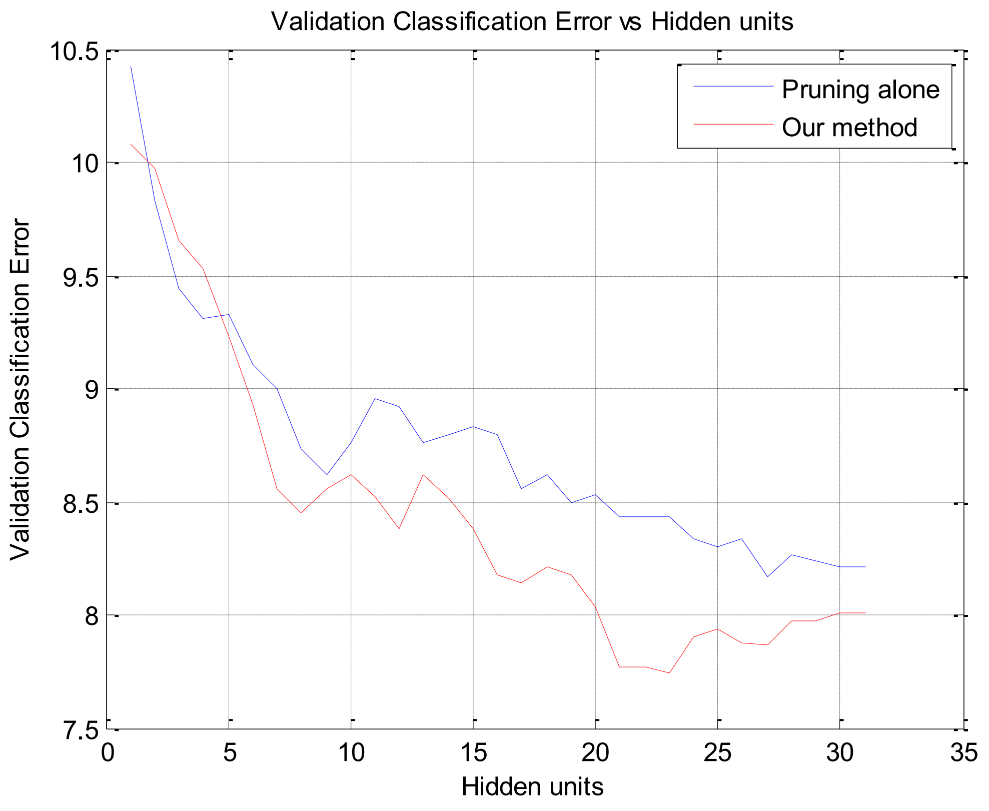

Figure 16 E_v vs N_h, for gongtrn, N_h=30, N_{it}=15

6.2.2 speech_map data set

Data set speech_map has 34 inputs and 39 different classes, so it is very hard to train this file. Figure 17 shows that by doing pruning at each iteration we are optimizing the validation error very well, here N_{it}=25 and N_h=40. Figure 18 shows the 3-D contour. Comparing it with pruning alone shows that, the best network size is 36, as after that classification error increases again, here N_h=40 and N_{it}=20.

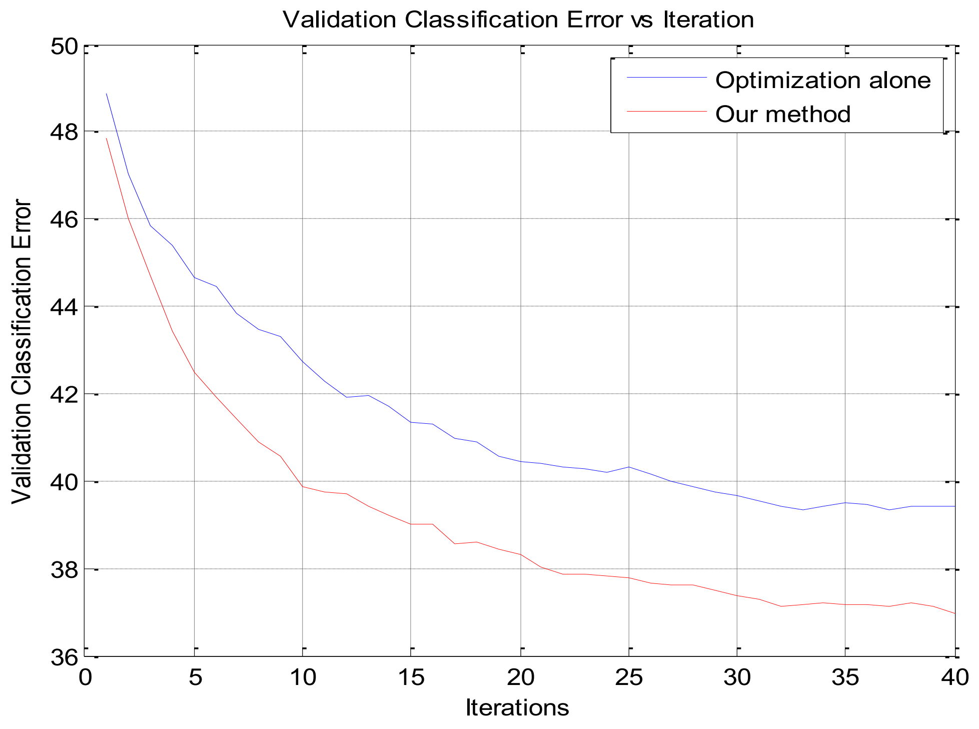

Figure 17 E_v vs N_{it}, for speech_map, N_{it}=25, N_h=40

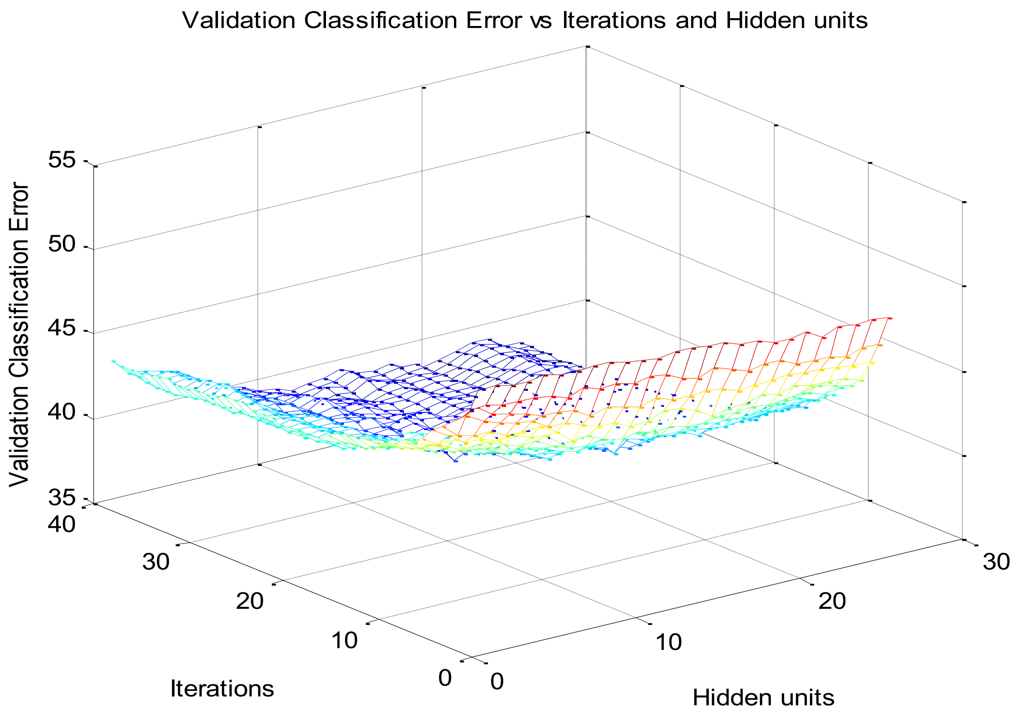

Figure 18 E_v vs N_h and N_{it}, for speech_map, N_{it}=25, N_h=40

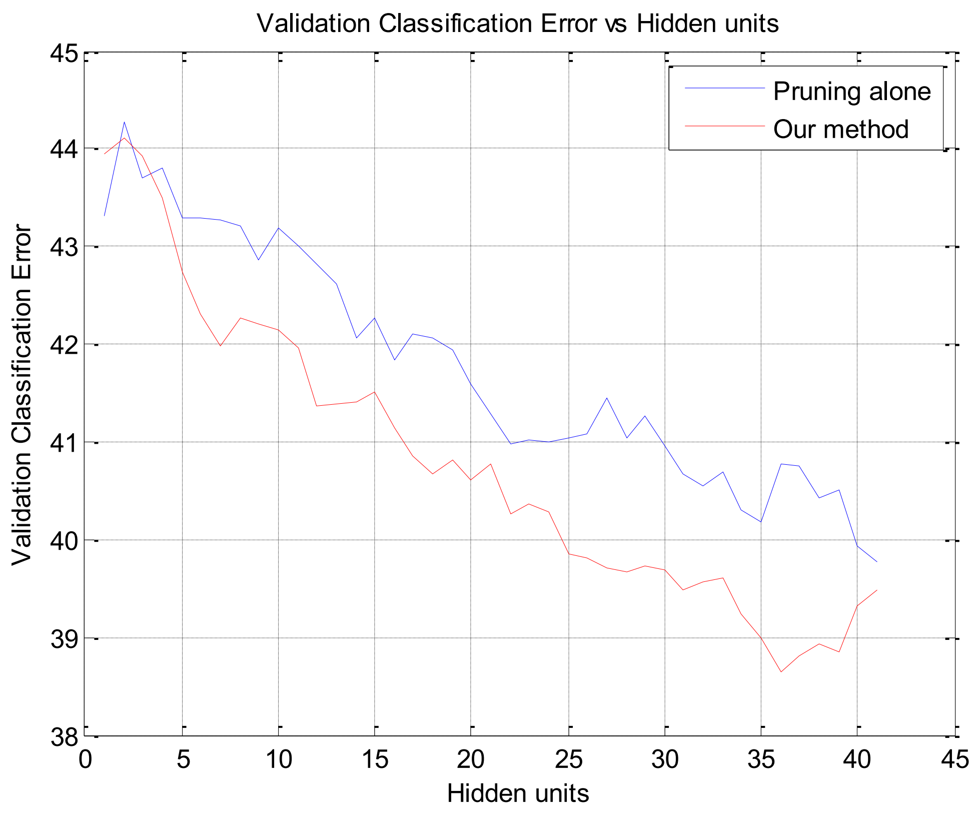

Figure 19 E_v vs N_h, for speech_map, N_{it}=20, N_h=40

CHAPTER 7

CONCLUSION

Based on the validation error curves obtain in results from the prescribed different approaches we can make following observations.

- Optimization over training iterations definitely reduces the training time, but is prone to give a large network size. Because the same large network size is used in every iteration.
- In the pruning alone method, after completion of training since we modify the network size, we get a small network as compared to optimization alone method. But, the number of iteration needed for training is much larger because training progresses on the same large network.
- The novel approach shown in this thesis relieves the drawbacks of the above two methods and therefore attains a much better result in terms of network size, training time and validation error.

APPENDIX A

DESCRIPTION OF DATA SETS USED FOR APPROXIMATION AND CLASSIFICATION

Twod - Inversion of surface scattering parameters

This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements.

The training data file contains 1,768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform PDF [58, 59].

Single2 – Inversion of back scattering parameters

This training data file consists of 16 inputs, 3 outputs and 10,000 training patterns, and represents the training set for inversion of surface permittivity, the normalized surface rms roughness, and the surface correlation length found in back scattering models from randomly rough dielectric surfaces. The first 16 inputs represent the simulated back scattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarization. The remaining 8 are various combinations of ratios of the original eight values. These ratios correspond to those used in several empirical retrieval algorithms [60, 61].

Oh7 - Radar Scattering from Bare Soil Surfaces

This data set is given in [62]. 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. The file has 20 inputs, 3 outputs and 15,000 training patterns.

Mattrn – Matrix inversion data

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. the file has 2,000 training patterns.

Gongtrn data set

The raw data consists of images from hand printed numerals collected from 3,000 people by the Internal Revenue Service. We randomly chose 300 characters from each class to generate 3,000 character training data. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numerals.[63]

Speech class data set

The speech samples are first preemphasized and it is converted into frequency domain by taking DFT. Then it is passed through Mel filter banks and the inverse DFT is applied on the output to get Mel-Frequency Cepstrum Coefficients (MFCC). Each of MFCC(n), MFCC(n)-MFCC(n-1) and MFCC(n)-MFCC(n-2) would have 13 features, which results in a total of 39 features. Each class corresponds to a phoneme.

REFERENCES

[1] An efficient piecewise linear network, Rawat,Rohit. The University of Texas at Arlington, 2009.

[2] R. C. Odom, P. Pavlakos, S. Diocee, S. M. Bailey, D. M. Zander, and J. J. Gillespie, “Shaly sand analysis using density-neutron porosities from a cased-hole pulsed neutron system,” in *SPE Rocky Mountain regional meeting proceedings: Society of Petroleum Engineers*, 1999, pp. 467–476.

[3] A. Khotanzad, M. H. Davis, A. Abaye, and D. J. Maratukulam, “An artificial neural network hourly temperature forecaster with applications in load forecasting,” IEEE *Transactions on Power Systems*, vol. 11, no. 2, pp. 870–876, May 1996.

[4] S. Marinai, M. Gori, and G. Soda, “Artificial neural networks for document analysis and recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 1, pp. 23–35, 2005.

[5] R. B. J. Kamruzzaman, R. A. Sarker, Artificial Neural Networks: Applications in Finance and Manufacturing. Idea Group Inc (IGI), 2006.

[6] L. Wang and X. Fu, Data Mining With Computational Intelligence. Springer- Verlag, 2005.

[7] G. Edwards and J. P. Tate, “Target recognition and classification using neural networks,” in *Proceedings of MILCOM 2002*, vol. 2, Oct 2002, pp. 1439–1442.

[8] E. F. M. Filho and A. C. P. L. de Carvalho, “Target recognition using evolutionary neural networks,” in *Proceedings of V^{th} Brazilian Symposium on Neural Networks*, 1998, Dec 1998, pp. 226–231.

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

[10] M. T.Manry, R. Shoults, and J. Naccarino, “An automated system for developing neural network short term load forecasters,” in *Proceedings of the 58th American Power Conference*, Apr 1996, pp. 237–241.

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

[12] Y. LeCun, B. Boser, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, pp. 541–551, 1989.

[13] M. T. Manry, C.-H. Hsieh, and H. Chandrasekaran, “Near-optimal flight load synthesis using neural nets,” in Neural Networks for Signal Processing IX, 1999. *Proceedings of the 1999 IEEE Signal Processing Society Workshop*, 1999, pp. 535–544.

[14] P. Hong, Z. Wen, and T. S. Huang, “Real-time speech-driven face animation with expressions using neural networks,” *IEEE Transaction on Neural Networks*, vol. 13, no. 4, pp. 916–927, July 2002.

[15] P. Muneesawant and L. Guan, “Automatic machine interaction for content-based image retrieval using a self organizing tree map structure,” *IEEE Transaction on Neural Networks*, vol. 13, no. 4, pp. 821–834, July 2002.

[16] I. Lapidot, H. Gunterman, and A. Cohen, “Unsupervised speaker recognition based on competition between self-organizing maps,” *IEEE Transaction on Neural Networks*, vol. 13, no. 4, pp. 877–887, July 2002.

[17] M. Attik, L. Bougrain and F. Alexandre. (2005, Neural network topology optimization. *Lecture Notes in Computer Science 3697*pp. 53.

[18] W. C. Carpenter and J. F. Barthelemy. (1994, Common misconceptions about neural networks as approximators. *J. Comput. Civ. Eng. 8(3),* pp. 345-358.

[19] A. A. Abdurrab, M. T. Manry, J. Li, S. S. Malalur and R. G. Gore. A piecewise linear network classifier. Presented at *International Joint Conference on Neural Networks*.

[20] T. Cover and P. Hart. (1967, Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory 13(1),* pp. 21-27.

[21] IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 8, NO. 3, MAY 1997 519 An Iterative Pruning Algorithm for Feedforward Neural Networks Giovanna Castellano, Anna Maria Fanelli, *Member, IEEE,* and Marcello Pelillo, *Member, IEEE*

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing— Vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[23] S. Y. Kung and J. N. Hwang, "An algebraic projection analysis for optimal hidden units size and learning rates in backpropagation learning," in *Proc. IEEE Int. Conf. Neural Networks,* San Diego, CA, vol. 1, 1988, pp. 363–370.

[24] D. C. Plaut and G. E. Hinton, "Learning sets of filters using backpropagation," *Comput. Speech Language,* vol. 2, pp. 35–61, 1987.

[25] D. J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-36, pp. 1162–1168, 1988.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature,* vol. 323, pp. 533–536, 1986.

[27] X.-H. Yu, "Can backpropagation error surface not have local minima," *IEEE Trans. Neural Networks,* vol. 3, pp. 1019–1021, 1992.

[28] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computa.,* vol. 1, pp. 151–160, 1989.

[29] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large automatic learning, rule extraction, and generalization," *Complex Syst.,* vol. 1, pp. 877–922, 1987.

[30] Y. Le Cun, "Generalization and network design strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, and L. Steels, Eds. Amsterdam: Elsevier, 1989, pp. 143–155.

[31] Y. Chauvin, "Generalization performance of overtrained backpropagation networks," in *Neural Networks—Proc. EURASIP Wkshp. 1990*, L. B. Almeida and C. J. Wellekens, Eds. Berlin: Springer-Verlag, 1990, pp. 46–55.

[32] G. G. Towell, M. K. Craven, and J. W. Shavlik, "Constructive induction in knowledge-based neural networks," in *Proc. 8th Int. Wkshp. Machine Learning*, L. A. Birnbaum and G. C. Collins, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 213–217.

[33]An integrated growing-pruning method for feedforward network training Pramod L. Narasimhaa,_, Walter H. Delashmit, Michael T. Manry, Jiang Li, Francisco Maldonado

[34] K. H. M. Stinchcombe and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feed-forward networks," *Neural Networks*, vol. 3, no. 5, pp. 551 – 560, 1990.

[35] D.W. Ruck, “The multi-layer perceptron as an approximation to a bayes optimal discriminant function,” *IEEE Transactions on Neural Networks*, vol. 1, no. 4, 1990.

[36] Q. Yu, S. J. Apollo, and M. T. Manry, “Map estimation and the multi-layer perceptron,” *Proceedings of the 1993 IEEE Workshop on Neural Networks for Signal Processing*, pp. 30–39, September 1993.

[37] R. G. Gore, J. Li, M. T. Manry, L. M. Liu, C. Yu and J. Wei, "Iterative Design of Neural Network Classifiers Through Regression", *International Journal on Artificial Intelligence Tools*, Vol 14, Issues 1&2, 2005.

[38] M. R. Hestenes and E. Steifel, “Methods of conjugate gradients for solving linear systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[39] F. J. Maldonado, M. T. Manry, and T.-H. Kim, “Finding optimal neural network basis function subsets using the schmidt procedure,” in *Proceedings of the International Joint Conference on Neural Networks*, ser. 20-24, vol. 1, July 2003, pp. 444 – 449.

[40] A. J. Shepherd, Second-Order Methods for Neural Networks, ser. *Perspectives in Neural Computing*. Springer, 1997.

[41]Convergence of a Batch Training Algorithm for Feed-forward Networks Sanjeev S. Malalur[1], Changhua Yu[2] and Michael T. Manry[3]

[42] An efficient hidden layer training method for the multilayer perceptron Changhua Yua,_, Michael T. Manry, Jiang Lic, Pramod Lakshmi Narasimha

[43] S. McLoone and G. Irwin, “A variable memory Quasi-Newton training algorithm,” *Neural Processing Letters*, vol. 9, pp. 77-89, 1999.

[44]Sanjeev 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

[45] Finding Optimal Neural Network Basis Function Subsets Using the Schmidt Procedure. F. J. Maldonado, M. T. Manry, and Tae-Hoon Kim

[46] Fast Generation of a Sequence of Trained and Validated Feed-Forward Networks Pramod L. Narasimha, Walter Delashmit, Michael Manry, Jiang Li and Francisco Maldonado

[47] S. Raudys, Statistical and Neural Classifiers: An integrated approach to design, Springer-Verlag, 2001.

[48] W.E. Weideman, M.T. Manry, and H.C. Yau, "A comparison of a nearest neighbor classifier and a neural network for numeric handprint character recognition," *Proceedings of IJCNN'89*, vol. I, pp. I-117 to I-120, Washington D.C., June 1989.

[49] S-C Huang and Y-F Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Transactions on Neural Networks*, January 1991, pp. 47-55.

[50] J. Sietsma and R.J.F. Dow, "Neural net pruning-Why and how ?," in *Proceedings of IJCNN'88*, Vol. I, pp. 325-332.

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

[52] Mu-Song Chen and M.T. Manry, "Basis Vector Analyses of Back-Propagation Neural Networks," *Proceedings of the 34th Midwest Symposium on Circuits and Systems*, Monterey, California, May 1991.

[53] M.T. Manry, X. Guan, S.J. Apollo, L.S. Allen, W.D. Lyle, and W. Gong, "Output weight optimization for the multi-layer perceptron," *Conference Record of the* Twenty-Sixth Annual Asilomar Conference on Signals, Systems, and Computers, Oct. 1992, vol 1, pp. 502-506.

[54] D. S. Levine, *Introduction to Neural and Cognitive Modeling*, Hillsdale, NJ: Lawrence Erlbaum Assoc., 1991.

[55]Nils J. Nilsson, *The Mathematical Foundations of Learning Machines*, Morgan Kaufmann Publishers, San Mateo, California, 1990.

[56] T. Kohonen, "An introduction to neural computing," *Neural Networks*, Vol. 1, 1988, pp.3-16.

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

[58] M. S. Dawson, A. K. Fung and M. T. Manry. (1993, Surface parameter retrieval using fast learning neural networks. *Remote Sens. Rev. 7(1),* pp. 1-18.

[59] M. S. Dawson, J. Olvera, A. K. Fung and M. T. Manry. Inversion of surface parameters using fast learning neural networks. *Presented at IGARSS'92*.

[60] A. K. Fung, Z. Li and K. S. Chen. (1992, Backscattering from a randomly rough dielectric surface. *IEEE Trans. Geosci. Remote Sens. 30(2),* pp. 356-369.

[61] A. K. Fung. (1994), Microwave scattering and emission models and their applications. *Norwood, MA: Artech House, 1994.*

[62] Y. Oh, K. Sarabandi and F. T. Ulaby. (1992, An empirical model and an inversion technique for radar scattering from bare soil surfaces. *IEEE Trans. Geosci. Remote Sens. 30(2),* pp. 370-381.

[63] W. Gong, H. C. Yau, and M. T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," *Progress in Neural Networks*, vol. 2, 1994, pp. 253-269.

BIOGRAPHICAL INFORMATION

Jignesh Patel was born in 1989. He did his Bachelor of Engineering from Dharmsinh Desai University, Nadiad, Gujarat in Electronics and Communication Engineering in May 2010. He obtained his Master of Science degree in Electrical Engineering from the University of Texas at Arlington in May 2012.