Pruning Artificial Neural Network using Back Propagation Training with Application to Pattern Classification

Sahil Singla

Department of Computer Engineering

Delhi Technological University

Shahbad Daulatpur, Bawana Road, New Delhi

Pruning Artificial Neural Network using Back Propagation Training with Application to Pattern Classification

Thesis submitted in partial fulfillment for the degree of

Master of Technology

in

Computer Science And Engineering

Submitted by

Sahil Singla (2K12/CSE/18)

Under the guidance of

Mr. Manoj Sethi

Department of Computer Engineering

Delhi Technological University

Shahbad Daulatpur, Bawana Road, New Delhi

Department of Computer Engineering
Delhi Technological University
Delhi-110042, Delhi, India

Certificate

This is to certify that the work in the thesis entitled ***Pruning Artificial Neural Network using Back Propagation Training with Application to Pattern Classification*** by **Sahil Singla** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science in the department of Computer Engineering, Delhi Technological University, Delhi.

Place: Delhi
Date: July 2, 2014

Mr. Manoj Sethi
Faculty, CoE Department
Delhi Technological University

ACKNOWLEDGEMENT

I would like to express my hearty thanks to my guide, Mr. Manoj Sethi, Department of Computer Engineering, Delhi Technological University for his guidance, support, and encouragement throughout the thesis work. Without his knowledge, endless efforts, patience, and answers to my numerous questions, this thesis would have never been possible. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research.

I extend my thanks to our HOD, Prof. Rajiv Kapoor for his valuable advices and encouragement. I would also acknowledge the resources I used at the CoE department.

Finally, I would like to thank my parents and colleagues for their support and motivation to complete this thesis work.

Sahil Singla

2K12/CSE/18

ABSTRACT

Artificial Neural Network is a **machine** which models the way in which human brain perform its task of learning from new environment. Neural Network has been extensively used for task of learning patterns from training samples for purpose of classification. Pattern Classification involves mapping the given set of input features to two or more classes. Formerly, completely connected neural network with Back Propagation learning was used to predict membership of data instance to particular class. But same task can be done by using the neural network of smaller size and less complexity. This work aims to propose a new paradigm to prune an artificial neural network using error back propagation learning algorithm. In this work, neural network is trained partially and redundant weights are removed. There are two issues involved in the method. First is when the network should be pruned? Second is heuristic to measure importance of weights in network. In particular performance of pruned neural network is compared with its completely connected version using four different datasets and significant increase in learning speed is observed, while maintaining similar generalization ability.

Keywords

Mean Square Error (MSE), Back Propagation (BP), partially connected neural network (PCNN), feed forward neural network.

Contents

List of figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Artificial neural networks are prototyped after human brain which processes nerve signals from input nerve cells and propagate them to other neurons. In complete biological nervous system, all the nerve cells may not be completely connected. This calls for the need of multilayered neural network structure which has only subset of all the relevant interconnections among its neurons. Such networks are known as partially connected neural networks (PCNNs). The aim to construct PCNNs is to have a reduced neural network topology equivalent or better in performance than the completely connected model. A lesser number of connections in the network can help improve generalization and reduce the network complexity, hardware, and storage requirements, and training and testing time [1].

A host of methods for dealing with PCNNs have been proposed. Some of methods are: the ontogenic, the non-ontogenic, and the hybrid methods.
In onto genic methods the topology of the neural network is modified during the learning phase whereas in non-onto genic methods topology of the network is defined prior to the learning phase and remains unchanged during the learning process Hybrid methods are combinations of neural networks with other artificial intelligence (AI) techniques. These AI techniques include: symbolic knowledge and genetic programming [1].
The ontogenic methods based on the back propagation algorithm can be classified into three groups: growing, pruning, and the methods which combine both growing and pruning techniques [1].

Neural network pruning is defined as a process of cropping the initial assumed neural network architecture. One of the main goals of pruning algorithm is to arrive at optimal structure that can generalize well with lesser complexity. Pruning methods start by training a neural network with a topology bigger than needed. Due to it being oversized, the initial network will be less sensitive to initial conditions such as weight initialization and learning parameters. Thus, the initial network is expected to learn reasonably fast. After this, the network is trimmed until the smallest topology that correctly maps the data is found. Growing methods start with a small topology which increases until the neural network achieves a good level of performance for the given method.

In this work, we have introduced a paradigm by which we use traditional back propagation based learning method to prune the neural network. Instead of training network completely, we leave it partially trained and remove the unnecessary weights. We try to figure out redundant weights before completion of training. This work measures the effectiveness of proposed method against the completely connected feed forward network trained using back propagation.

1.2 Organization of Thesis

The thesis is organized as follows: Chapter 2 introduces the literature survey, chapter 3 describes techniques of pruning neural network, chapter 4 discusses proposed work, and chapter 5 describes implementation and results.

Chapter 2

Literature Survey

Pruning is defined as a cropping the size of network within the assumed initial architecture [2]. Variety of methods have been proposed in past to prune artificial neural networks. Earlier methods were based on either the sensitivity of weight or penalty based methods. Sensitivity based methods modify the trained network i.e. network is trained, sensitivity of weights are calculated and less sensitive weights are removed. Sensitivity determines the importance of weight in network. Several heuristic used to determine the sensitivity of the network.

Select the weight with smallest magnitude to remove, because smaller weights contribute less the output and thus are less sensitive for network. These are magnitude based pruning (MBP) methods [13].

E.D. Karnin (1990) proposes a method which is based on sensitivity of cost function on inclusion / exclusion of each weight. Thus his task is to find weight on removal of which there is minimum increase in error. He use shadow array to keep track of incremental change in weights of neural network. Array is then sorted and least sensitive weight is deleted [4].

Le Cun et al. measured saliency of weight by performing second derivative of the error with respect to weight. This method was optimal brain damage (OBD). It prunes iteratively on a well trained network to a reasonable level, compute ’saliencies’, delete low ’saliency’ weights and resume training [5].

Penalty based methods add penalty term to the cost function. Thus minimizing the cost function would drive useless weights to zero [6]. Weigend et al. introduces penalty term based on complexity of network as function of weights relative to constant w_o. Role of this additional penalty term is controlled by parameter lambda. Very large value of lambda makes penalty term strong and thus weights are forced to become zero [7]. These methods are also called weight decay methods.

Huynh and Setiono [8] propose cross validation method. In cross validation method the whole dataset is divided into two parts that is training set and cross validation set. The pruning criterion is still based on the magnitude of each weight but a validation step is additionally used to test the pruned network. If the pruned network outperforms the one before pruning, then the pruned network is accepted and the pruning process can be carried out further. Otherwise the network is restored to the size before the current pruning step.

S. belciug et al. proposes yet another method which uses back propagation training to create partially connected neural network. It deletes those weights which don't pass certain threshold value after training [9].

Real-world applications prefer simpler and more efficient methods. But most of methods are very complex to implement and less efficient also. For example the main drawback of the OBD is its relatively low computational efficiency. MBP methods often remove important weights of the network as they assume that small weights are irrelevant [13].

Chapter 3

Pruning Neural Network

3.1 General Concept

While building an artificial neural network the designer comes across the problem of choosing right architecture of neural network for task to accomplish. A problem which can be solved by network of given size can be solved by network of larger size also [4]. But using smaller size network offers several benefits:

The cost of computation grows linearly with number of connections or weights. Smaller network is thus efficient in forward computation and learning both.

Neural network training is done using finite set of training examples; larger artificial neural network will tend to memorize the pattern, thus leading to poor generalization [4].

Several researchers have thus proposed work to reduce the size of given neural network. Such algorithms are called as artificial neural network pruning algorithms.

The trimmed network is of smaller size and is likely to give higher accuracy than before its trimming. Researchers have suggested many pruning algorithms for optimizing the architecture of neural networks. Based on the techniques used for pruning, the pruning methods can be classified as penalty term methods, cross validation methods, magnitude based methods and sensitivity based methods [2].

3.2 Different Pruning Methods

3.2.1 Sensitivity Based Methods

The general idea of these methods is to train a network in performing a given task and then to compute how important is the existence of a connection or node. Then, the least important connection nodes are removed and the remaining network is retrained. In general, the sensitivity measurement does not interfere with training and requires an extra amount of computational effort. The key issue in the implementation of these techniques is finding a way to measure how sensitive is the solution to the removal of a connection or a node. Early approaches attempt to remove a connection by evaluating the change in the network's output error. If the error increases too much, then the weight must be restored back again. More sophisticated approaches evaluate the change in error for all the connections on training data and then remove the one connection which produces the least error increment. Both approaches takes lot of time to execute in practice.

1. Optimal Brain Damage

Le Cun et al. have proposed the optimal brain damage (OBD) method that approximates the measure of saliency of a weight by estimating the second derivative of the network output error with respect to that weight. In this method pruning is carried out iteratively on a well trained network to a reasonable level, compute saliencies, delete low saliency weights and resume training. Main steps are as follow:

1. Create a neural network with larger architecture.

2. Train the network until some stopping criterion is satisfied.

3. After training is done, compute the second derivatives for each of the weights.

4. Evaluate the saliencies $h_{i\,i}W_i$ for each weight and sort the weights by saliency.

5. Delete some of the low-saliency weights.

6. Go to 2 and repeat until some overall stopping criterion is reached.

This approach to weight elimination has been termed *optimal brain damage* [5].

2. Shadow Array Based Method

E.D. Karnin [4] measures the sensitivity of error function w.r.t. to each connection and removes the weight with low sensitivity. Sensitivity of weight w_{ij} is given as

$S_{ij} = - (E (w^f)) – E(0)) w^f / w^f - 0$

Rather than actually removing weight and calculating error they calculate S by summing changes in weight during training process.

After training each weight has an estimated sensitivity and lowest sensitivity weight can be deleted.

3.2.2. Penalty Based Methods

The method modifies the error function so that back propagation function prunes the network by driving weights to zero. Weights may also be removed when they fall below certain threshold.

Weigend et al. [7] minimizes the following cost function:

$\sum(t - o)^2 + \lambda \sum(w_i^2/w_o^2)/(1+ w_i^2/w_o^2)$

Where T is set of all the training patterns and C is set of all connections.

Second term represent the complexity of network as function of weight magnitude relative to constant w_o.

When lambda is large it is similar to the weight decay problems, When lambda is infinite it drives weight to zero. Thus it requires some tuning for this parameter and depends on problem at hand.

3.2.3 Magnitude Based Methods

Magnitude based methods (MBP) assumes that small magnitude weights are less important than large magnitudes [13]. Thus they use magnitude of weight |w| as measure of saliency of weights. There is not much theoretical background behind these methods. These methods are more or less driven by assumptions and perform poorly in practice.

MBP based methods sometimes remove those weights which are critical in performance of neural network as they think small weight are useless.

3.2.4 Cross Validation Based Methods

In addition to training and testing set we have third set called as cross validation set. In this method prior to training process whole dataset is divided to three sets- training, cross validation, testing. Cross validation set is used to check the performance of current state of pruned neural network in comparison to state before this step. Cross validation methods follow steps as shown below:

- Train the network to achieve desired accuracy.
- Prune the trained network on basis of magnitude of weights.
- Check the performance of pruned network and compare with previous unpruned network.
- If performance increases save pruned network for further steps otherwise discard pruned network.
- Iterate over these steps.

  This method is proposed by Huynh and Setiono [8].

Chapter 4

Proposed Work

4.1 Introduction

In this chapter we have discussed traditional feed forward neural network with back propagation learning and proposed method.

4.2 Feed Forward Neural Network With Back Propagation Learning.

Feed forward neural network is neural network which is completely connected. It means every neuron in each layer is connected to every neuron in next adjacent layer. It mainly uses back propagation learning algorithm for training purpose.

In feed forward neural network back-propagation learning algorithm has two phases.

First is feed forward computation and second is backward error propagation. In feed forward computation, training example is presented to input layer of neural network. This input pattern is then passed from one layer to another layer. Finally output is generated by output layer when pattern reaches output layer. In backward error propagation, the output pattern is compared with desired pattern. If output pattern is different from desired pattern, difference between them is calculated and error is propagated backwards [23].

A neuron determines its output by first computing net weighted input as before:

$X = \sum_{i=1}^{n} x_i w_i$

where n is the number of inputs, and x_i is value of input, w_i is weight associated with that input and Θ is the threshold applied to the neuron. Next, this weighted input value is passed through the activation function. However neurons in the back-propagation network use a sigmoid activation function:

Y= sigmoid(X)

Where sigmoid(x) is $1/ (1+e^{-x})$

The derivative of this function is easy to compute. It also guarantees that the neuron output is bounded between 0 and 1.

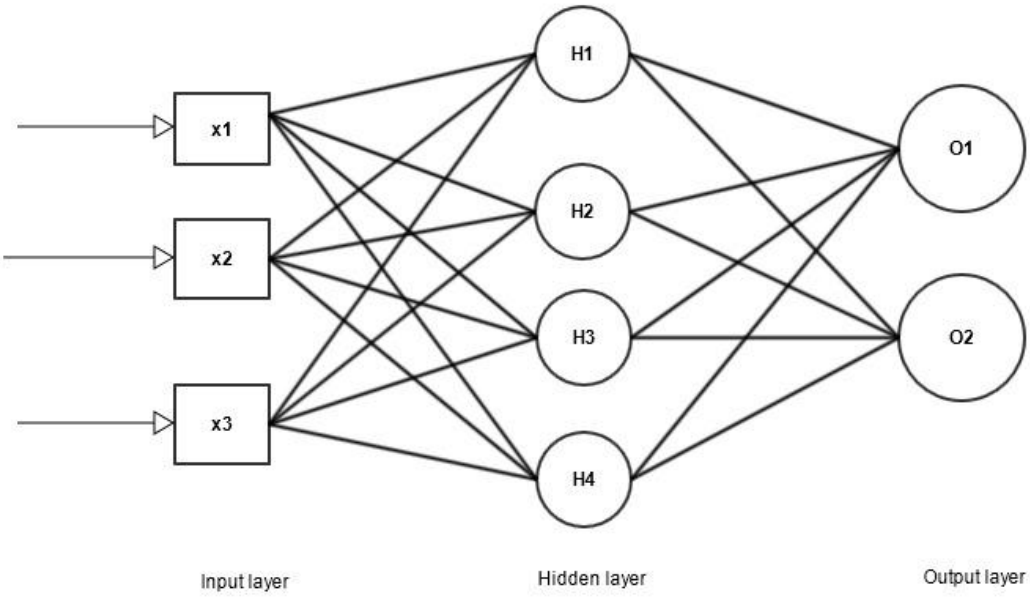


Fig. 4.1 Back propagation in three layer feed forward neural network.

Detailed back propagation learning algorithm for feed forward is described below.

4.2.1 Algorithm

Main Steps of the Algorithm:

Create a feed-forward network with n_{input}, inputs, m_{hidden} hidden units, and n_{output} output units.

- Initialize all network weights to small random numbers in uniform range [-e_{init}, +e_{init}].

Where $e_{init} = \frac{\sqrt{6}}{\sqrt{Lin}+\sqrt{Lout}}$

- Until the termination condition is met, do

For each *(x, y)* in *training examples,* do

Propagate the input forward through the network:

1, Input the instance (x,y) to the network and compute output for every node in network.

Propagate the errors backward through the network:

2. For each network output unit t, calculate its error term δt

$$\delta = ot(1 - ot)(yt - ot)$$

3. For each neuron u in hidden layer *h,* calculate its error term δu

$$\delta u = \text{ou}(1 - ou) \sum_{for\ all\ output\ t} wtu\delta t$$

4. Update each network weight *wi j*

Wi j = Wi j + $\Delta wi\ j$

Where Wi j= $\alpha * \delta j *$ Xi j

Flowchart for above algorithm is shown in figure 4.2.

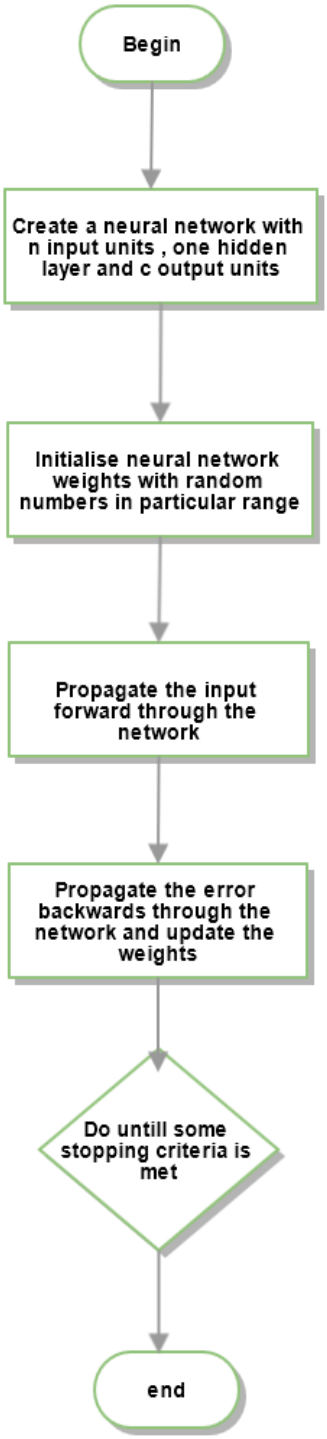


Fig. 4.2 Flowchart for BP learning algorithm for feed forward neural networks.

4.2.2 Stopping Criteria

The training process is repeated until the Mean square error is less than 0.001 or predefined number of epochs is done.

4.3 Proposed Method

4.3.1 Motivation

Most of the work done in area of neural network pruning is based on either training the neural network completely and then pruning the connections based on sensitivity of weights or pruning the neural network while training is going on. But training the network to get best accuracy and then determining the redundant weights is time consuming. Here, we propose the algorithm whereby we try to estimate sensitivity of the weights even before network is trained completely. Resultant neural network constructed using proposed method is SCNN as shown in fig. 4.3. SCNN has only fewer connections than completely connected neural network.

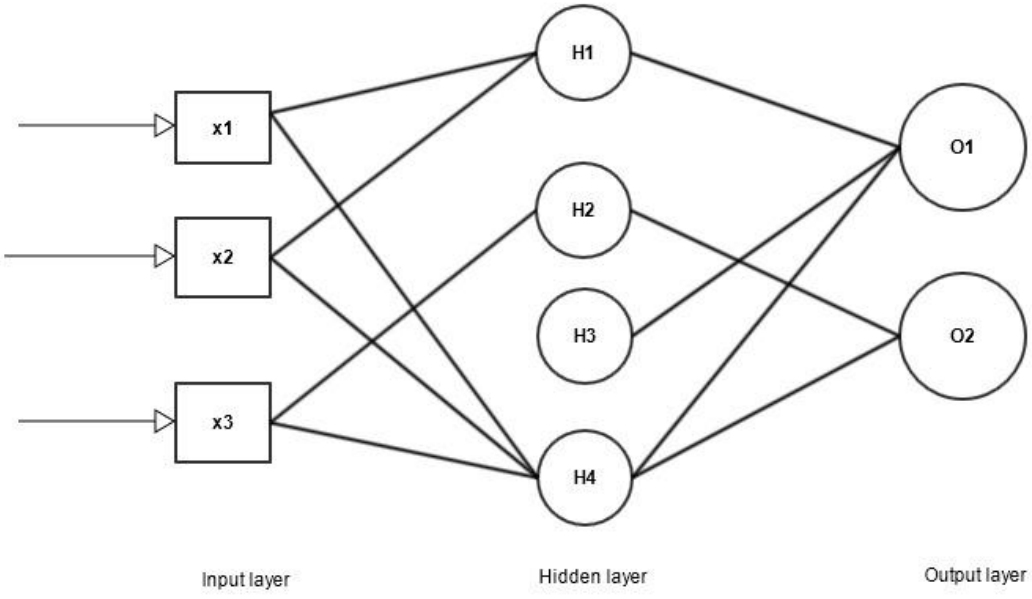


Fig. 4.3 SCNN using Proposed Method.

4.3.2 Design Issues

There are two major design issues in this proposed work.

- First is after how many epochs of training should network be considered for pruning?
- Other is, heuristic to find notion of less useful weights i.e. determining the criteria to find redundant synaptic in network.

In order to address the first issue, we have taken in account notion of how many features are learnt by network before we prune it. As neural network learn features or pattern in training samples through weight modification, it is important not to delete useful weights. One of the measures of learning is change in magnitude of the weights. Weights that don't undergo much change in their value are slow at learning, thus are good candidate for deletion. Thus we conclude that network be considered for pruning only it has learnt sufficient features. Measure of learning by network is Mean squared error. If MSE is less than some threshold value, it means network has learnt sufficient features to be considered for pruning. Value of the threshold depends on slope of graph b/w No of epochs and MSE while performing back propagation training. If slope is more, threshold should be more. If slope is less, threshold should be less. Reason behind this idea is, more slope means that network is learning very fast and threshold should be good enough so that pruning is done early. Whereas fewer slopes means network is learning very slow, thus pruning can be performed after considerable no of epochs.

Heuristic to determine less useful weight is based on change in magnitude of weight from initial weight to weight after network is considered for pruning. If change in weight is less than some threshold value, we set the corresponding weight to zero. If all the incoming connection to a hidden node is zero, then we can safely remove that node also.

This proposed algorithm works well for problem where number of input features is very large. The flowchart representation of proposed method is shown in fig 4.4.

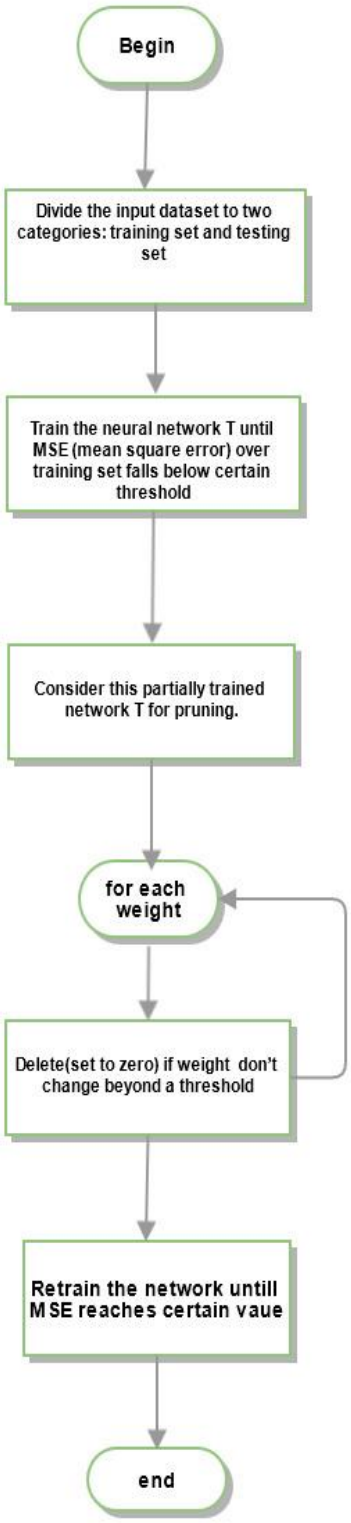


Fig. 4.4 Flowchart for Proposed method.

4.3.3 Algorithm

Main Steps of the Algorithm

INPUT The Multilayer feed forward network T with no. of layers L and n number of input features. Each layer l (1< l <L) contains x_l number of nodes. It comprises of an output layer which contains number of nodes equal to number of classes.

BEGIN

1. Divide the input dataset to two categories: training set and testing set. Training set is used to train the network. Testing set is used to measure the accuracy of trained network.
2. Train the neural network T until MSE (mean square error) over training set falls below certain threshold θ. Threshold is chosen such that network has learnt important features.
3. Consider this partially trained network T for pruning.
4. For all weights w_{ij} in network T

   4.1 Delete(set to zero) those weights which don't change beyond a threshold $\boldsymbol{\alpha}$ i.e. difference b/w initial weights and weights after partial training is less than some predetermined value.

5. Retrain this pruned network until mean squared error falls down to 0.001.
6. Determine the accuracy of trained network using test set.

OUTPUT:

The pruned multilayer feed forward neural network T.

Chapter 5

Implementation and Results

5.1 Datasets

In this work we have used four different datasets.

Description of the datasets is as follows:

1. **Pen based handwritten digit dataset:** It contains 7494 sample of handwritten digits each of which is characterized by 16 features ranging from 0-100. Each of samples belongs to one of ten classes (0-9).

   No. of Features: 16

   No of Classes: 10

   No. of Examples: 7494

   This dataset is obtained from UCI Machine learning Repository *https://archive.ics.uci.edu/ml/datasets/PenBased+Recognition+of+Handwritten+Digits.*

2. **SEMEION Handwritten Digit Data Set:** 1593 handwritten digits from around 80 persons were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values. Then each pixel of each image was scaled into a Boolean (1/0) value using a fixed threshold. The dataset was created by Tactile Srl, Brescia, Italy and donated in 1994 to Semeion Research Center of Sciences of Communication, Rome, Italy for machine learning research.

This dataset is obtained from UCI Machine learning repository *https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit*

No of features: 256

No of classes: 10

No of examples: 1593

3. **Iris dataset**: Irises are classified into three classes: setosa, versicolour and verginica. Each category has 50 patterns and each pattern possesses four attributes namely sepal length, sepal width, petal length and petal width.
   This dataset is obtained from UCI Machine learning repository *https://archive.ics.uci.edu/ml/datasets/Iris*

   No of features: 4

   No of classes: 3

   No of examples: 150

4. **Wisconsin Breast cancer dataset:** Wisconsin cancer dataset consists of 699 samples. It is used to diagnose breast cancer as either benign or malignant. Each pattern consists of 9 real value attributes as an input vector and two classes as an output vector. Out of 699 samples 458 are benign and 241 are malignant patterns.
   This dataset is obtained from UCI Machine learning repository *https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29*

   No of features: 9

   No of classes: 2

   No of examples: 699

5.2 Data Preprocessing

Data pre-processing is a crucial step before engaging it to pattern extraction process. Data-gathering methods are often roughly handled, resulting in out-of-range values (e.g., Age: 0), impossible data combinations (e.g., Living entity: Human, Hands: Four), missing values, etc. Analyzing data that has not been carefully checked for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis.

Data pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc. The product of data pre-processing is the final training set.

However, we have taken account of normalization only because all other processes have been taken care of by the source UCI repository itself. To normalize a set of data, the original data range is mapped into another scale. Normalization is needed to pre-process data so as to increase the efficiency of algorithm i.e. it helps to bring the data closer to the requirements of the algorithms. Variables can be normalized (to unit zero mean and unit variable, or to the interval [0, 1]), data elements can be normalized (when all their attributes have the same units). Following are the steps of normalizing data sets:

. Find out the Minimum (Min) and Maximum (Max) of original datasets.

. Decide the Minimum (MinN) and Maximum (MaxN) for normalized scale.

. Consider a number (X) from the data set.

. The Normalized value for the number(X) is given by the formula:

MinN+ (X-Min)*(MaxN-MinN)/ (Max-Min)

5.3 Implementation

We have implemented training and testing of feed forward network with BP and proposed method in MATLAB R2011a. A snapshot of the implementation is shown in appendix B.

For purpose of comparative analysis we set same initial weights and architecture for both methods. Network is trained until it converges to predetermined MSE value or reaches fixed number of epochs whichever is earlier. This experiment is performed for each dataset for ten times, each time dividing training and testing set randomly.

5.3.1 Feed Forward Neural network using back propagation algorithm

a) Pen based handwritten digit dataset:

Architecture of 16-25-10 is used i.e. input feature is 16-D vector with one hidden layer having 25 hidden units and one output layer having 10 output units. Training is stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs: 100

Learning Rate: **0.1**

Training pattern: **4996**

Testing pattern: **2498**

b) SEMEION handwritten digit dataset:

Architecture of 256-25-10 is used i.e. input feature is 256-D vector with one hidden layer having 25 hidden units and one output layer having 10 output units. Training is stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs: 100

Learning Rate: 0.1

Training Patterns: 1062

Testing Patterns: 531

c) Iris dataset

Architecture of 4-10-3 is used i.e. input feature is 4-D vector with one hidden layer having 10 hidden units and one output layer having 3 output units. Training is stopped whenever MSE falls below 0.001 or 200 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs: 200

Learning Rate: 0.1

Training Patterns: 100

Testing Patterns: 50

a) Wisconsin Breast cancer dataset

Architecture of 4-10-2 is used i.e. input feature is 4-D vector with one hidden layer having 10 hidden units and one output layer having 2 output units. Training is stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs: 100

Learning Rate: 0.1

Training Patterns: 466

Testing Patterns: **233**

5.3.2 Proposed Algorithm

a) Pen based handwritten digit dataset:

Architecture of 16-25-10 is used i.e. input feature is 16-D vector with one hidden layer having 25 hidden units and one output layer having 10 output units. Training is stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs:	**100**
Learning Rate:	**0.1**
Training pattern:	**4996**
Testing pattern:	**2498**
Threshold θ for MSE for pruning	**0.02**
Threshold α for change in weights	**0.05**

b) SEMEION handwritten digit dataset:

Architecture of 256-25-10 is used i.e. input feature is 256-D vector with one hidden layer having 25 hidden units and one output layer having 10 output units. Training is stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs:	**100**

Learning Rate:	**0.1**
Training Patterns:	**1062**
Testing Patterns:	**531**
Threshold θ for MSE for pruning	**0.02**
Threshold α for change in weights	**0.054**

c) Iris dataset

Architecture of 4-10-3 is used i.e. input feature is 4-D vector with one hidden layer having 10 hidden units and one output layer having 3 output units. Training is stopped whenever MSE falls below 0.001 or 200 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs:	**200**
Learning Rate:	**0.1**
Training Patterns:	**100**
Testing Patterns:	**150**
Threshold θ for MSE for pruning	**0.02**
Threshold α for change in weights	**0.024**

d) Wisconsin Breast cancer dataset

Architecture of 4-10-2 is used i.e. input feature is 4-D vector with one hidden layer having 10 hidden units and one output layer having 2 output units. Training is

stopped whenever MSE falls below 0.001 or 100 Epochs are iterated. Following parameters are used:

Maximum Number of Epochs:	**100**
Learning Rate:	**0.1**
Training Patterns:	**466**
Testing Patterns:	**233**
Threshold θ for MSE for pruning	**0.02**
Threshold α for change in weights	**0.05**

5.4 Results

5.4.1 Performance Evaluation

MSE is considered criteria for performance of network; the one with lower MSE has higher accuracy. So we train both proposed algorithm and feed forward neural network with back propagation algorithm unless MSE falls below fixed value 0.01.Thus method which trains the neural network faster is considered better.

Threshold value for MSE to find when to prune can be judged from graph of MSE vs. No. of epochs. When the slope is less in graph, threshold value is also small and vice-versa. This graph between MSE and no. of epochs for feed forward with BP and proposed method is very instructive. It indicates either proposed method takes less no. of epochs for learning or both take same no. of epochs. In later case proposed method reaches lower value of MSE.

Following graphs are obtained for four different datasets

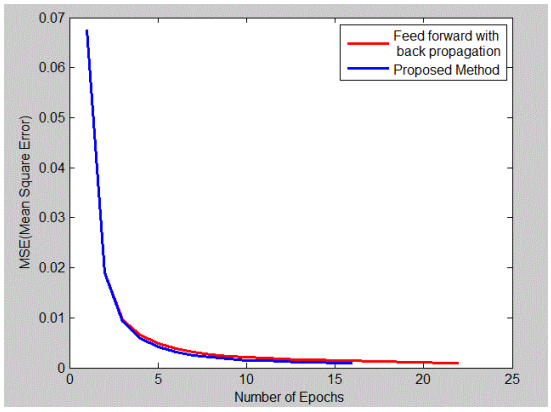


Fig. 5.1 MSE vs. No. of epochs for pen based handwritten digit dataset.

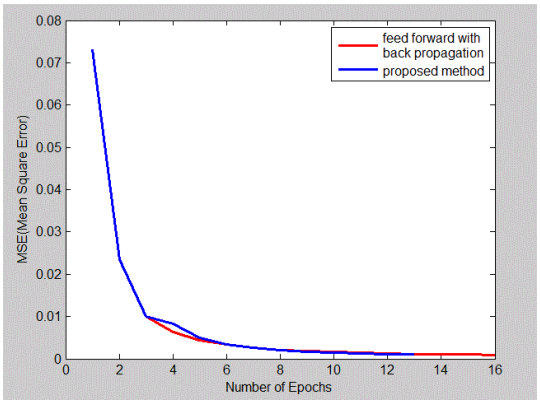


Fig. 5.2 MSE vs. No. of epochs for SEMEION dataset.

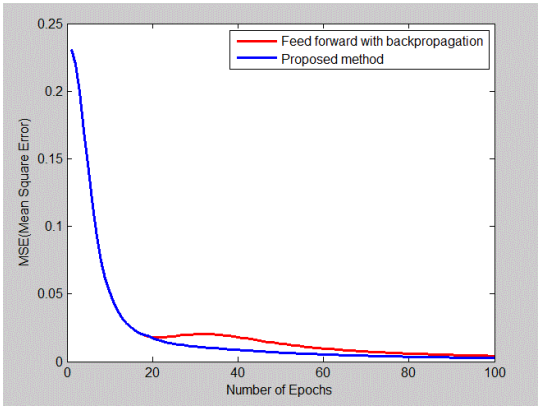


Fig. 5.3 MSE vs. No. of epochs for Iris dataset.

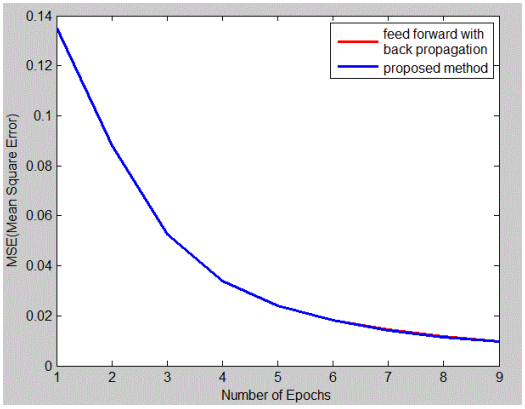


Fig. 5.4 MSE vs. No. of epochs for Wisconsin breast cancer dataset.

Performance is based on how much time does algorithm spends in training the given architecture. Here we input same architecture for both learning methods, initial weights are also same. Table 5.1 and Table 5.2 depict the performance of proposed method and feed forward with BP respectively for four different datasets.

Dataset	**Initial Architecture**	**No. of Epochs(training)**	**Accuracy**	**Training time**	**Testing time**	**No of connections pruned**
Pen based Handwritten digit	16-25-10	47	92.2	65.9	0.52	74
SEMEION	256-25-10	35	91.4	26.52	0.126	2831
Iris	4-10-3	94	94.4	0.96	0.006	22
Cancer	9-10-2	23	96.8	1.15	0.023	5

Table 5.1 Performance of proposed method on different datasets.

Dataset	Architecture	No. of Epochs(training)	Accuracy	Training time	Testing time
Pen based Handwritten digit	16-25-10	48	92.2	68.6	0.96
SEMEION	256-25-10	38	91	29.08	0.27
Iris	4-10-3	98	94.8	1.14	0.01
Cancer	9-10-2	23	96.7	1.26	0.023

Table 5.2 Performance of feed forward with BP learning.

Training time for both methods is compared in Fig.5.5 for each dataset. Results clearly indicate that proposed algorithm is faster at learning than feed forward with BP training. Comparison also depicts that for large datasets like Pen based handwritten digit and SEMEION handwritten digit dataset difference of training time is very significant.

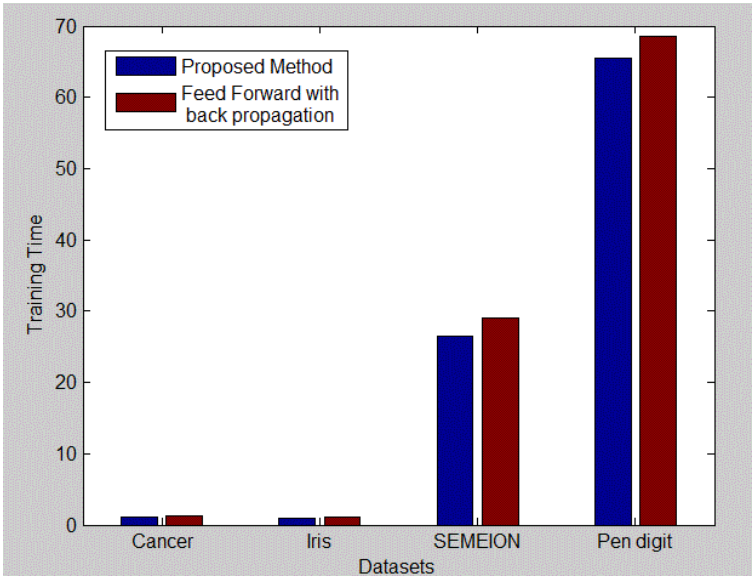


Fig. 5.5 Comparison of methods for training time.

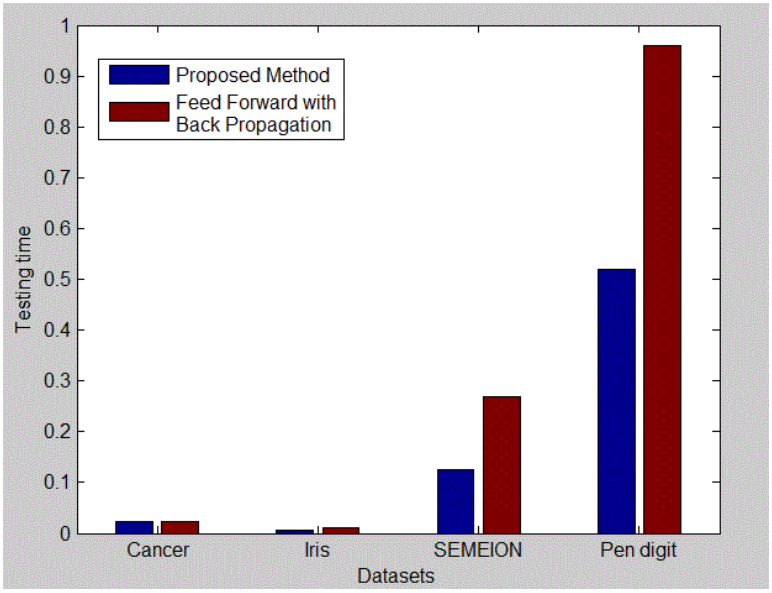


Fig. 5.6 Comparison of methods for testing time.

Forward computation is faster in pruned network because size of pruned network is less. Figure 5.6 shows the comparison of testing time of both methods for each dataset. It clearly shows that testing time for pruned neural network is lesser than completely connected feed forward neural network. Number of pruned connections in large dataset is high in number. Therefore, in large datasets difference in testing time is quite substantial.

Chapter 6

Conclusion and Future Work

Neural network are well researched and established tool for pattern classification problems. More commonly, fully connected neural network with back propagation learning are used. Neural Network is modeled after brain. Pruning such neural network will bring us closer to dream of modeling human brain, which contains several incomplete connections among its neurons.

This paper proposed a pruning algorithm based on back propagation learning. The performance of proposed learning algorithm with traditional Feed forward with BP is compared. In this analysis, four different datasets are used. Results shows that proposed algorithm is faster at learning as compared to Feed forward network with BP, while maintaining similar generalization ability or even sometimes better also. Resultant pruned network also has advantage of faster forward computation as is evident from results.

Future work includes obtaining precise notion for various parameters, which can further improve the algorithm.

References

[1] D. Elizondo and E. Fiesler, “A survey of partially connected neural network”, International Journal of Neural Systems, vol. 8, pp. 535-558, 1997.

[2] M. Gethsiyal Augasta and T. Kathirvalavakumar, “A novel pruning algorithm for optimizing feed forward neural network of classification problems”, Neural Process Letters 34:241–258, DOI 10.1007/s11063-011-9196-7, 2011.

[3] S. Haykin, Neural Networks, a comprehensive foundation (2nd Edition): Prentice Hall, 1999.

[4] E.D Karnin, “A simple procedure for pruning back-propagation trained neural networks”, IEEE Transactions on Neural networks, vol. 1, no. 2, 10.1109/72.80236, June 1990.

[5] Yann Le Cun, JS Denker and SA Solla, “Optimal brain damage”, Advances in Neural Information Processing Systems, vol. 2, Morgan Kaufmann, San Mateo, pp. 598–605, 1990.

[6] R.Reed, “Pruning algorithms—A survey”, IEEE Transactions on Neural Networks, vol. 4, no. 5, 10.1109/72.248452, September 1993.

[7] A. S. Weigend, D. E. Rumelhart and B. A. Huberman, “Generalization by weight elimination with application to forecasting”, In Advances in Neural Information Processing Systems (NIPS) - Natural and Synthetic 3 (San Mateo, California, 1991), R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., Morgan Kaufmann Publishers, pp.875–882.

[8] TQ Huynh, R. Setiono, “ Effective neural network pruning using cross validation”, Proceedings of IEEE International Joint Conference On Neural Networks, vol. 2, pp. 972–977, 10.1109/IJCNN.2005.1555984, 2005.

[9] S. Belciug, E. El-Darzi, “A partially connected neural network-based approach with application to breast cancer detection and recurrence”, Intelligent Systems (IS), pp.191-196, 10.1109/IS.2010.5548358, July2010.

[10] J. Sietsma and R. J. F. Dow, "Neural net pruning- why and how", In Proceedings IEEE International Conference on Neural Networks, vol. 1, pp. 325–333, 10.1109/ICNN.1988.23864, 1988.

[11] Z. Reitermanov´a, "Feedforward neural networks – architecture optimization and knowledge extraction", WDS'08 Proceedings of Contributed Papers, Part I, pp. 159–164, 2008.

[12] Rudy Setiono, "A penalty-function approach for pruning feed forward neural networks", Neural Computation, vol. 9, no. 1, pp. 185-204, 10.1162/neco.1997.9.1.185, Jan 1997.

[13] C. Bishop, "Neural networks for pattern recognition", Clarendon Press, Oxford 1995.

[14] UCI machine learning repository, https://archive.ics.uci.edu/ml/datasets.htm.

[15] S. Belciug, "A statistical comparison between an unsupervised neural network and a partially connected neural network in the detection of breast cancer", Annals of the University of Craiova-Mathematics and Computer Science Series, vol. 37, pp. 71-77, 2010.

[16] Yue-Seng Goh and Eng-Chong Tan, "Pruning neural networks during training by backpropagation", Proceedings of 1994 IEEE Region 10's Ninth Annual International Conference, vol. 2, pp. 805-808, 10.1109/TENCON.1994.369200 ,1994.

[17] Giovanna Castellano and Anna Maria Fanelli, "An iterative pruning algorithm for feed forward neural networks", IEEE Transactions on Neural Networks, vol. 8, no. 3, May 1997.

[18] Fangju Ai, "A new pruning algorithm for Feedforward Neural Networks", Fourth International Workshop on Advanced Computational Intelligence (IWACI), pp. 286-289, 10.1109/IWACI.2011.6160018, 2011.

[19] G. Thimm and E. Fiesler, "Neural network pruning and pruning parameters", Presented on the 1st OnlineWorkshop on Soft Computing, Nagoya, Japan, 1996.

[20] T. Kavzoglu and C. A. O. Vieira, "An analysis of artificial neural network pruning algorithms in relation to land cover classification accuracy", In Proceedings of the Remote Sensing Society Student Conference, Oxford, UK, pp. 53-58, 1998.

[21] V. Arulmozhi, "Classification task by using matlab neural network tool box – A beginner's view", International Journal of Wisdom Based Computing, vol. 1, August 2011.

[22] Christos Stergiou and Dimitrios Siganos, "Neural networks", http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

[23] Michael Negnevitsky, Artificial intelligence: guide to intelligent systems, Second edition.

[24] Saurabh Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture", International Journal of Engineering Trends and Technology, vol. 3, no. 6, 2012.

[25] M. Gethsiyal Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networks - a comparative study", Central European Journal of Computer Science, vol. 3, no. 3, pp. 105-115, DOI: 10.2478/s13537-013-0109-x, 2013.

Appendix A

Source Code

Function Specific to particular datasets

Function for Pen based handwritten dataset

Pen.m

```
function [ ] = pen( Numhidden )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


load('X.mat');
load('Y.mat');

X=X';
Y=Y';

sizeMN= size(X);
sizeTar= size(Y);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);
enitHO= sqrt(6)/(sqrt(numOutput)+sqrt(Numhidden));
enitIH= sqrt(6)/(sqrt(numInputs)+sqrt(Numhidden));

for i=1:10



per= randperm(numExample);

weightHO= (2*rand(Numhidden,numOutput)-1)*enitHO;
weightIH= (2*rand(numInputs,Numhidden)-1)*enitIH;

backPg(X,Y,weightIH,weightHO,per,Numhidden);

prop(X,Y,weightIH,weightHO,per,Numhidden);

fprintf('-----------------------------------------');
end
```

```
end
```

Function for SEMEION based handwritten digit dataset

SEMEION.m

```
function [ ] = mnist( Numhidden )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


load('semeion.mat');


X=X';
Y=Y';

sizeMN= size(X);
sizeTar= size(Y);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);

enitHO= sqrt(6)/(sqrt(numOutput)+sqrt(Numhidden));
enitIH= sqrt(6)/(sqrt(numInputs)+sqrt(Numhidden));

for i=1:10
per= randperm(numExample);

weightHO= (2*rand(Numhidden,numOutput)-1)*enitHO;
weightIH= (2*rand(numInputs,Numhidden)-1)*enitIH;

backPg(X,Y,weightIH,weightHO,per,Numhidden);

prop(X,Y,weightIH,weightHO,per,Numhidden);

fprintf('----------------------------------------');
end
```

```
end
```

Function for Iris dataset

```
function [ ] = iris( Numhidden )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here



load('iris_dataset');

X=irisInputs;
Y=irisTargets;

sizeMN= size(X);
sizeTar= size(Y);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);
enitHO= sqrt(6)/(sqrt(numOutput)+sqrt(Numhidden));
enitIH= sqrt(6)/(sqrt(numInputs)+sqrt(Numhidden));

for i=1:10

per= randperm(numExample);

weightHO= (2*rand(Numhidden,numOutput)-1)*enitHO;
weightIH= (2*rand(numInputs,Numhidden)-1)*enitIH;

backPg(X,Y,weightIH,weightHO,per,Numhidden);

prop(X,Y,weightIH,weightHO,per,Numhidden);
fprintf('-----------------------------------------------');
end
end
```

Function for Wisconsin breast cancer dataset

```
function [ ] = cancer( Numhidden )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here



load('cancer_dataset');

X=cancerInputs;
Y=cancerTargets;

sizeMN= size(X);
sizeTar= size(Y);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);
enitHO= sqrt(6)/(sqrt(numOutput)+sqrt(Numhidden));
enitIH= sqrt(6)/(sqrt(numInputs)+sqrt(Numhidden));

%for i=1:10

per= randperm(numExample);

weightHO= (2*rand(Numhidden,numOutput)-1)*enitHO;
weightIH= (2*rand(numInputs,Numhidden)-1)*enitIH;

backPg(X,Y,weightIH,weightHO,per,Numhidden);

prop(X,Y,weightIH,weightHO,per,Numhidden);
fprintf('-----------------------------------------------');
%end
end
```

Common functions for all the datsets

Function for back propagation algorithm: backPg.m

```
function [ y,mse ] = backPg( inputs,targets,weightIH,weightHO ,per,numHidden)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

tic;

numEpoch=100;
alpha= 0.1;
%numHidden= 10;
sizeMN= size(inputs);
sizeTar= size(targets);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);

train= int32((2*numExample)/3);
test= int32(numExample/3);

%weightIH= rand(numInputs,numHidden)*0.46;

%weightHO= rand(numHidden,numOutput)*0.67;

InitialweightIH=weightIH;
InitialweightHO=weightHO;

% randomize the inputs
%per= randperm(numExample);

inp= zeros(size(inputs));
tar= zeros(size(targets));
 for i=1: numExample
inp(:,per(i))= inputs(:,i);
tar(:,per(i))=targets(:,i);

end

 inputs=inp;
 targets=tar;

 k=1;
```

```
for t=1 : numEpoch

   %fprintf('Iteration %d\n',t);

  for j=1: train
%%%  calculate the output of network

%numPattern=randi([1,numExample]);

numPattern= j;



hiddenVal= zeros(numHidden,1);
for j=1 : numHidden
   hiddenVal(j,1)=sum(weightIH(:,j).*inputs(:,numPattern));
   hiddenVal(j,1)=sigmoid(hiddenVal(j,1));
end

outputVal= zeros(numOutput,1);
for j=1 : numOutput
   outputVal(j,1)=sum(weightHO(:,j).*hiddenVal(:,1));
   outputVal(j,1)=sigmoid(outputVal(j,1));
end

%%%  Calculate the error in output

outputError=zeros(numOutput,1);
for i=1:numOutput
 error(i)=   targets(i,numPattern)-outputVal(i);
outputError(i)= outputVal(i)*(1- outputVal(i))*(targets(i,numPattern)-outputVal(i));
end

%%%% weights changes in HO layer

for i=1 : numHidden
   for j=1 : numOutput
      weightHO(i,j)= weightHO(i,j)+ alpha* hiddenVal(i,1)*outputError(j);



   end
end
```

```
%%%% weight change in IH layer

hiddenError= zeros(numHidden,1);

for i=1: numHidden
   hiddenError(i)=0;
   for j=1 : numOutput
      hiddenError(i)= hiddenError(i)+ weightHO(i,j)*outputError(j);
   end
   hiddenError(i)=hiddenError(i)*hiddenVal(i)*(1-hiddenVal(i));
end

for i=1: numInputs
   for j=1: numHidden
      weightIH(i,j)=weightIH(i,j)+ alpha*hiddenError(j)*inputs(i,numPattern);
   end
end




 end
 % end of an epoch

 % Mean squared error

mse(k)=  mean(error.^2)  ;



y(k)=k;
k=k+1;
if(mse(k-1)<0.001)break;
end

end
t
fprintf('Trained Succesfully');
result=1;

%show(weightIH,weightHO,InitialweightIH,InitialweightHO);
figure;
plot(y,mse,'Color','r','LineWidth', 2, 'MarkerSize', 7);
xlabel('Number of Epochs');
ylabel('MSE(Mean Square Error)');
```

```
toc;

tic;
%% checking for unseen values
c=0;

for j=train+1 : numExample
  % numPattern=j;
  numPattern=j;

hiddenVal= zeros(numHidden,1);
for j=1 : numHidden
   hiddenVal(j,1)=sum(weightIH(:,j).*inputs(:,numPattern));
   hiddenVal(j,1)=sigmoid(hiddenVal(j,1));
end

outputVal= zeros(numOutput,1);
for j=1 : numOutput
   outputVal(j,1)=sum(weightHO(:,j).*hiddenVal(:,1));
   outputVal(j,1)=sigmoid(outputVal(j,1));
end

%fprintf('\nActual values');
for i=1: numOutput
%fprintf(' %d ',targets(i,numPattern));
end

%fprintf('\ntarget values');
for i=1: numOutput
%fprintf(' %f ',outputVal(i,1));
end



  [a,b]=max(targets(:,numPattern));
  [e,d]=max(outputVal(:,1));

   if(b==d)

      c=c+1;
   end
```

```
end
acc= c*100/test;
fprintf('\ncorrectly classified %f \n',acc);

toc;
end
```

Function for proposed algorithm: prop.m

```
function [ y,mse ] = prop( inputs,targets,weightIH,weightHO,per ,numHidden)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

tic;

numEpoch=100;
alpha= 0.1;
%numHidden= 10;
sizeMN= size(inputs);
sizeTar= size(targets);
numInputs=sizeMN(1);
numExample=sizeMN(2);
numOutput=sizeTar(1);

train= int32((2*numExample)/3);
test= int32(numExample/3);

%weightIH= rand(numInputs,numHidden)*0.46;

%weightHO= rand(numHidden,numOutput)*0.67;

InitialweightIH=weightIH;

InitialweightHO=weightHO;

% randomize the inputs
%per= randperm(numExample);
%per
inp= zeros(size(inputs));
tar= zeros(size(targets));
```

```
 for i=1: numExample
inp(:,per(i))= inputs(:,i);
tar(:,per(i))=targets(:,i);

end

 inputs=inp;
 targets=tar;

 %inputs
 %targets
flag=1;

 k=1;
v=0;
for t=1 : numEpoch
  %  fprintf('Iteration %d\n',t);
  for j=1: train
%%%  calculate the output of network

%=randi([1,numExample]);

numPattern= j;



hiddenVal= zeros(numHidden,1);
biasVal= zeros(numHidden,1);

for j=1 : numHidden
   hiddenVal(j,1)=sum(weightIH(:,j).*inputs(:,numPattern)); %+biasVal(j,1);
   hiddenVal(j,1)=sigmoid(hiddenVal(j,1));
end

outputVal= zeros(numOutput,1);
biasOutVal=zeros(numOutput,1);

for j=1 : numOutput
   outputVal(j,1)=sum(weightHO(:,j).*hiddenVal(:,1)) ;%+ biasOutVal(j,1);
   outputVal(j,1)=sigmoid(outputVal(j,1));
end

%%%  Calculate the error in output

outputError=zeros(numOutput,1);
for i=1:numOutput
   error(i)=   targets(i,numPattern)-outputVal(i);
```

```
outputError(i)= outputVal(i)*(1- outputVal(i))*(targets(i,numPattern)-outputVal(i));
end
%outputError(2)= outputVal(2)*(1- outputVal(2))*(targets(2,numPattern)-outputVal(2));

%%%% weights changes in HO layer

for i=1 : numHidden
   for j=1 : numOutput
     if(weightHO(i,j)==0)continue;end
       weightHO(i,j)= weightHO(i,j)+ alpha* hiddenVal(i,1)*outputError(j);
   end
end

%%%% weight change in IH layer

hiddenError= zeros(numHidden,1);

for i=1: numHidden
   hiddenError(i)=0;
   for j=1 : numOutput
      hiddenError(i)= hiddenError(i)+ weightHO(i,j)*outputError(j);
   end
   hiddenError(i)=hiddenError(i)*hiddenVal(i)*(1-hiddenVal(i));
end

for i=1: numInputs
   for j=1: numHidden
     if(weightIH(i,j)==0)continue; end
      weightIH(i,j)=weightIH(i,j)+ alpha*hiddenError(j)*inputs(i,numPattern);
   end
end




  end
 mse(k)=  mean(error.^2)  ;
y(k)=k;
k=k+1;

   % change in weight less than threshold
 if(mse(k-1)<0.02 && flag==1)
    %avg= show(weightIH,weightHO,InitialweightIH,InitialweightHO);
    %avg
     flag=0;
     for i=1: numInputs
       for j=1: numHidden
          if(weightIH(i,j)~=0&& abs(weightIH(i,j)-InitialweightIH(i,j))<0.05 )
```

```
          %  biasVal(j,1)= weightIH(i,j)*inputs(i,numPattern);
            weightIH(i,j)=0;v=v+1;
            %fprintf('\n %d %d',i,j);

         end
       end
      end

  for i=1 : numHidden
   for j=1 : numOutput
      if(weightHO(i,j)~=0 && abs(weightHO(i,j)-InitialweightHO(i,j))<0.05)
        % biasOutVal(j,1)= weightHO(i,j)*hiddenVal(i);
         weightHO(i,j)=0;v=v+1;
        %fprintf('\n %d %d',i,j);
      end

   end
  end

 end
if(mse(k-1)<0.001)break;
end

end
t
fprintf('Trained Succesfully');
result=1;

%show(weightIH,weightHO,InitialweightIH,InitialweightHO);
hold on;
plot(y,mse,'Color','b','LineWidth', 2, 'MarkerSize', 7);
hold off;
xlabel('Number of Epochs');
ylabel('MSE(Mean Square Error)');
toc;
fprintf('\n\nNo of weights removed %d, total weights %d \n',v,(numInputs+numOutput)*numHidden);
%weightIH
%weightHO
tic;
%% checking for unseen values
c=0;

for j=train+1 : numExample
  % numPattern=j;
  numPattern=j;

hiddenVal= zeros(numHidden,1);
```

```
for j=1 : numHidden
   hiddenVal(j,1)=sum(weightIH(:,j).*inputs(:,numPattern));
   hiddenVal(j,1)=sigmoid(hiddenVal(j,1));
end

outputVal= zeros(numOutput,1);
for j=1 : numOutput
   outputVal(j,1)=sum(weightHO(:,j).*hiddenVal(:,1));
   outputVal(j,1)=sigmoid(outputVal(j,1));
end

%fprintf('\nActual values');
for i=1: numOutput
%fprintf(' %d ',targets(i,numPattern));
end

%fprintf('\ntarget values');
for i=1: numOutput
%fprintf(' %f ',outputVal(i,1));
end



  [a,b]=max(targets(:,numPattern));
  [e,d]=max(outputVal(:,1));

   if(b==d)
          c=c+1;

   end




end
acc=c*100/test;
fprintf('\ncorrectly classified %f \n',acc);

toc;
 end
```

Other helping functions:

Sigmoid.m

```
function [ output ] = sigmoid( x )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

y=  exp(-1*x);

output=  1/(1+y);

end
```

show.m

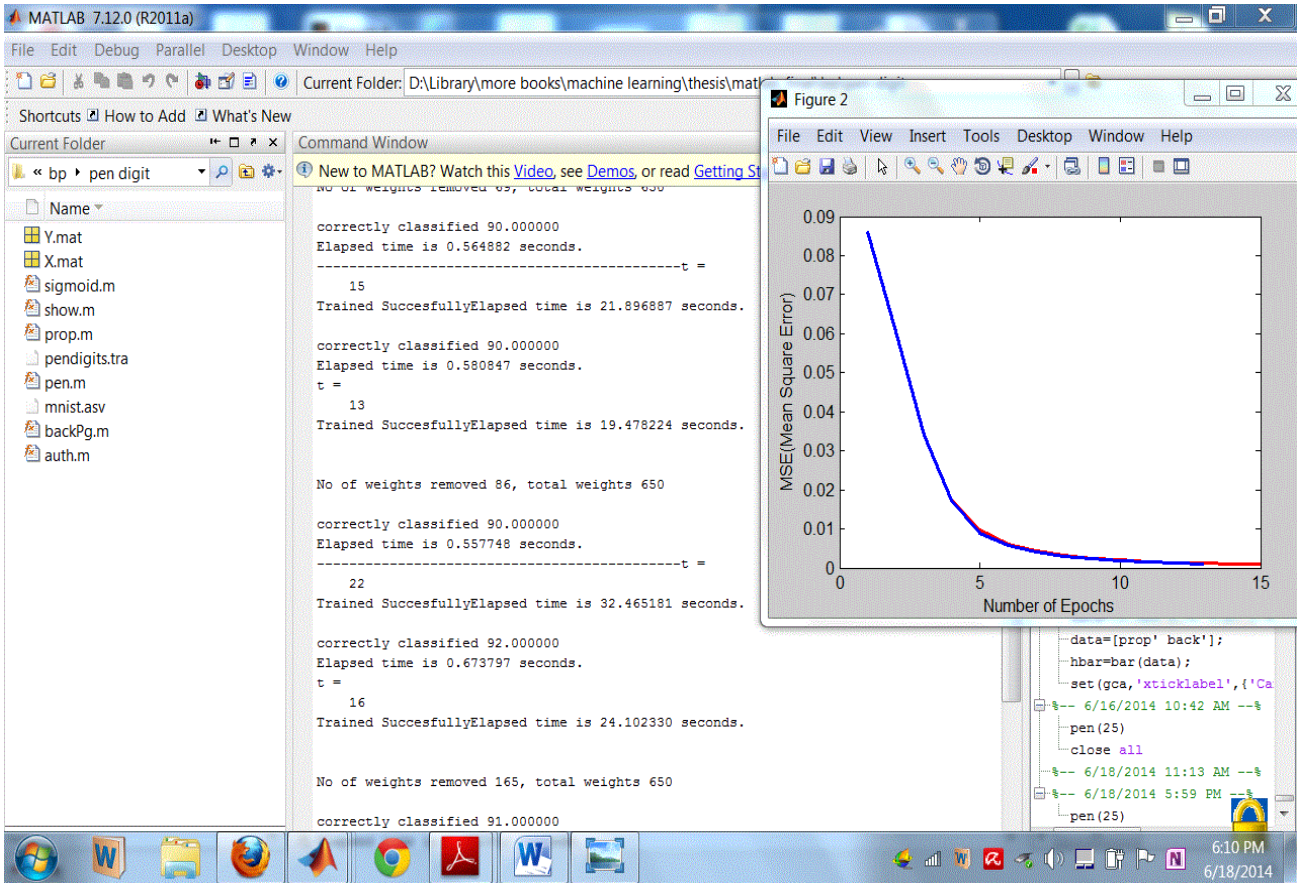
```
function [ avg ] = show( theta1,theta2,initialtheta1,initialtheta2 )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

x=size(theta1);
hidden= x(2);
input=x(1);  %including no bias

y=size(theta2);
output=y(2);
sum=0;
%fprintf('Weights IH layer');

for i=1 : hidden
  %  fprintf('Weights for hidden unit: %d \n',i);
    for j=1 : input
        diff=initialtheta1(j,i)-theta1(j,i);
        sum=sum+abs(diff);
      %  fprintf('Hidden %d  : Before and after %d   %d   Difference  %d\n',i,initialtheta1(j,i),theta1(j,i),diff);
    end

end
```

```
%fprintf('Weights HO layer');

for i=1 :output
 %   fprintf('Weights for output unit: %d \n',i);
   for j=1 : hidden
       diff2=initialtheta2(j,i)-theta2(j,i);
       sum=sum+abs(diff2);
      % fprintf('Output  %d  :Before and after %d   %d   Difference
%d\n',i,initialtheta2(j,i),theta2(j,i),diff2);
   end

end

avg= sum/(hidden*(input+output));

end
```

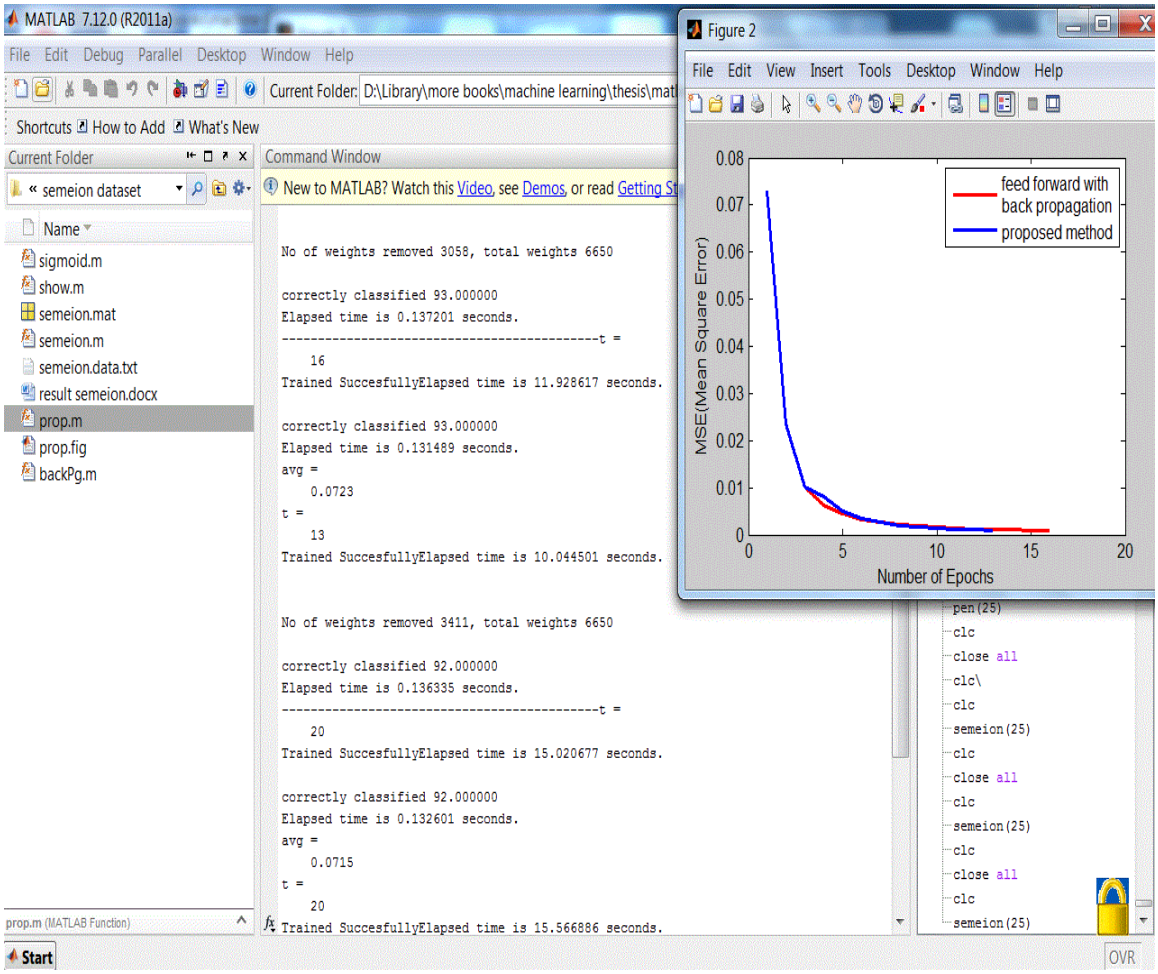
APPENDIX B

SCREEN SHOTS

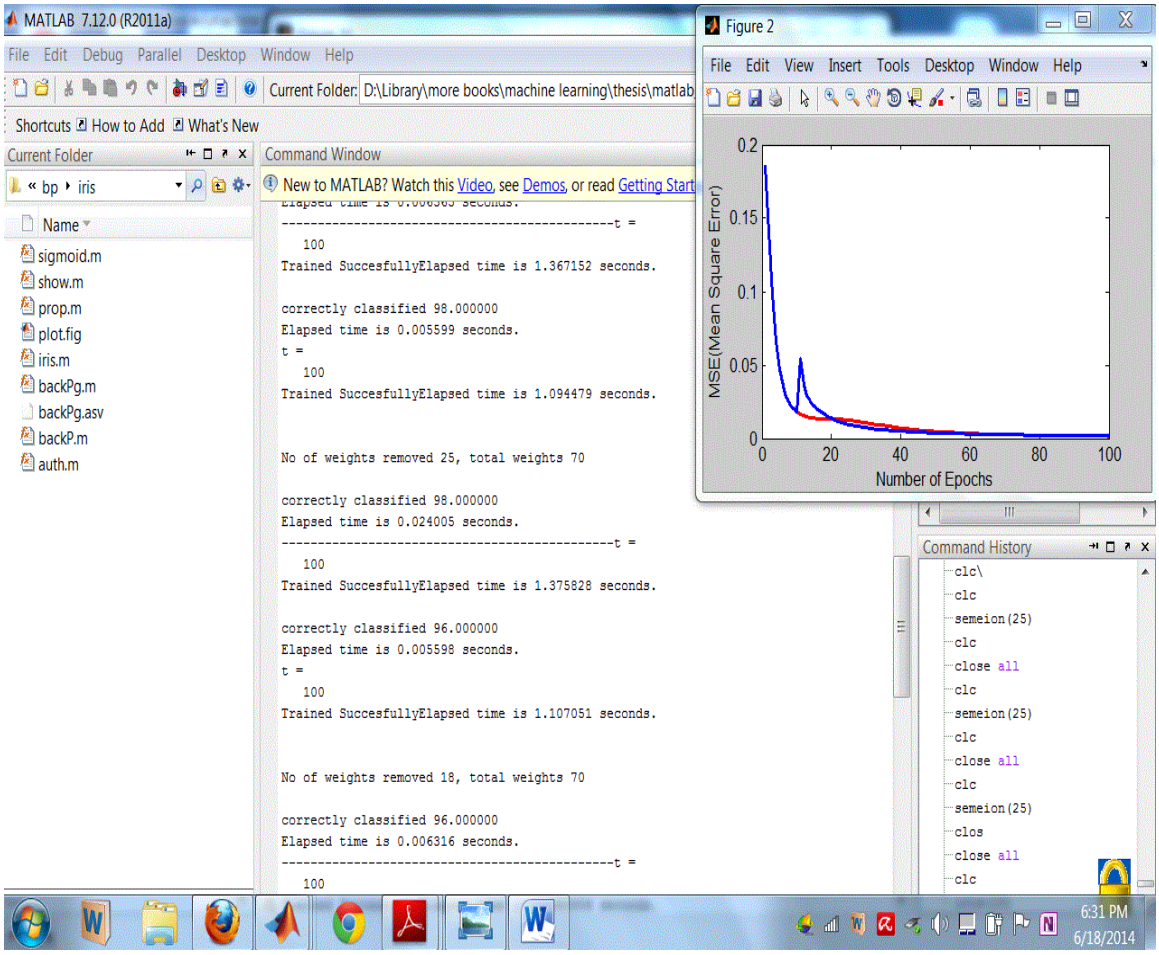
Screen shot for pen based handwritten digit dataset



Screen shot for SEMEION digit dataset



Screen shot for Iris dataset



Screen shot for Wisconsin breast cancer dataset