CHAPTER 4

RESULTS

4.1Introduction

The results of network training simulations that have been conducted are presented in this chapter. The goals of these experiments were to determine the data set composition and neural network architecture that would produce the best predictive performance for life insurance datasets and hence provide an intelligent decision support.

These experiments address the three main issues of this research.

We have investigated predictive performance for applied first and second order algorithms. A large number of simulations have been developed with varying parameters of network like number of hidden layers, number of neurons, initial weights, architectures, training methods etc.An initial ANN model was selected and an iterative approach was adopted in improving the predictive performance of the initial neural networkarchitecture. This process mainly entailed modifying the number of hidden neurons, training functions, transfer functions, datasets etc. and evaluating the performance of the resulting trained neural networks with some statistical measures and plots.The findings indicated that data set composition and neural network architecture had a critical influence on the predictive performance of different methods.

Also the speed of convergence toward minimum error is an important issue and experiments have been conducted to judge the convergence time. Time taken to reach toward the final set targets from the initial gradient values has been observed for different training functions and for a variety of model simulations. Time taken is closely related to factors like number of epochs used for the convergence toward final solution which mainly depends upon size of dataset and platform configuration.

We have also applied the achieved solutions based on previous and the proposed method to predict the performance of the models for new instances of the datasets and to check for the predictive accuracy for different models.

Plots like ROC curve, confusion matrix, regression plots and performance curves have been plotted to understand the performance and convergence behavior of the models developed.

Several ANN model configurations have been selected for investigation; these mainly include multilayer feed-forward architecture with different training algorithms. The models typically consist of three layers, namely an input, hidden and output layer. They are initially configured in form of a network object and set with required input, initial parameters and the target output values. Following set of steps have been adopted to evaluate the performance and desired results in each simulation.

Repeat for each selected initial ANN model based upon feed–forward network*(implemented in SPSS and MATLAB):*

1. Select initial input and output dimensions of the model.

2. Configure initial network architecture.

3. Select and compose the data sets needed to train, validate and test the model.

4. Repeat until network training completes by:

    4.1 Either achieving the set target or any other stopping criterion.

    4.2 Observe and test the network performance and training time.

    4.3 Incrementally modify the number of hidden layer neurons and other parameters.

The findings of these experiments have been compared with each other in the comparative study covered in the next chapter.

4.2Performance of Neural Network Models

The process of neural network training involves tuning the values of the weights and biases vectors of the network to optimize the network performance, as defined by the network performance function '*net.performFcn'*. The default performance function for multilayer feed-forward networks is mean square error 'MSE' — it defines the

average squared error between the network outputs and the target outputs. It is defined as follows:

$$F = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2 \qquad (4.1)$$

It is possible to estimate the quality of the training by computing Mean Squared Error (MSE). The function actually expresses the difference between the actual and target outputs provided in the network; the performance of network is better if MSE is smaller (becomes closer to zero). Other types of error measures performance functions like MAE (Mean absolute error performance function) and SSE (Sum squared error performance function) are also available in MATLAB but MSE is commonly used error function.

4.2.1 Performance FunctionsAvailable in Neural Network Toolbox

- *'mae'* Mean absolute error performance function
- *'mse'* Mean squared error performance function
- *'sse'* Sum squared error performance function

There are two different ways in which training can be implemented: incremental mode and batch mode. In incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In batch mode, all the inputs in the training set are applied to the network before the weights are updated. We have applied batch mode training with the '*train'* command for computation of '*perf*' values which correspond to MSE.

The term '*perf*' in algorithm actually represents computation of MSE for the given dataset. After experimental investigations, the best results obtained for different algorithms including newly developed normalized variation have been presented in Table 4.1 shown below. Table provides a summary of the results performance in mean percentage of trained networks for all models. Each network model has been

trained from three different pattern sets, namely training, validation and testing dataset.

Table 4.1 Experiment results of employing different gradient-based learning algorithms of the first and second order including adaptive algorithms and the proposed algorithm

Training Algorithm	Training Function	Min. gradient	Neurons in hidden layer	Final epochs	Training time	Training performance	Starting gradient value	Final gradient value
Conjugate gradient	**traincgp**	**0.0001**	**15**	**135**	**0:07:24**	**0.0390**	**0.6760**	**6.96e-05**
Scaled conjugate gradient	**trainscg**	**0.0001**	**15**	**119**	**0:04:41**	**0.0375**	**0.447**	**8.04e-05**
Levenberg Marquardet	**trainlm**	**0.0001**	**15**	**71**	**0:01:39**	**0.0138**	**0.447**	**7.86e-05**
Steepest (gradient) descent	**traingd**	**0.0001**	**15**	**10^3**	**0:19:06**	**0.0581**	**0.447**	**0.0421**
Gradient descent with adaptive learning rate	**traingda**	**0.0001**	**15**	**10^3**	**0:14:04**	**0.0464**	**2.57**	**0.0816**
Gradient descent with momentum	**traingdm**	**0.0001**	**15**	**10^3**	**0:14:24**	**0.0584**	**0.447**	**0.0427**
Normalized adaptive (Proposed Method)	***norm adaptive***	**0.0001**	**15**	**209**	**0:03:41**	**0.0499**	**2.57**	**9.95e-03**

Different types of plots describing the performances, error gradient variations, convergence behavior, error histograms showing the error summaries for in different bins, regression plots for the methods applied have been shown and explained in the following sections. In addition plots to test the model accuracy like ROC and confusion matrix have also been discussed.

4.3Graphsfor Neural Network Models

In the following section, graphs like performance function in terms of Mean Squared Error (MSE) known as performance plot, error gradient graphs, regression parameter graphs, histograms etc. have been plotted for various applied methods and observed to analyze their convergence and behavior to achieve the set target values of error gradient. A large number of simulations for predictive models with different configurations of neural networks have been tested in MATLAB before describing the best results; while employing all the above said algorithms.

4.3.1 Neural Network Training Window

'nntraintool' command opens the neural network training GUI. It has been used in the algorithms to invoke the training window for the neural network models. This function can be used to make the training GUI visible before the training has occurred and presents the facilities to study the training progress and plot the required graphs related to training and call other important functions. Network training functions handles all activity within the training window.

To access additional useful graphs, related to the current network trained, we can click their respective buttons in the training window, during or after training completes. '*nntraintool close'*command closes the training window.

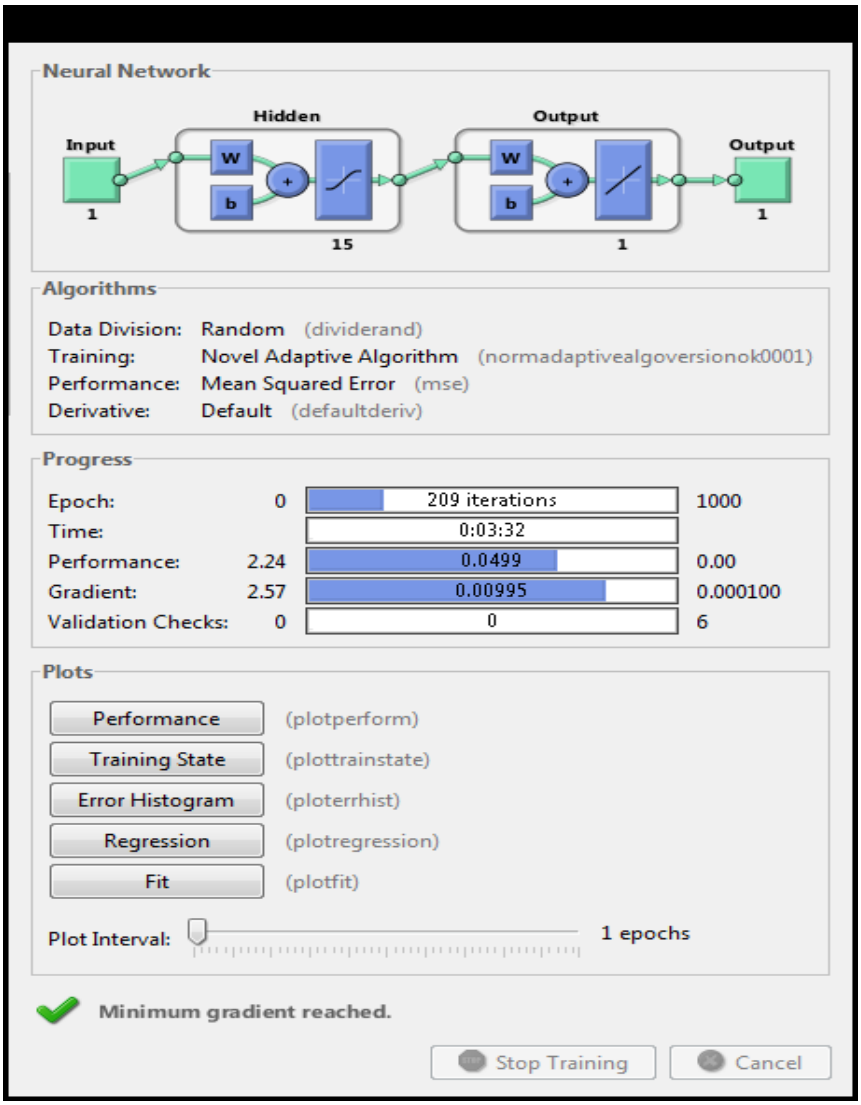

Fig. 4.1 (a) Neural network training window

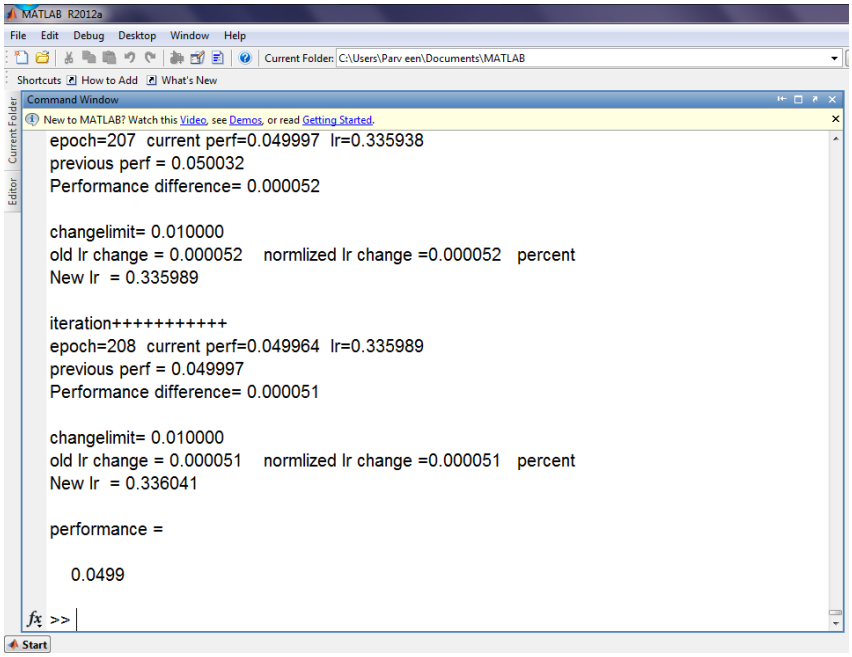

Fig. 4.1 (b) A snapshot for neural networktraining process

Note: *[Figures 4.1(a) and 4.1(b) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

One sample training window for our work (here for the proposed algorithm) has been displayed in the figure 4.1(a) below. Figure 4.1(b) displays the snapshot for the training process in progress while calculating the necessary parameters during program execution in each epoch. This also displays the final performance value achieved by the applied algorithm.

4.3.2Performance Graphs

As seen in the previous section, training performances of the applied first and second order algorithms have been presented in the Table 4.1.Figures 4.2(a) to 4.2(g) as shown below demonstrate the performance curves for mean square error (MSE) versus training epochs for the applied methods including the adaptive techniques and newly suggested methods. The graphs have been plotted according to logarithmic scale and the best training performance values with the final epoch values, when the training stops have been mentioned in the graph. These performance curves have been plotted from the buttons available on the training window. Alternatively, we can make use of '*plotperform'* function available in neural toolbox. *plotperform*(*TR*) plots the training, validation, and test performances according to the training record object *TR* returned by the function train.

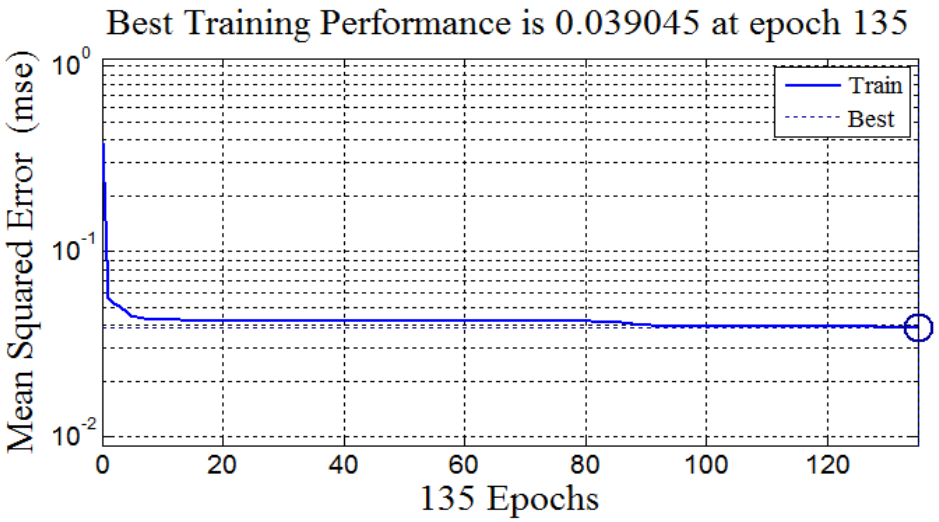

Fig. 4.2(a) Training performance graph with conjugate gradient learning

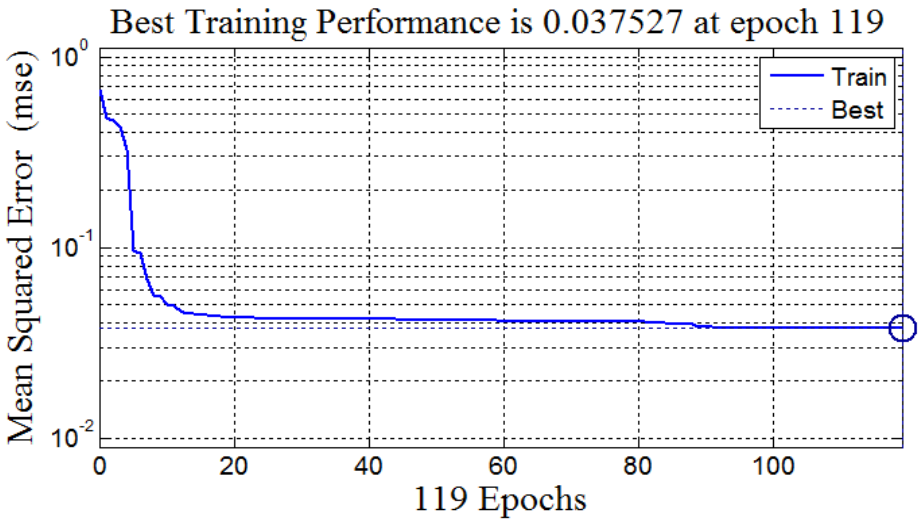

Fig.4.2(b) Training performance graph with scaled conjugate gradient learning

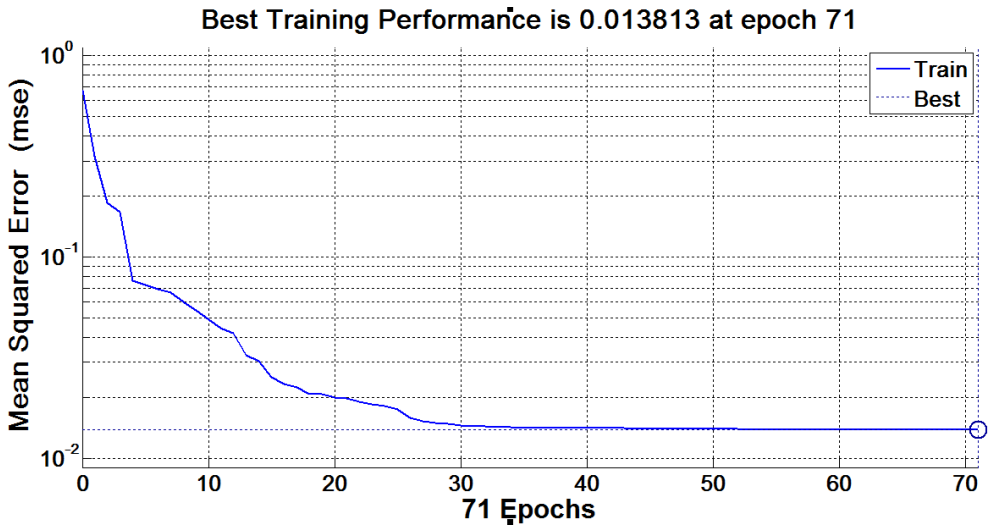

Fig.4.2(c) Training performance graph with Levenberg Marquardet learning

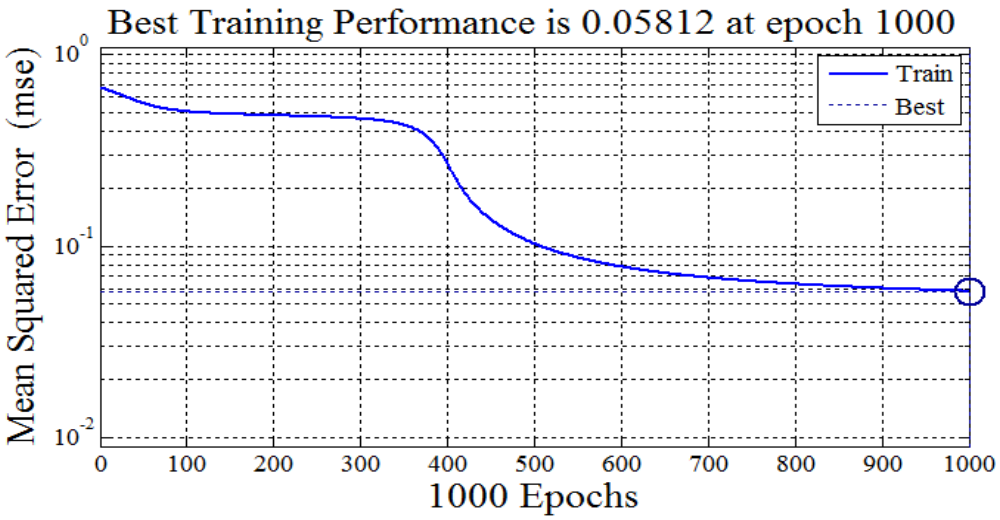

Fig.4.2(d) Training performance graph with simple gradient descent learning

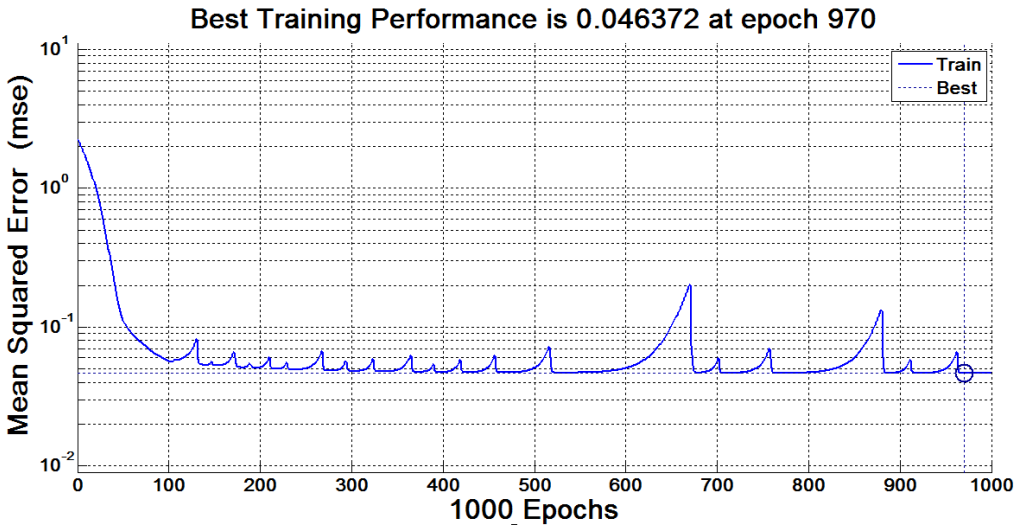

Fig. 4.2(e) Training performance graph with gradient descent adaptive learning

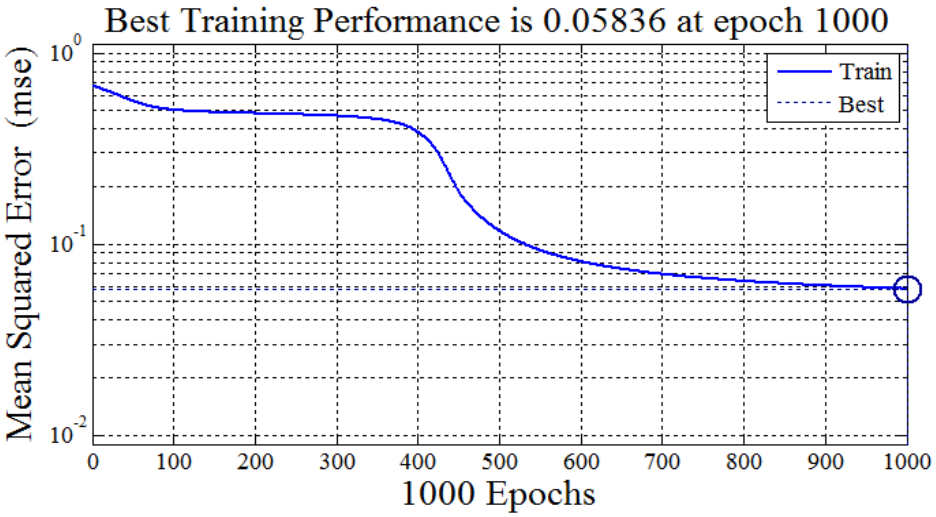

Fig.4.2(f) Training performance graph with gradient descent adaptive momentum

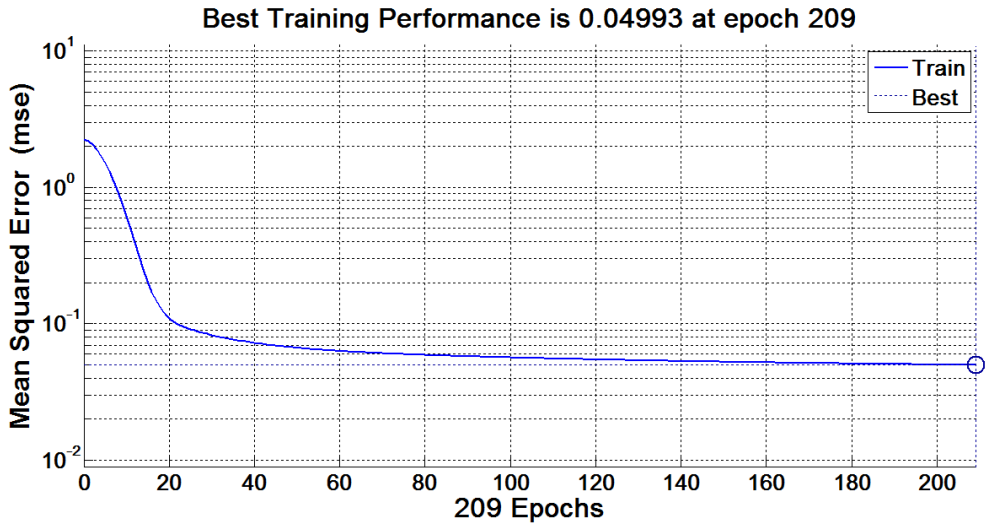

Fig. 4.2(g) Training performance graph with normalized adaptive learning algorithm

Note: *[Figures 4.2(a) to 4.2(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

We can see clearly see in the performance graphs that second order methods were able to converge toward solution in a better way than first order methods. Best training performance and the epoch value has been shown in the graph for each algorithm. Convergence of different algorithms is tested for error gradient target of 0.0001. It has been clearly observed that second order methods are able to accomplish the target value of error gradient of the order of 10^{-4} like CGM found the solution in 135 epochs (perf value = 0.0390) and SCGM converged in 119 epochs(perf value = 0.0375) and Levenberg Marquardet method was the best to achieve it only in 71 epochs (perf value = 0.0138).

On the other hand, steepest decent and existing adaptive methods are not able to achieve the solution even in 1000 epochs (perf values are more than 0.500) but the proposed normalized adaptive algorithm is able to converge in 209 epochs (perf value = 0.499). It has been observed that the performance of the proposed algorithm is better than existing adaptive methods but it is found below second order algorithms.

4.3.3Error Gradient Curves

Figures 4.3(a) to 4.3(g) shown below show the graph for error gradient vs. epochs. The graphs have been plotted according to logarithmic scale and the final error gradient values with the final epoch values, when the training stops have been mentioned in the graph. We can clearly observe in the gradient plots that how error gradient varies with

number of epochs and falls down from the initial gradient values toward the set target for minimum gradient. These gradient curves have been plotted from the training state button available on the training window. Alternatively, we can make use of '*plottrainstate'* function available in neural toolbox. *plottrainstate*(*TR*) plots the training state from a training record object *TR*returned by function '*train'*.

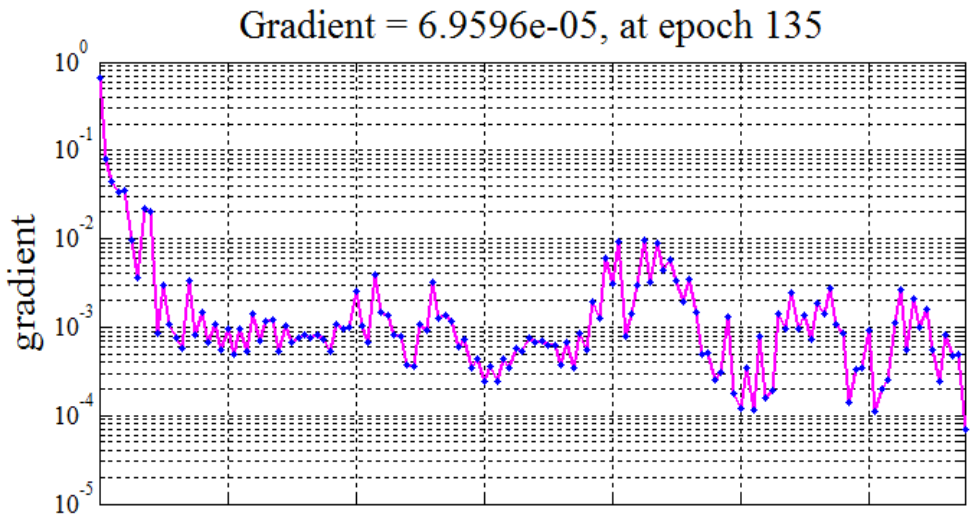

Fig. 4.3(a)Error gradient graph with conjugate gradient learning

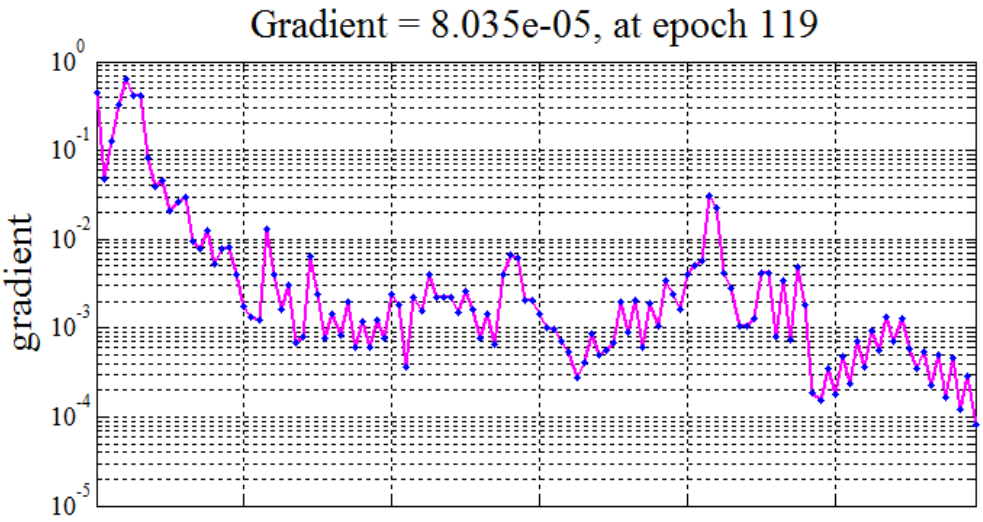

Fig.4.3(b)Error gradient graph with scaled conjugate gradient learning

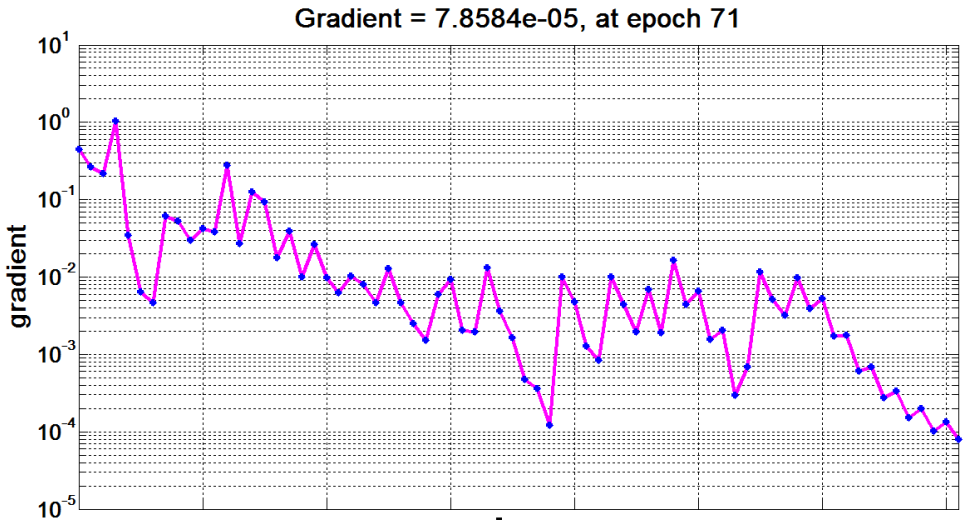

Fig.4.3(c) Error gradient graph with Levenberg Marquardet learning

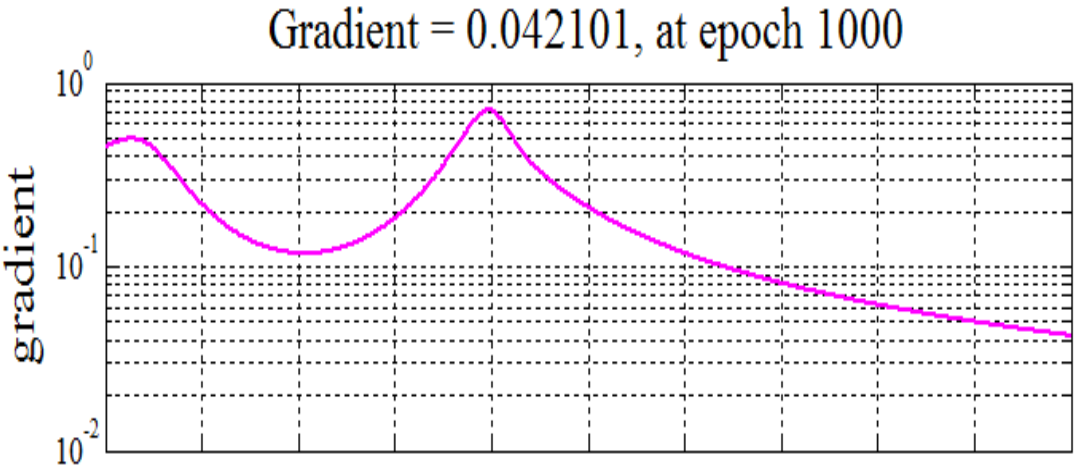

Fig. 4.3(d)Error gradient graph with simple gradient descent learning

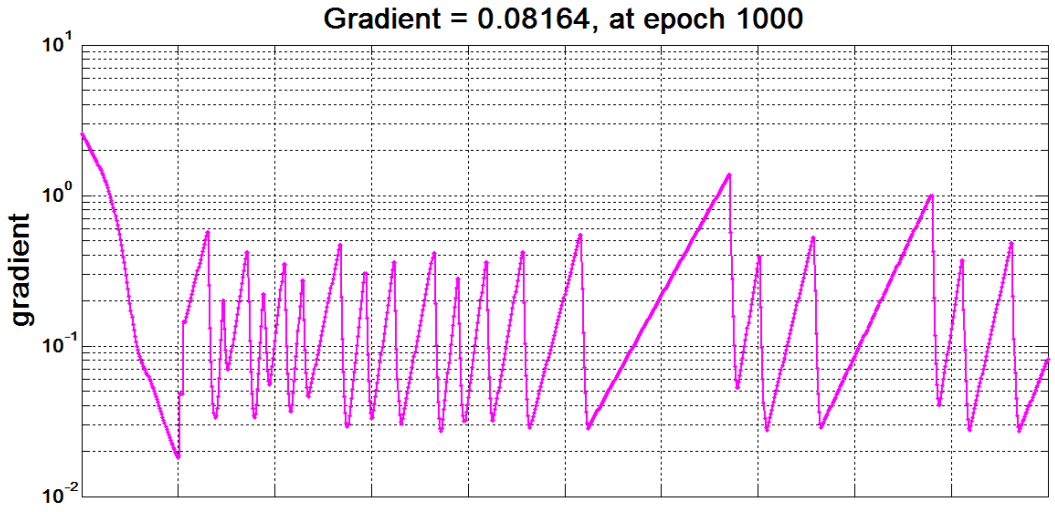

Fig. 4.3(e)Error gradient graph with gradient descent adaptive learning

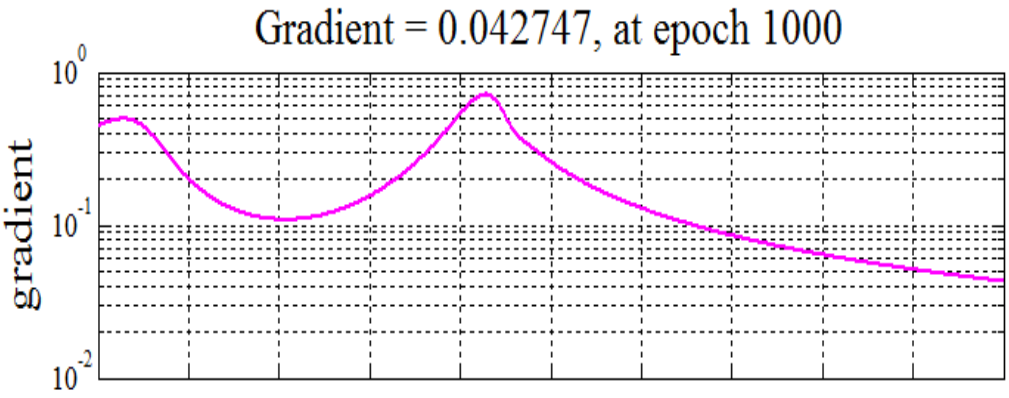

Fig. 4.3(f)Error gradient graph with gradient descent adaptive momentum

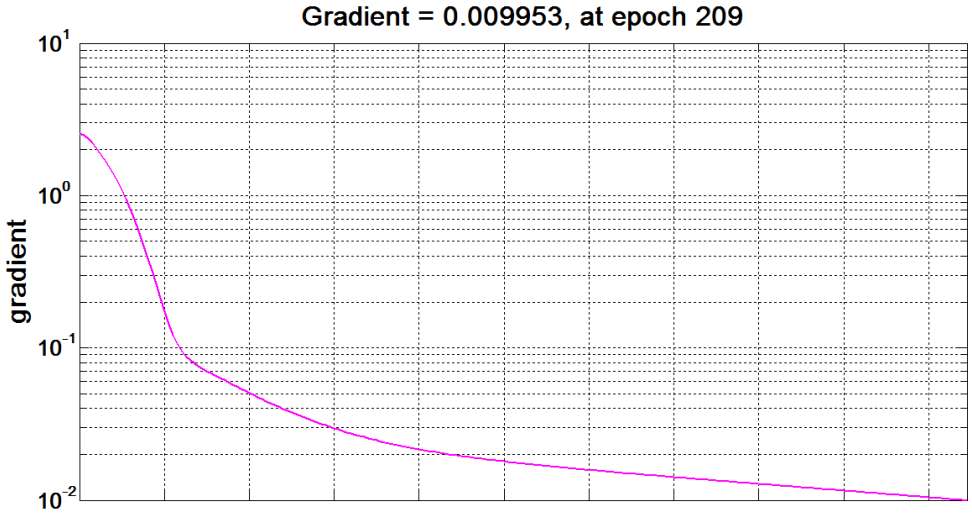

Fig. 4.3(g) Error gradient graph with normalized adaptive learning algorithm

Note: *[Figures 4.3to 4.3(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

We can see clearly in the error gradient graphs thatsecond order methods could reach the error gradient values of the order of 10^{-5}. Levenberg Marquardt method was able to achieve best minimum error gradient value of 7.86e-05 and also in the fastest time of 0:01:39 minutes and in 71 epochs. No other algorithm was able to minimize the error gradient till this extent. First order methods and adaptive techniques have also failed to achieve the set target of minimum of error gradient even in 1000 epochs. But the proposed algorithm was able to achieve the set target and reached final value of 9.95e-03 and it was reached in 0:03:41 minutes and in 209 epochs. Its final gradient value is not close to second order techniques but far better than first order methods and other adaptive methods.

4.3.4 Regression Plots

Another important plot to observe the correctness of the developed model is to create a regression plot, which shows the relationship between the outputs of the network model and the target values. If training of model were perfect, then we find that network outputs and the targets would be exactly equal, but the relationship is rarely perfect in actual practice.

Three plots in different colors (blue, green and red) generally represent the training, validation, and testing data. The dashed line in each plot represents the perfect results – 'outputs = targets'. The solid line represents the best fit linear regression line between outputs and targets. The regression coefficient *'R'* value is an indication of the relationship between the outputs and targets. If *'R' = 1*, this indicates that there is an exact linearrelationship existing between outputs and targets. If *'R'* is close to zero, then there is no existing linear relationship between outputs and targets.

These regression graphs have been plotted from the buttons available on the training window and have been shown in Figures from 4.4(a) to Figure 4.4(g). Alternatively we can make use of '*plotregression'* function available in neural toolbox. *plotregression(targets,outputs)* plots the linear regression of targets relative to outputs.

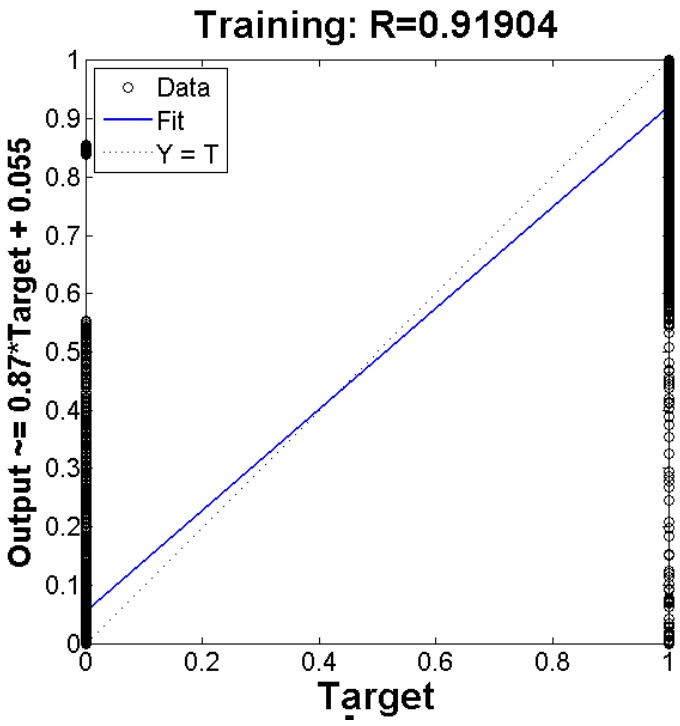

Fig. 4.4(a) Regression plot for conjugate gradient algorithm

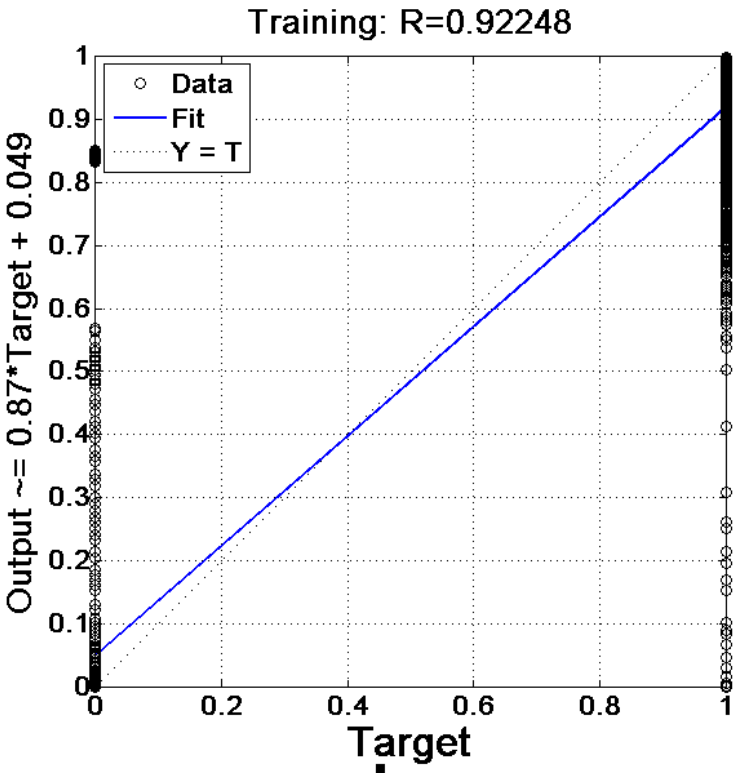

Fig. 4.4(b) Regression plot for scaled conjugate gradient algorithm

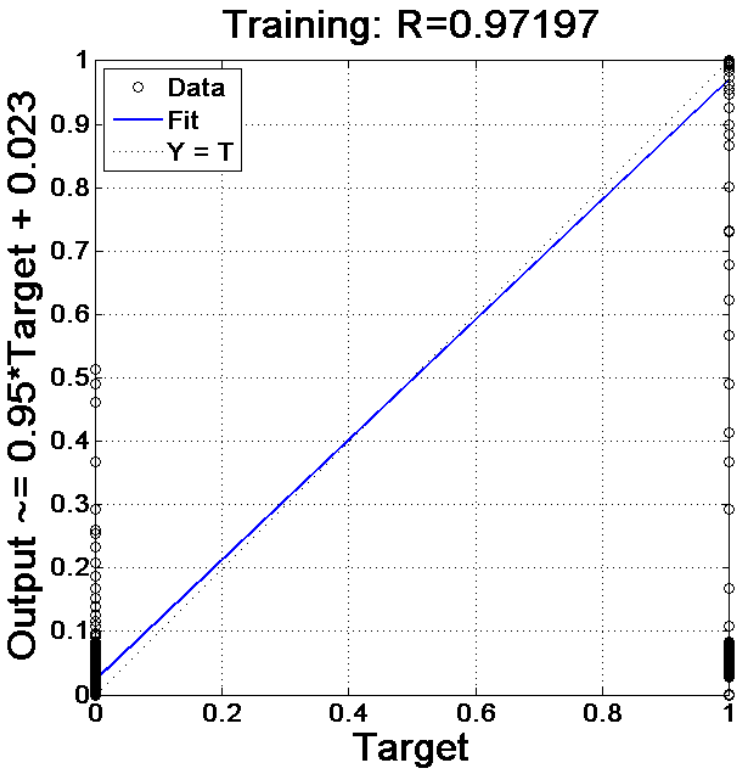

Fig. 4.4(c) Regression plot for Levenberg Marquardet algorithm

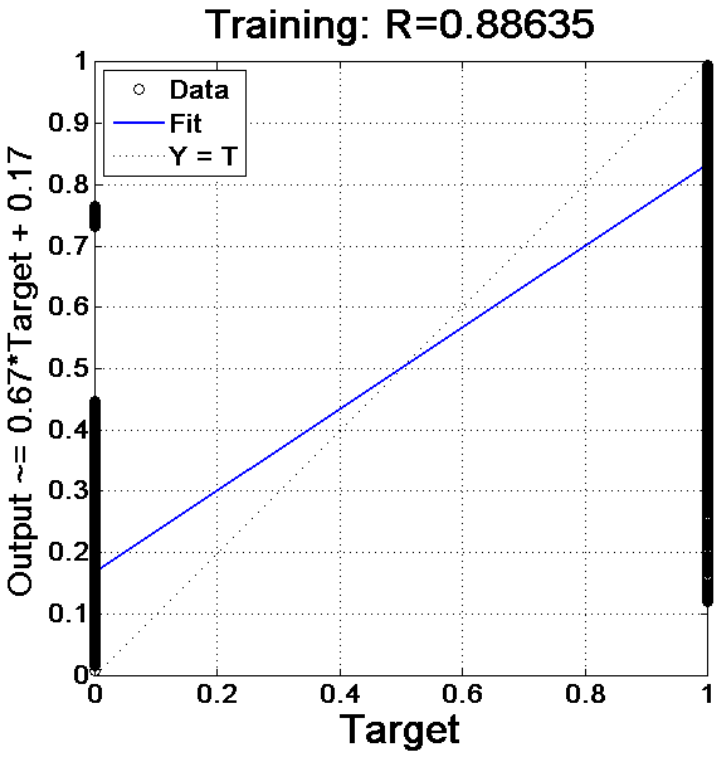

Fig. 4.4(d) Regression plot forsimple gradient descent algorithm

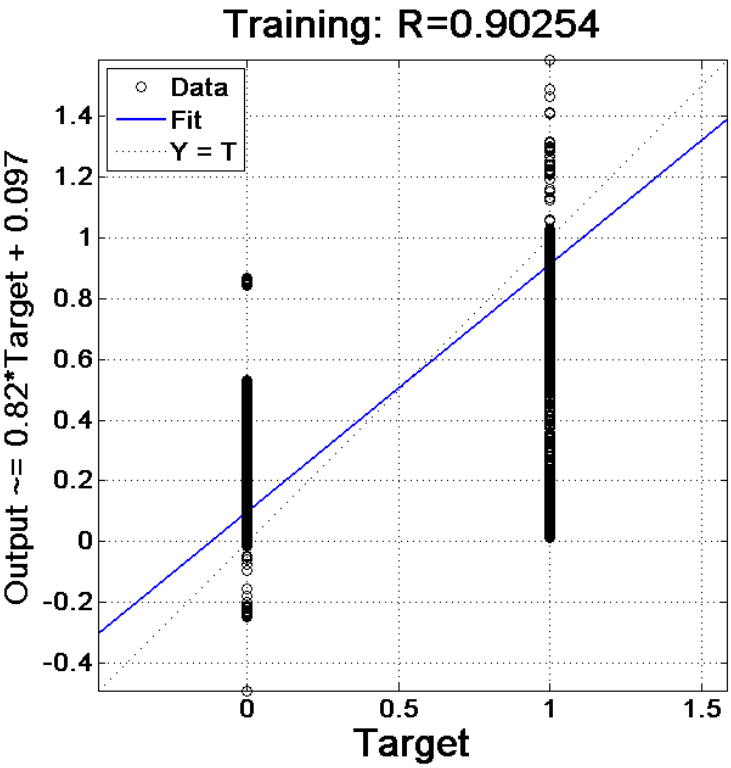

Fig. 4.4(e) Regression plot forgradient descent adaptive learning algorithm

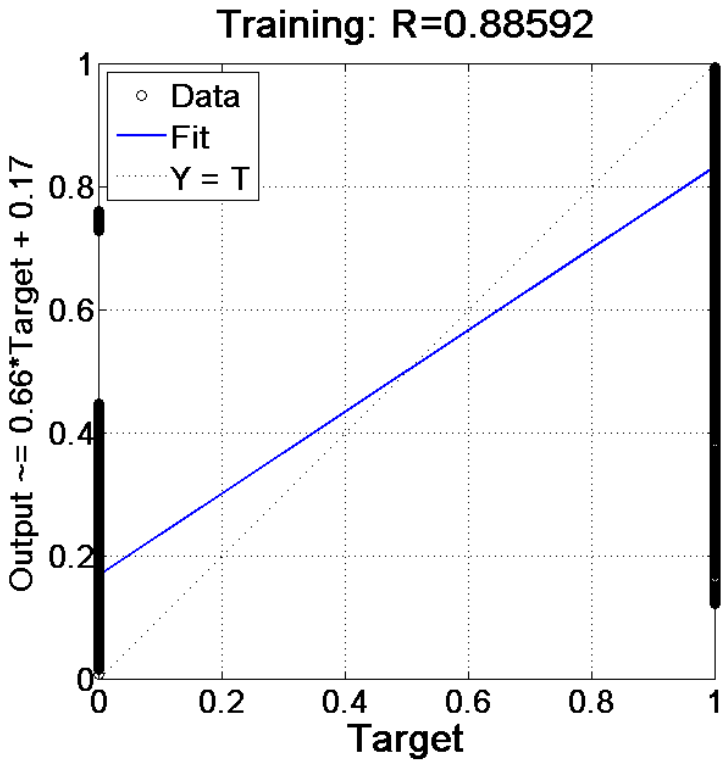

Fig. 4.4(f) Regression plot forgradient descent adaptive momentum algorithm

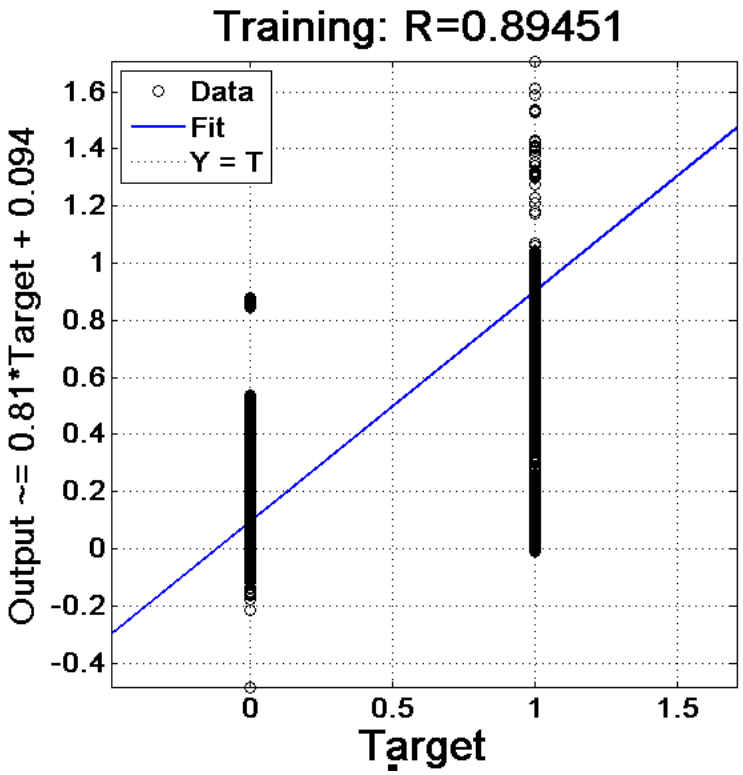

Fig. 4.4(g) Regression plot for normalized adaptive learning algorithm

Note: *[Figures 4.4 to 4.4(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

For all above method applied, the training data indicates a good fit. Levenberg Marquardet algorithm has shown the best value of 0.97197. Other second order methods also show *R* values that greater than 0.9. For the proposed method this is coming as 0.89451 which is again a very good fit for the predictive accuracy of the model.Dotted line in the plots depicts the perfect fit and blue line shows the actual fit for the training data. It is shown for training data in case we use '*nntool'* but with '*nftool'* and codingwe get three color lines for training, validation and test data. However, we can make use of built-in functions explicitly to get the other lines in the plot.

4.3.5 Histogram Plots

Once the network has been trained and validated, we can use the network object to calculate the network response to any input. We need to judge the error differences for the new instances or for the entire dataset. We can plot the histograms and find the error values as the difference between target values and predicted values. Rising of histogram bars around the line of zero error as denoted with orange color denotes the accuracy of designed model. These histogram plots have been plotted from the buttons available on the training window. Alternatively, we can make use of '*ploterrhist'* function available in neural toolbox. *ploterrhist(e) plots a histogram of error values 'e'.*

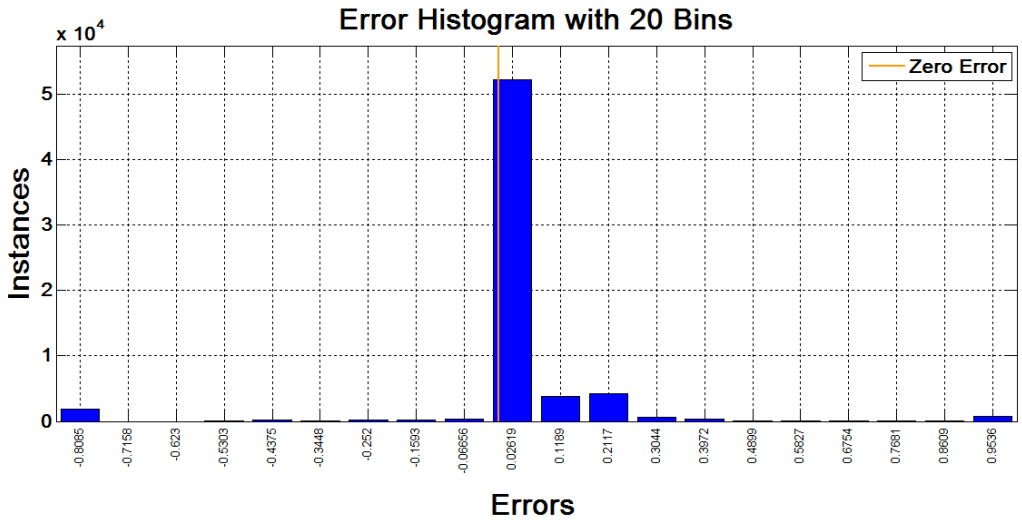

Fig. 4.5(a) Histogram plot for conjugate gradient algorithm

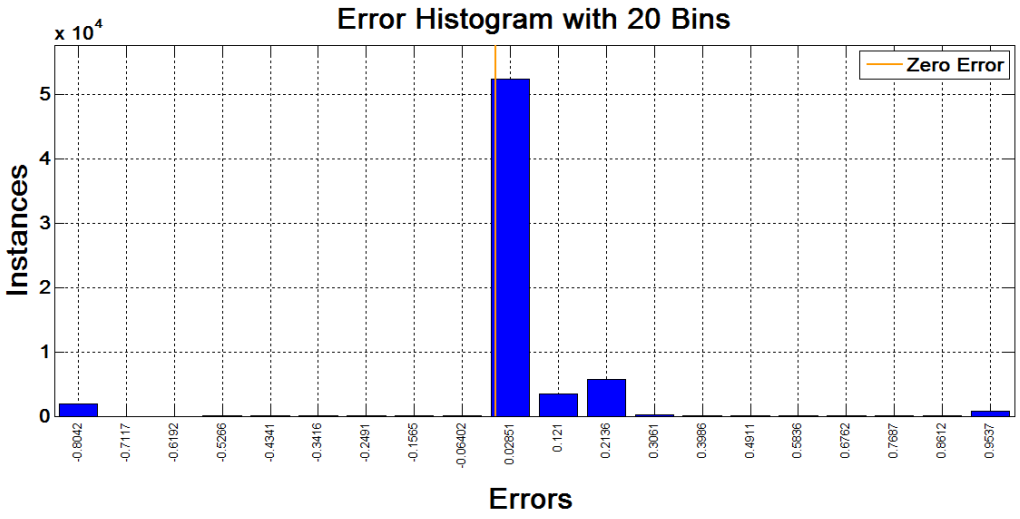

Fig. 4.5(b) Histogram plot for scaled conjugate gradient algorithm

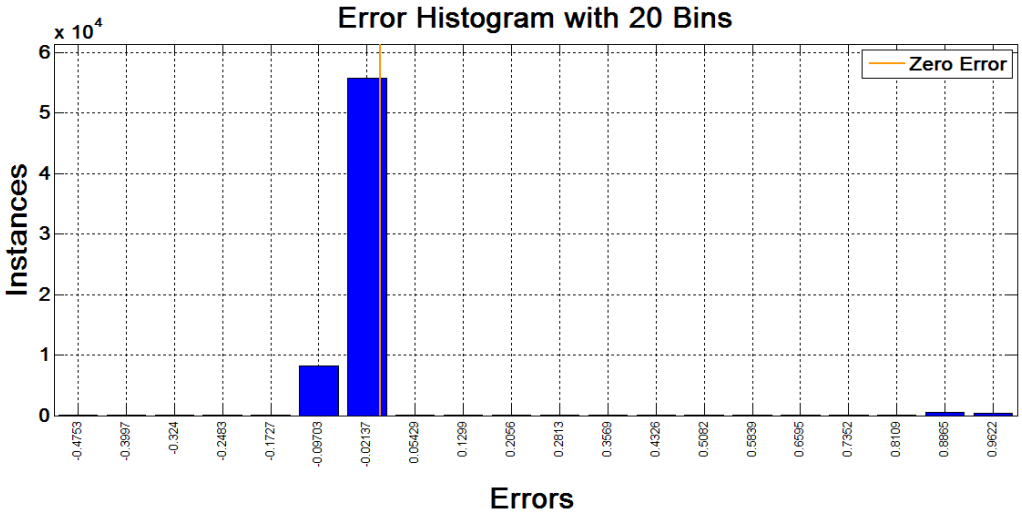

Fig. 4.5(c) Histogram plot for Levenberg Marquardetalgorithm

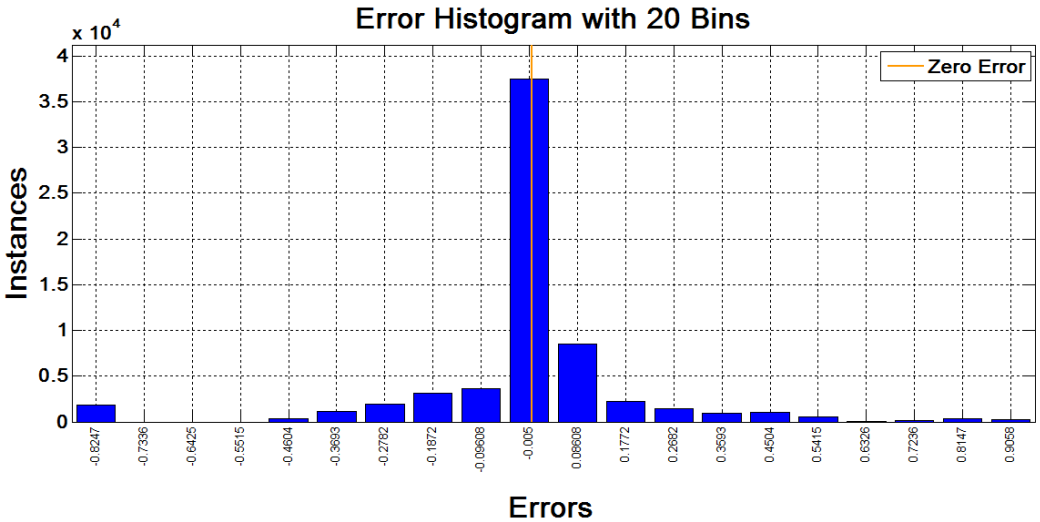

Fig. 4.5(d) Histogram plot forsimple gradient descent algorithm

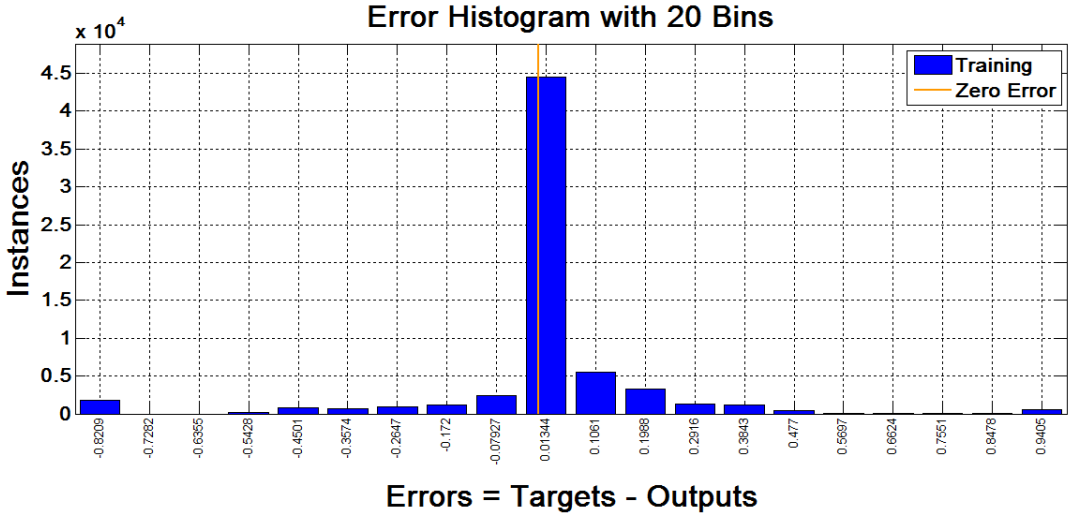

Fig. 4.5(e) Histogram plot for gradient descent adaptive learning algorithm

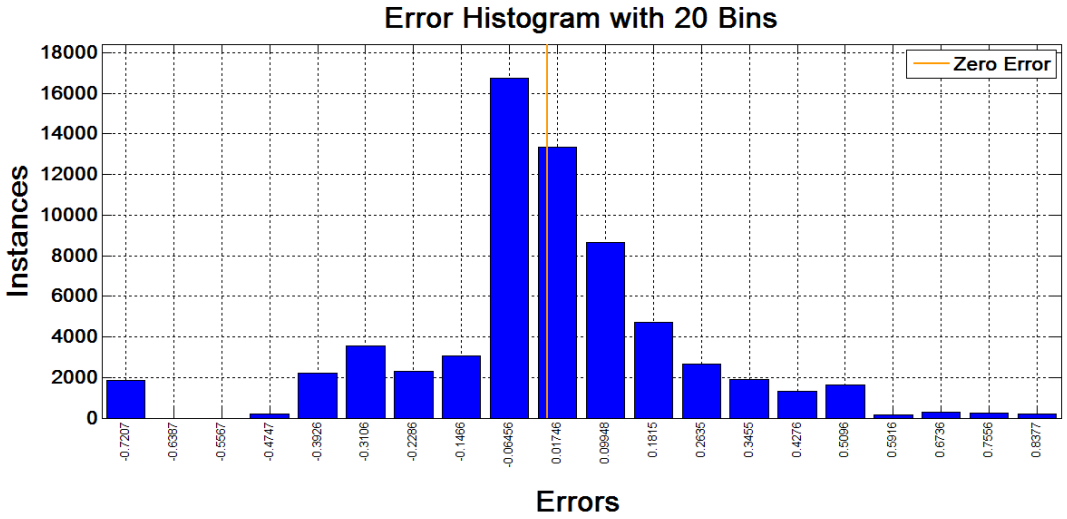

Fig. 4.5(f) Histogram plot for gradient descent adaptive momentum algorithm

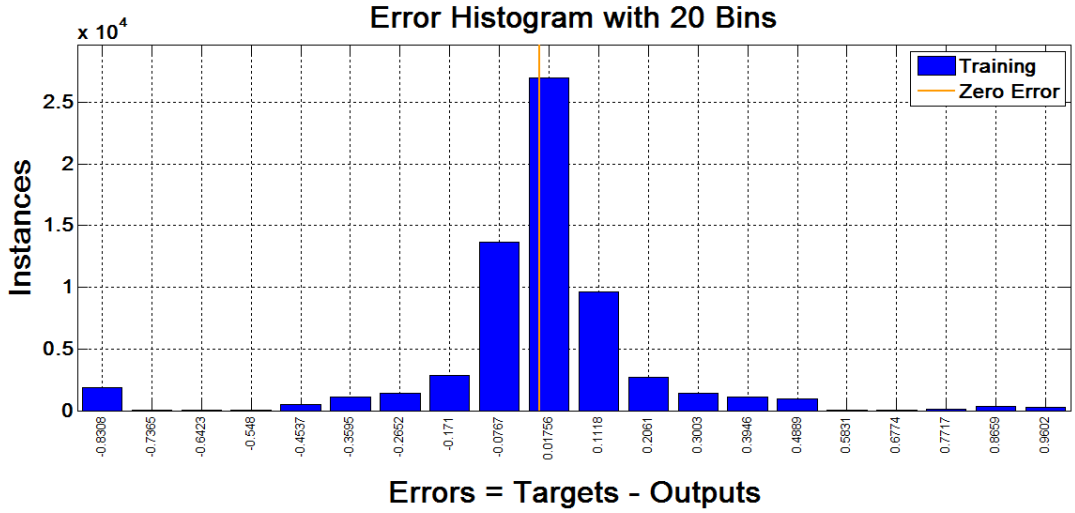

Fig. 4.5(g) Histogram plot for normalized adaptive learning algorithm

Note: *[Figures 4.5(a) to 4.5(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

Figures 4.5(a) to 4.5(g) show the error histograms with 20 bins (default value of bins) only for the training data for developed artificial neural network based prediction model. Forall the three steps of training, validation, and test in we can use '*nftool'*or make use of built-in functions. As it is shown in the figures above, the zero error is illustrated with an orange line in the middle.If the histogram goes high in middle near the zero line it means a model with more accuracy. We are getting very good histogram plots for the proposed method as the bar is high near zero error line.

4.4 Graphs for Conjugate Variations

Conjugate gradient based methods are second order training methods and researchers have developed a large number of variations for conjugate based methods. Another set of data has been tested only for conjugate based variations and a comparative table of results observed has been presented below.Since second order methods are more accurate than first order methods therefore a target value of minimum error gradient '1e-05' was set for all the training algorithms. All first order methods failed to achieve the set target of minimum error, second order conjugate algorithms have partially achieved the set target and scaled conjugate gradient was able to converge completely toward the set target for the specified network configuration.

Table 4.2 Experiment results of employing different variations of second order conjugate gradient based learning algorithms

Training Algorithm	Training function	Min. gradient	Neurons in hidden layer	Final epochs	Training time	Training performance	Starting gradient value	Final gradient value
Conjugate gradient (Polak–Ribiere update)	traincgp	1e-05	20	97	0:06:24	.0374	0.514	.01280
Conjugate gradient (Fletcher–Reeves update)	traincgf	1e-05	20	62	0:05:16	.0401	0.634	.00255
Conjugate gradient (Powell-Beale update)	traincgb	1e-05	20	39	0:02:52	.0400	0.433	.00301
Scaled conjugate gradient	trainscg	1e-05	20	517	0:24:21	.0149	.716	9.77e-06

It is clear from the table 4.2 shown above that scaled conjugate method is showing the best results.

4.4.1 Performance Graphs for Conjugate Method Variations

Training performance for conjugate gradient methods based upon mean squared error (MSE) has been presented in the following figures. As shown in Figures 4.6(a) to 4.6(g), performance has been observed to analyze the respective convergence and behavior of different conjugate based methods to achieve the minimum of error gradient. Following figures demonstrate MSE vs. numbers of epochs plot during the training of network with conjugate algorithms.

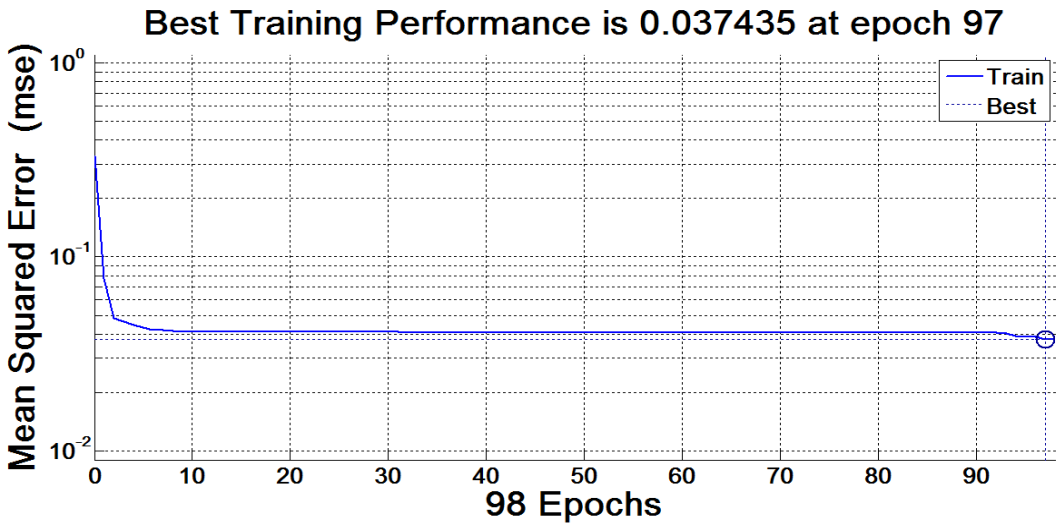

Fig. 4.6(a) Performance plot for conjugate gradient(Polak–Ribiere update)

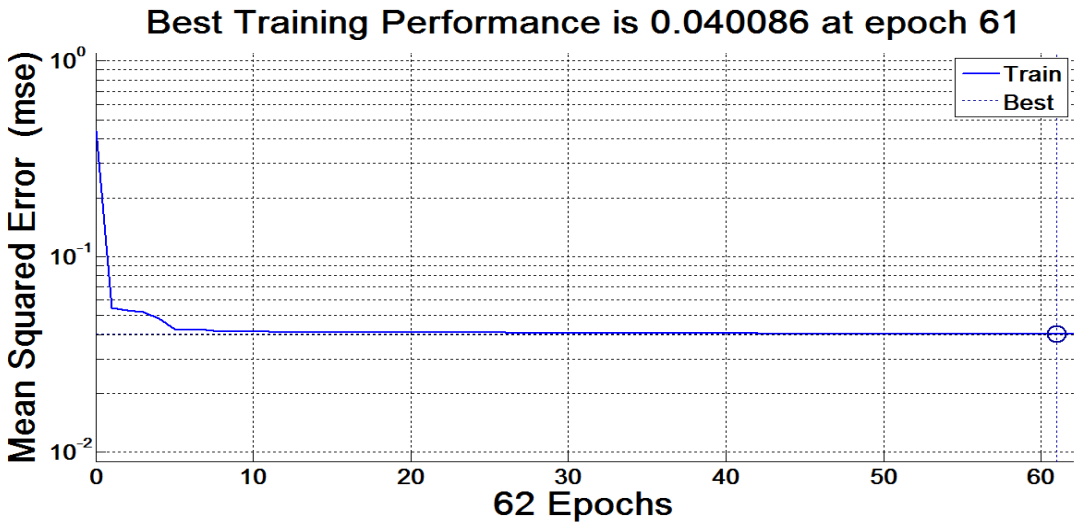

Fig. 4.6(b) Performance plot for conjugate gradient(Fletcher–Reeves update)

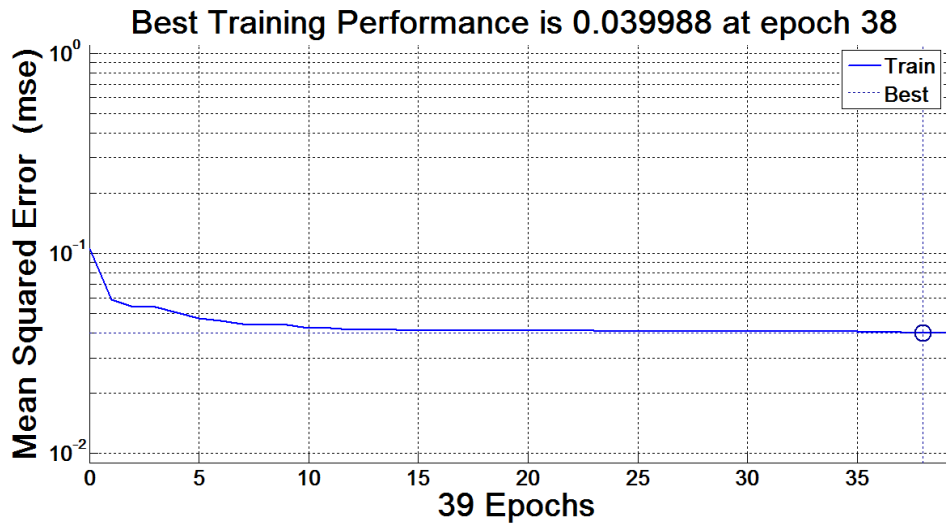

Fig. 4.6(c) Performance plot for conjugate gradient(Powell-Beale update)

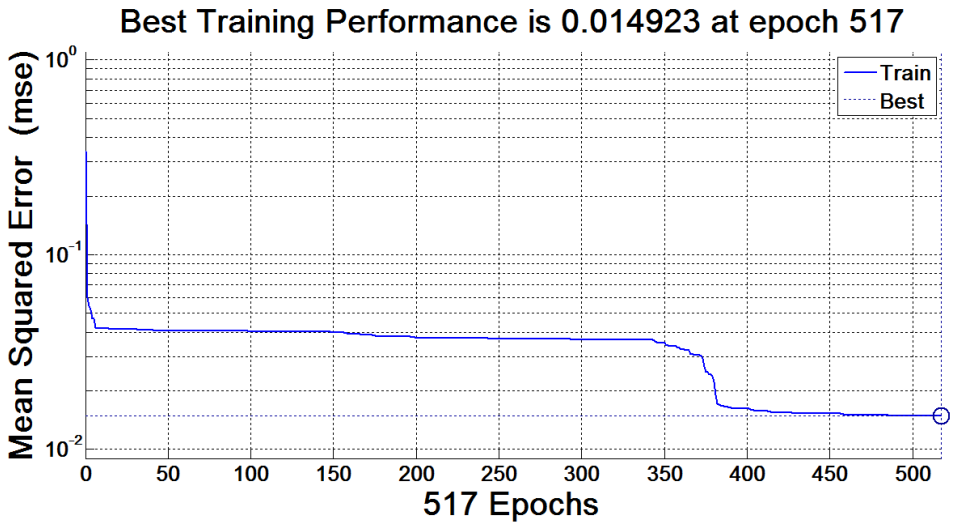

Fig. 4.6(d) Performance plot forscaled conjugate gradient

The performance graphs for 20 neurons in the hidden layer and error gradient target of 10^{-5} has been shown above for all the conjugate based variations. SCGM (scaled conjugate gradient method) has shown the best MSE performance value of 0.0149 but taken slightly more time and more number of epochs (517 epochs) to reach at the final solution.

4.4.2 Error Gradient Graphs for Conjugate Method Variations

In the following figures from 4.7(a) to 4.7(g), error gradient plots for conjugate method variations have been shown. The graph demonstrates the plot of error gradient values vs. number of epochs. Graphs clearly show how the error gradient varies with

the number of epochs when conjugate based variations are applied to train the network models along with back propagation learning.

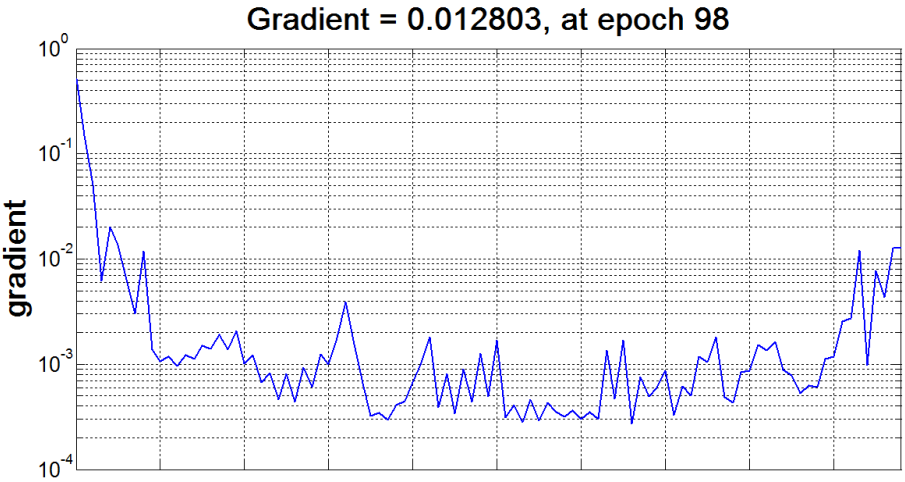

Fig. 4.7(a) Error gradient plot for conjugate gradient(Polak–Ribiere update)

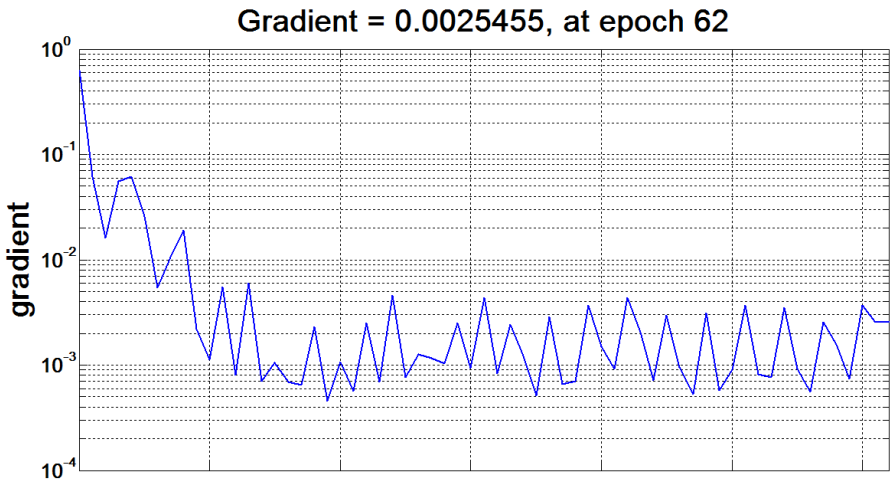

Fig. 4.7(b) Error gradient plot for conjugate gradient(Fletcher–Reeves update)

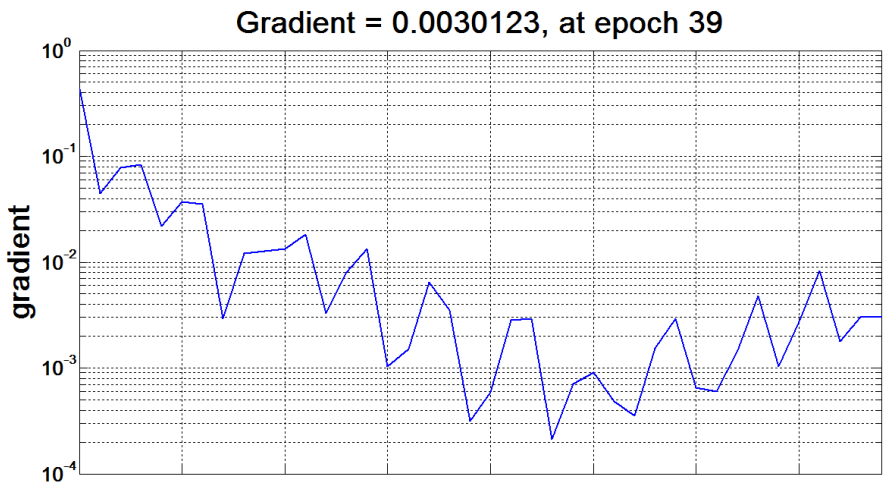

Fig. 4.7(c) Error gradient plot for conjugate gradient(Powell-Beale update)

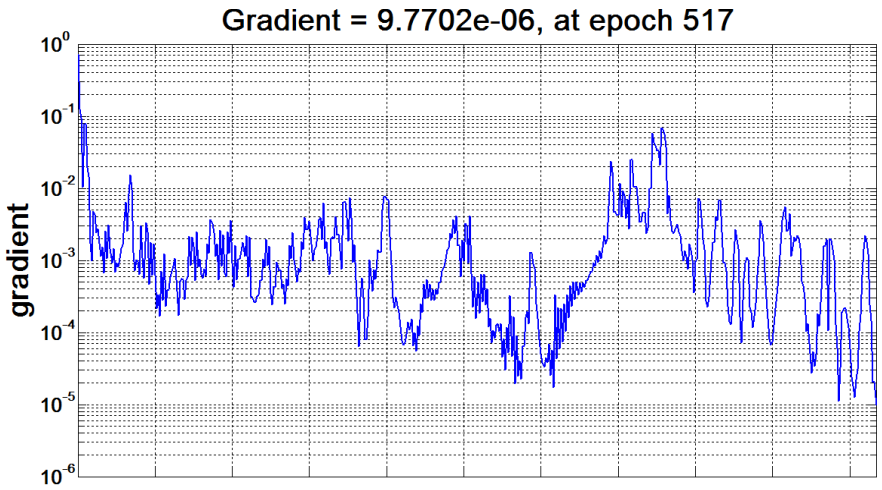

Fig. 4.7(d) Error gradient plot forscaled conjugate gradient

The error gradient graphs for 20 neurons in the hidden layer and error gradient target of 10^{-5} has been shown above for all the conjugate based variations. SCGM (scaled conjugate gradient method) has shown the best minimum error gradient value of 9.77e-06 but taken slightly more time and more number of epochs (517 epochs) to reach at the final solution.

4.5Employment of the Proposed Model for Predicting New Instances

Once the prediction model based on neural network has been trained by any suitable training algorithm and has been verified then it is ready for prediction and employment in a decision support system. The model is presented as a network object with optimal values of weights which have been tuned perfectly for the historical data. Now it is generally assumed that in similar kind of situations similar kind of results will be obtained. We can test the model for the new data sets as shown below.

[Predicted output value] = net ([New predictor values)]

Y=f(x) or output=net(input)

The test can be performed on a single set of inputs or a large number of inputs stored in form of a matrix. Figures 4.8(a) and 4.8(b) shown below present the snapshot for employing the prediction model for the new data values.

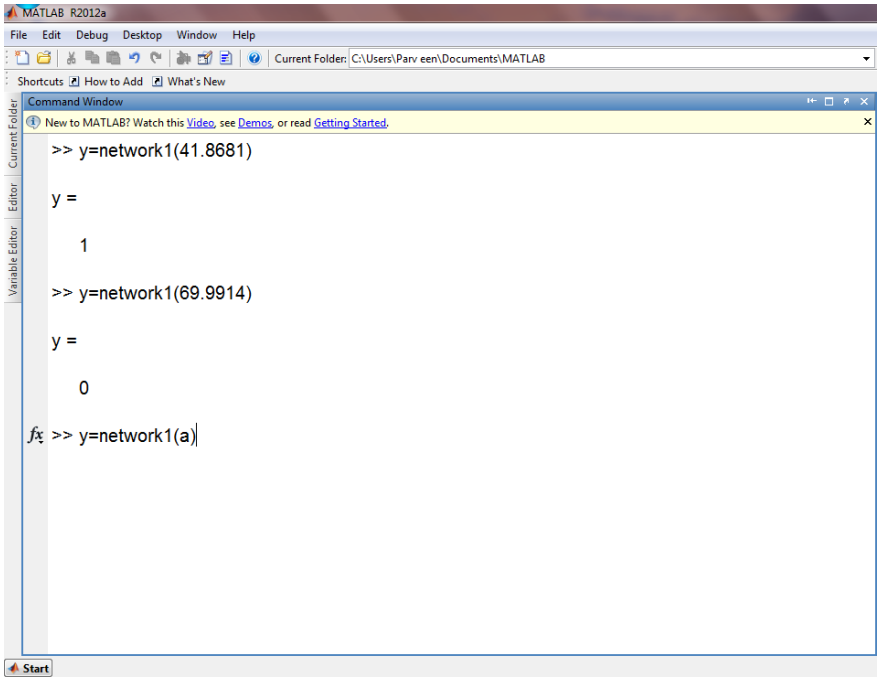

Fig. 4.8(a) Snapshot for employing the developed neural network model for predicting the new instances

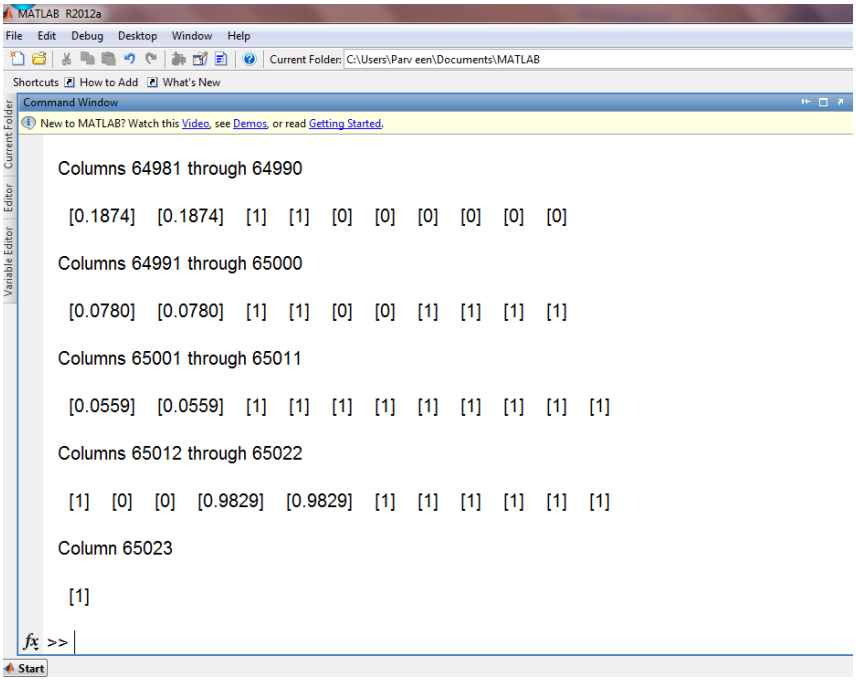

Fig. 4.8 (b)Snapshot of vectored output for predicted values for an input vector

Note: *[Figures 4.8(a) to 4.8(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

4.6Checking the Prediction Accuracyand Other Plots

It is clear from the above results, performance plots, gradient plots, histogram charts and regression curves that we are getting very good performance values for the developed models especially for the second order and the proposed algorithm. But still there are some other kind of statistical checks and plot that can be used to test the predictive accuracy or the performance of the developed models. Other checks like ROC curves confusion matrix, sensitivity, specificity statistical test etc. can be employed but more suitable for discrete cases and more suitable for the classification not for prediction which comes under continuous category. Models can be tested for individual inputs or can also be tested for a vector input to generate all possible outputs. New models have been used with the new datasets and their accuracy has been tested.

However ROC curves and confusion matrices for the applied algorithms have been presented here.

4.6.1 ROC Curves

An Receiver operating characteristic (ROC) curve is the most commonly used way to visualize the performance of a classifier or predictor, and AUC (Area under curve) is a way to summarize its performance in a single number. As such, gaining a good understanding of ROC curves and AUC is beneficial for machine learning practitioners, data scientists and medical researchers.This is a commonly used plot that summarizes the performance of a classifier over all possible thresholds. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class. ROC (*receiver operating characteristic)* metricis used to test the quality of classifiers. For each threshold, two values are calculated, the True Positive Ratio (TPR) and the False Positive Ratio (FPR). For a particular class *i*, TPR is the number of outputs whose actual and predicted class is class *i*, divided by the number of outputs whose predicted class is class *i*. FPR is the number of outputs whose actual class is not class *i*, but predicted class is class *i*, divided by the number of outputs whose predicted class is not class *i*.

We can visualize the results of this function with '*plotroc'*.

Syntax: *plotroc(targets,outputs) or plotroc(targets1,outputs2,'name1',...)*

Description: *plotroc(targets,outputs)* plots the ROC receiver operating characteristic foreach output class. The more each curve hugs the left and top edges of the plot, the better is classification.*plotroc(targets1,outputs2,'name1',...)* generates multiple plots.

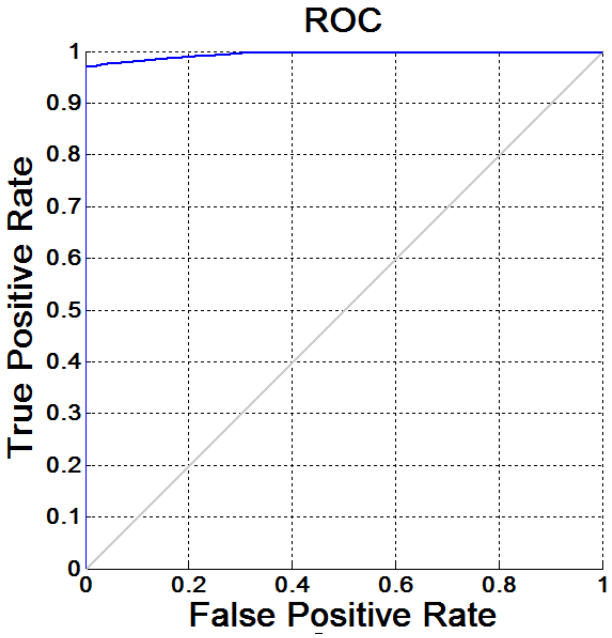

Fig. 4.9(a) ROC curve for Levenberg Marquardetalgorithm

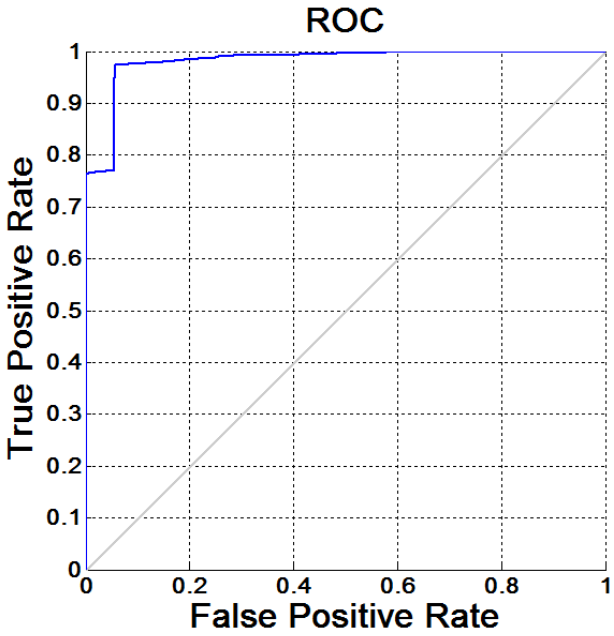

Fig. 4.9(b) ROC curve for normalized adaptive algorithm

Note: *[Figures 4.9(a) to 4.9(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

True Positive Rate: When it's actually yes, how often does it predict yes?

TP/actual yes = 100/105 = 0.95 also known as '*Sensitivity'* or '*Recall'*

False Positive Rate: When it's actually no, how often does it predict yes?

FP/actual no = 10/60 = 0.17

Figures 4.9(a) and 4.9(b) show the ROC curves for the developed models with Levenberg Marquardet and the proposed normalized algorithms.

We can clearly observe from the ROC curves that models developed for prediction with applied algorithms exhibit a good predictive accuracy. Because when ROC curve hugs the left and top edges of the plot, then better the prediction and classification capability.Accuracy can be measured by the area under the ROC curve (AUC). An area of 1 represents a perfect test; an area of 0.5 represents a worthless test. A rough guide for marking the accuracy of a diagnostic test according to AUC is that .90 to 1 means an excellent system, which is true for our developed models with the proposed algorithm.

4.6.2 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a predictionor classification model on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology is a bit confusing.

Syntax: *plotroc(targets,outputs)*

Description: *plotconfusion(targets,outputs)* returns a confusion matrix plot for the target and output data in targets and outputs, respectively.

In the confusion matrix, the rows correspond to the predicted class (Output Class), and the columns show the true class (Target Class). The diagonal cells show for how many (and what percentage) of the examples the trained network correctly estimates the classes of observations. That is, it shows what percentage of the true and predicted classes match. The off diagonal cells show where the classifier has made mistakes. The column on the far right of the plot shows the accuracy for each predicted class,

while the row at the bottom of the plot shows the accuracy for each true class. The cell in the bottom right of the plot shows the overall accuracy.

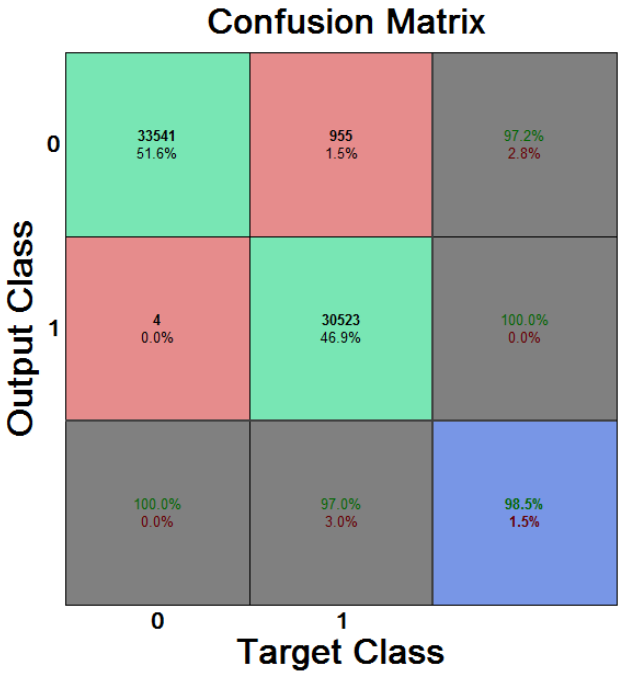

Fig. 4.10 (a) Confusion matrix for the developed model with Levenberg Marquardet algorithm

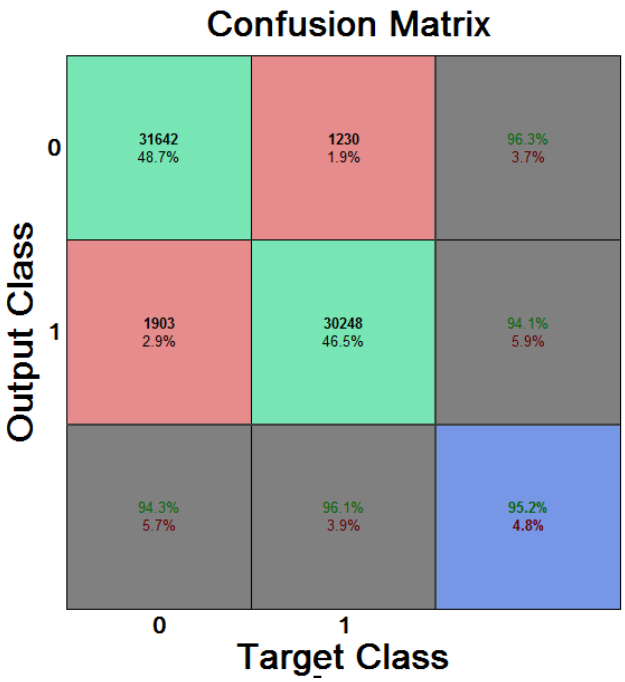

Fig. 4.10 (b) Confusion matrix for the developed model with proposed normalized adaptive algorithm

Note: *[Figures 4.10(a) to 4.10(g) have been plotted in MATLAB Neural Network Toolbox R2012a, V.7.14.0.739.]*

Figures 4.10(a) and 4.10(b) show the confusion matrix for the developed models with Levenberg Marquardet and the proposed normalized algorithms.In the figures shown above the first two diagonal cellsof light green color show the number and percentage of correct classifications by the trained network. For example, for the proposed algorithm based model, 31642(48.7%) cases are correctly predicted in one class and similarly 30248(46.5%) cases have been correctly predicted in the other class. This corresponds to 95.2% correct predictionsout of all 65023cases.In pink cells 1230 cases (1.9%) have been incorrectly predicted for one class by the model. Similarly 1903 (2.9%) cases for the second class have been incorrectly predicted. Thus total incorrect predictions are 4.8%.

Overall, 95.2% of the predictions are correct and 4.8% are wrong as seen in the blue cell, which shows that developed model exhibits an excellent predictive accuracy.