

Spring 2014

Big Data Analytics Using Neural networks

Chetan Sharma
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects



Part of the [Computer Sciences Commons](#)

Recommended Citation

Sharma, Chetan, "Big Data Analytics Using Neural networks" (2014). *Master's Projects*. 368.
DOI: <https://doi.org/10.31979/etd.72at-vza4>
https://scholarworks.sjsu.edu/etd_projects/368

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Big Data Analytics Using Neural networks

A Master's Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

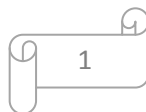
Master of Science

Advisor: Dr. Chris Tseng

by

Chetan Sharma

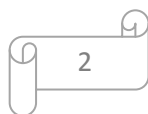
May 2014



© 2014

Chetan Sharma

ALL RIGHTS RESERVED



The Designated Master Project Committee Approves the Master's Project Titled

BIG DATA ANALYTICS USING NEURAL NETWORKS

by

Chetan Sharma

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Chris Tseng
Department of Computer Science

Date

Dr. Sami Khuri
Department of Computer Science

Date

Mr Sri Ram
Senior Manager, IBM Silicon Valley Lab

Date

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. Chris Tseng, my project advisor, for providing me with all his support and expert guidance with constant encouragement throughout the project. I would like to thank my committee members, Dr. Sami Khuri and Mr. Sri Ram, for their important suggestions. At last, I would like thank my family for their encouraging support through the course of the project.

ABSTRACT

Machine learning is a branch of artificial intelligence in which the system is made to learn from data which can be used to make predictions, real world simulations, pattern recognitions and classifications of the input data. Among the various machine learning approaches in the sub-field of data classification, neural-network methods have been found to be an useful alternatives to the statistical techniques. An artificial neural network is a mathematical model, inspired by biological neural networks, are used for modeling complex relationships between inputs and outputs or to find patterns in data. The goal of the project is to construct a system capable of analyzing and predicting the output for the evaluation dataset provided by the "IBM Watson: The Great Mind Challenge" organized by IBM Research and "InnoCentive INSTINCT (Investigating Novel Statistical Techniques to Identify Neurophysiological Correlates of Trustworthiness) : The IARPA Trustworthiness Challenge" organized by the office of The Director Of National Intelligence. The objective of this paper is to understand the machine learning using neural networks. At the end of the paper, the comparison between different learning strategies have been shown which are used to increase the accuracy of the predictions. From the trained neural network up to a satisfactory level, we will be able to classify any generalized input, process often termed as generalization capability of the learning system.

TABLE OF CONTENTS

Contents

1. Project Description:	10
2. Artificial Neural Network	11
2.1 Introduction	11
2.2 Back propagation Algorithm	12
2.3 Learning parameters	12
3. AWS Setup.....	13
4. Matlab neural network toolbox	28
5 IBM WATSON	38
5.1 Data Processing.....	38
5.1.1 Dataset Design :	38
5.1.2 Input data processing methods :.....	40
5.1.3 Output data predicted processing method	42
5.2 Implementation	43
5.2.1 Gradient descent back-propagation.....	43
5.2.2 Resilient back-propagation	45
5.2.3 Scaled conjugate gradient back-propagation.....	47
5.2.4 Momentum back-propagation	49
5.2.5 Adaptive Learning Rate back-propagation.....	51
5.2.6 Momentum and Adaptive Learning Rate back-propagation.....	52
5.2.7 Method I: Non zero vector dimensions/features only	54
5.2.8 Method II: Four times more true vectors.	55
5.2.9 Method III: Removed all the Constant columns	57
5.2.10 Method IV : Removing all the true vector constant columns	58
Comparison and Conclusion.....	59
6 INNOCENTIVE.....	60
6.1 Data Processing.....	60

6.1.1 Dataset Design :	60
6.1.2 Input data processing methods :	63
6.1.3 Output data predicted processing method	67
6.2 Implementation	69
6.2.1 Gradient descent back-propagation	69
6.2.2 Resilient back-propagation	70
6.2.3 Scaled conjugate gradient back-propagation	72
6.2.4 Momentum back-propagation	73
6.2.5 Adaptive Learning Rate back-propagation	75
6.2.6 Momentum and Adaptive Learning Rate back-propagation	76
6.2.7 Method I: Four classifications in desired output vector	78
6.2.8 Method II: Learning with two times more don't trust vectors.	80
6.2.9 Method III: Variation in threshold limit	81
6.2.10 Method IV : Focusing on reduced set of training vectors	82
6.2.11 Method V: Increasing the validation and test dataset	84
6.2.12 Method VI : Revising with important/ stringent vectors	85
6.2.13 Method VII : Excluding Noise from the network	90
6.2.14 Method VIII : Merging method VI and VII	91
Comparison and Conclusion	93
7. Future Work	95
8. References	96
9. Appendix	97
9.1 Algorithm to process the predicted output data:	97
9.2 Algorithm to prepare the training input dataset :	98
9.3 Algorithm to double the true vectors in input dataset data :	99

Figures

Figure 1: An Artificial Neural Network-----	11
Figure 2: Activation Functions -----	13
Figure 3: Transfer Function :sigmoid-----	13
Figure 4: Amazon web services sign up-----	14
Figure 5: Amazon Web Services Registration -----	14
Figure 6: AWS EC setup-----	15
Figure 7: AWS Launch New Instance-----	15
Figure 8: AWS Amazon Machine Image -----	16
Figure 9: AWS Instance Type -----	16
Figure 10: AWS Configure Instance Details -----	17
Figure 11: AWS Add Storage-----	17
Figure 12: AWS Tag Instance -----	18
Figure 13: AWS Configure Security group-----	18
Figure 14: AWS Review Launch Instance -----	19
Figure 15: AWS Key Pair -----	19
Figure 16: AWS new key pair-----	20
Figure 17 :AWS Launch Status-----	20
Figure 18: AWS Management console -----	21
Figure 19: AWS Instance Description-----	21
Figure 20: AWS Instance Monitoring-----	22
Figure 21: AWS Instance Connect -----	22
Figure 22: AWS Instance Connection details -----	23
Figure 23: AWS Instance Putty Connection -----	23
Figure 24: AWS Instance Putty Private Key Setup -----	24
Figure 25: AWS Load .pem private key -----	24
Figure 26: AWS Rendering .PEM file -----	25
Figure 27: AWS Save private key-----	25
Figure 28: AWS .PPK file generation -----	26
Figure 29: AWS Putty Private Key Setup -----	26
Figure 30: AWS Connected using putty -----	27
Figure 31: Matlab: Sample Dataset -----	28
Figure 32: Matlab: Neural Network toolbox-----	28
Figure 33: Matlab: Pattern Recognition tool -----	29
Figure 34: Matlab: select Data-----	29
Figure 35: Matlab: Validation and test data -----	30
Figure 36: Matlab: Network Architecture -----	30
Figure 37: Matlab: Train Network -----	31
Figure 38: Matlab: Neural Network Training phase-----	32
Figure 39: Matlab: NN Training completed -----	33

Figure 40: Matlab: Retrain Network -----	34
Figure 41: Matlab: Evaluate Network -----	34
Figure 42: Matlab: Save results -----	35
Figure 43: Matlab: Current Workspace -----	35
Figure 44: Matlab: Saving workspace -----	36
Figure 45: Matlab: Load Workspace -----	36
Figure 46: Matlab: Predicting Output-----	37
Figure 47: Training Dataset -----	38
Figure 48: Evaluation Dataset -----	39
Figure 49: True Vector Id's -----	42
Figure 50: Back-propagation-----	43
Figure 51: Gradient descent back-propagation Performance Graph -----	44
Figure 52: Resilient back-propagation Performance Graph -----	47
Figure 53: Scaled conjugate gradient back-propagation Performance Graph-----	48
Figure 54: Momentum back-propagation Performance Graph -----	50
Figure 55: Adaptive Learning Rate back-propagation Performnace Graph -----	52
Figure 56: Momentum and Adaptive Learning Rate Performance Graph-----	54
Figure 57: Zero vector dimensions/features Performance Graph -----	55
Figure 58: Four times more true vectors Performance Graph-----	56
Figure 59: Removed all the Constant columns -----	57
Figure 60: Removing all the true vector constant columns -----	58
Figure 61: Comparison and Conclusion Grpah -----	59
Figure 62: Training Dataset -----	62
Figure 63: Evaluation Dataset -----	63
Figure 64: Input data processing methods-----	64
Figure 65: Process the evaluation data-----	66
Figure 66: Predicted Vector Id's-----	67
Figure 67: Gradient descent back-propagation Performance Graph -----	70
Figure 68: Resilient back-propagation Performance Graph -----	71
Figure 69: Scaled conjugate gradient back-propagation Performance Graph-----	73
Figure 70: Momentum back-propagation Performance Graph -----	74
Figure 71: Adaptive Learning Rate back-propagation Performnace Graph -----	76
Figure 72: Momentum and Adaptive Learning Rate Performance Graph -----	77
Figure 73: Zero vector dimensions/features Performance Graph -----	79
Figure 74: Four times more true vectors Performance Graph-----	80
Figure 75: Focusing on reduced set of training vectors-----	84
Figure 76: Revising with important/ stringent vectors scenario 1 -----	88
Figure 77: Revising with important/ stringent vectors scenario 2-----	89
Figure 78: Excluding Noise from the network -----	91
Figure 79: Merging method VI and VII-----	92

1. Project Description:

The goal of the project is to construct a system capable of analyzing and predicting output for the large scale applications by efficiently handling the data consisting of very high dimensionality. Experiments using different learning strategies and data preprocessing methods will be carried out to increase the accuracy of the predictions.

The project will serve the part of the "IBM Watson, The Great Mind Challenge" technical intercollegiate competition and InnoCentive INSTINCT: The IARPA (Investigating Novel Statistical Techniques to Identify Neurophysiological Correlates of Trustworthiness)

Trustworthiness Challenge. The objective of the project is to server two purpose, one is to generate an neural network predicting results as close as possible to the ones which would have been generated by the, one of the most intelligent supercomputer in the world, IBM Watson for the IBM challenge and second is to generate a neural network capable to most accurately predict whether to trust or do not trust a person based on its biological and physical behavior. Therefore, the generated system will be validated with the cutting edge technology in today's world of machine learning, IBM Watson dataset and with the nationwide open challenge competition with a prize money of \$25000.

From the trained system up to a satisfactory level, we will be able to classify any generalized input , process often termed as generalization capability of the learning system.

2. Artificial Neural Network

2.1 Introduction

Artificial neural networks natural metaphor representation is a “Brain” of an individual.

The basic concept in machine learning using neural networks is based on the learning process of a living thing i.e. how we are able to learn new things with the help of our experience since childhood. An artificial neural network can be defined as a computing system made up of number of simple highly interconnected processing elements which processes information by their dynamic state response to external inputs. Information processing is carried out through connectionist approach to computation. An example of such a neural network appears in figure 2.1.

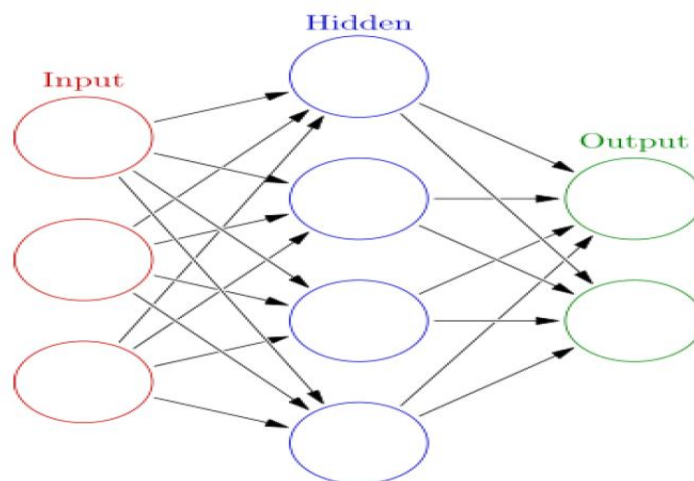


Figure 1: An Artificial Neural Network

Here, there can be any number of input, hidden and output layers connected in the network. In simplest terms, neural network, initially, makes random guesses and sees how far its answers are from the actual answers and makes an appropriate adjustment to its node-connection weights.

2.2 Back propagation Algorithm

Back propagation algorithm is used to train the artificial neural network described in previous section. It helps achieve desired outputs from provided inputs under supervised learning

2.3 Learning parameters

Learning rate : Learning rate is the rate at which we want neural network to learn training function. Keeping it too small, neural network convergence to desired results becomes extremely slow. Keeping it too large, neural network may not converge at all. Thus, it is very important to choose learning rate properly.

Hidden layer neurons : According to the Kolmogorov equation, any given continuous function can be implemented exactly by a 3-layer neural network with n neurons in the input layer, $2n+1$ neurons in the hidden layer and m neurons in the output layer.

Weight and biases : They are used to reduce the difference between actual and desired output. This kind of learning is termed as supervised learning in which neural network is feed with inputs and corresponding desired outputs.

Activation Function : The activation function of a node defines the output of that node given an input or set of inputs. In its simplest form, this function is binary—that is, either the neuron is firing or not.

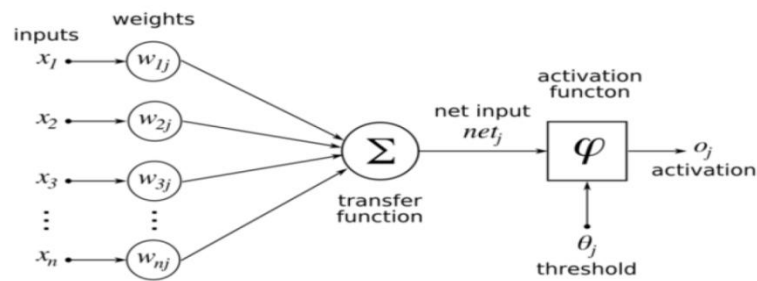


Figure 2: Activation Functions

Transfer Function :It calculate a layer's output from its net input. The sigmoid function introduces non-linearity in the network. Without it, the net can only learn functions which are linear combinations of its inputs

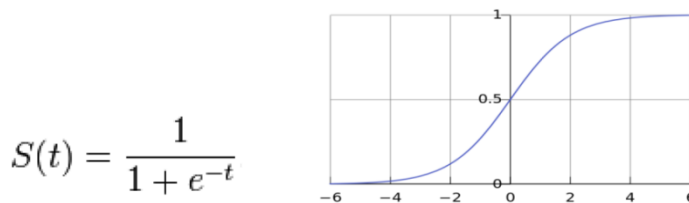


Figure 3: Transfer Function :sigmoid

3. AWS Setup

The project was implemented over the Amazon web services. following are the steps explaining the AWS setup procedure.

Go to the link : <http://aws.amazon.com/console/>. it will display the startup page of the AWS. To work on Amazon cloud we require to register to the website. Click on the sign in button as shown in the figure to proceed with the login.

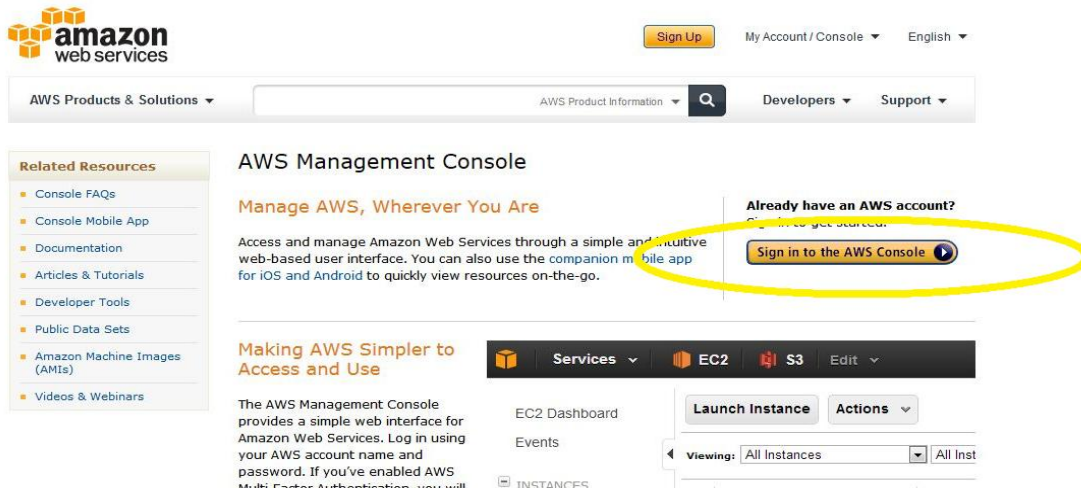


Figure 4: Amazon web services sign up

If you are signing for the first time for the Amazon cloud services, choose the first option which will take you to the registration page. after successful sign up, come back to this page and proceed with the second option to enter the AWS console.



Figure 5: Amazon Web Services Registration

there are multiple services offered by Amazon. we require the elastic computing as a need for our project. select the EC option as shown in figure.

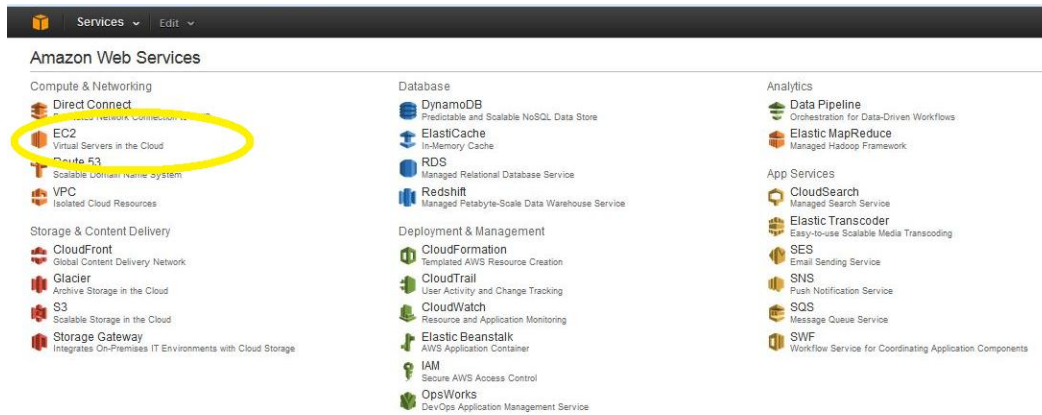


Figure 6: AWS EC setup

Click on the create new instance button for starting a fresh setup.

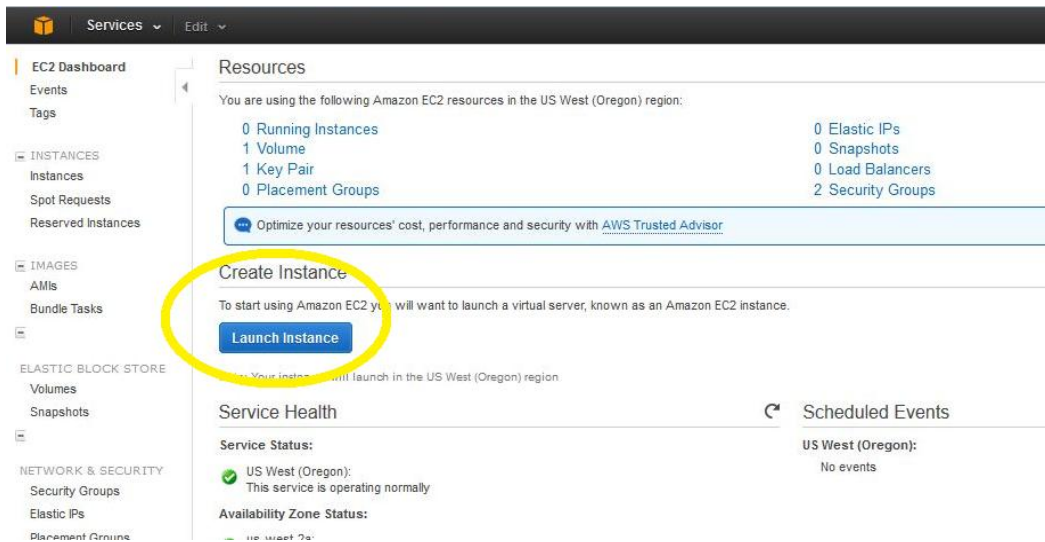


Figure 7: AWS Launch New Instance

This will take you to the process of building our machine as per the requirements . Amazon provides sever operating systems as shown below. choose the one by selecting the button on the right side of the page. It also allows us to choose among the 32 bit/64 bit options. Select ubuntu version-12 64 bit. As version 12 still the most stable release.

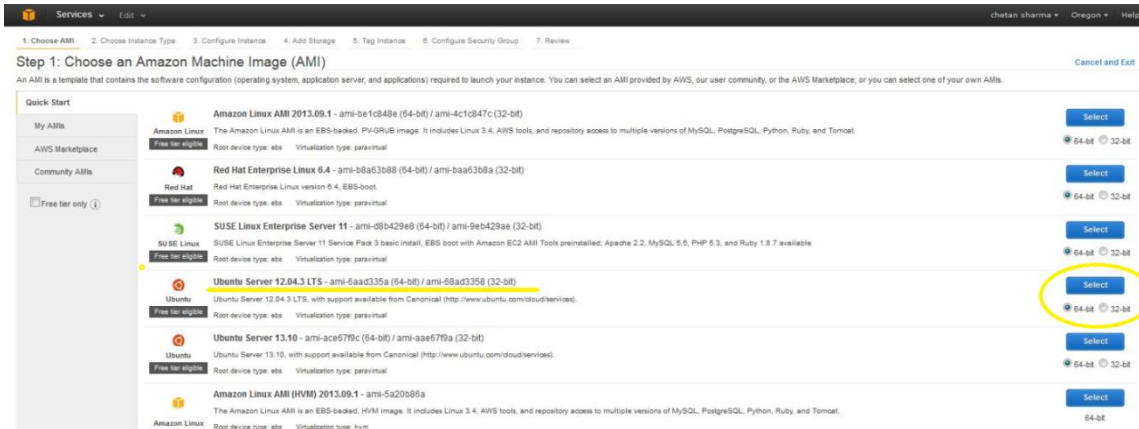


Figure 8: AWS Amazon Machine Image

The next page will provide the different computing units. they are pre configured into the sub categories of memory optimized, computation optimized and storage optimized. as we have high memory requirement . select M2.2 large. it provided sufficient computing power with enough ram to accommodate our training dataset and learning system's network size.

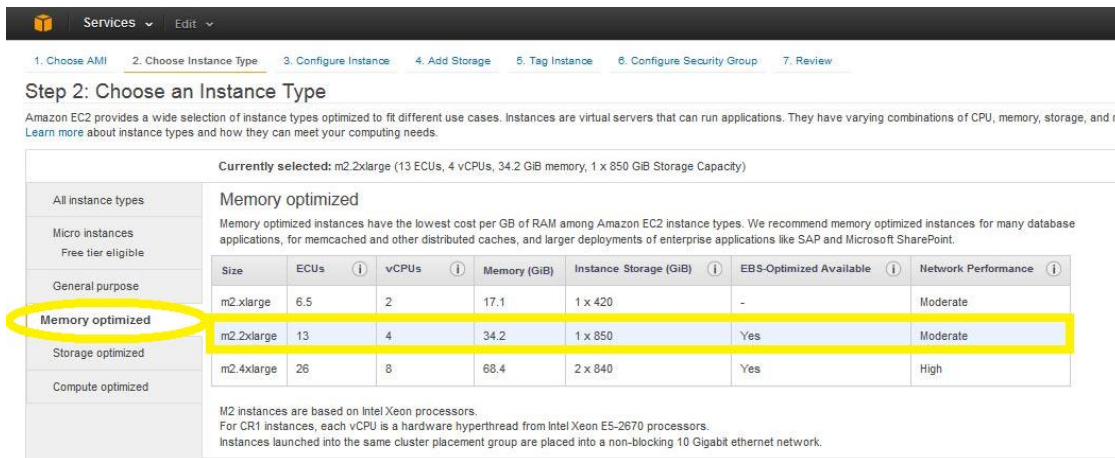


Figure 9: AWS Instance Type

The following page displays the number of instances of the machine we have selected are required to be instantiated. Select 1 and leave other configurations option as it is

and click on next button. the default setting of the configurations well suits our project requirement.

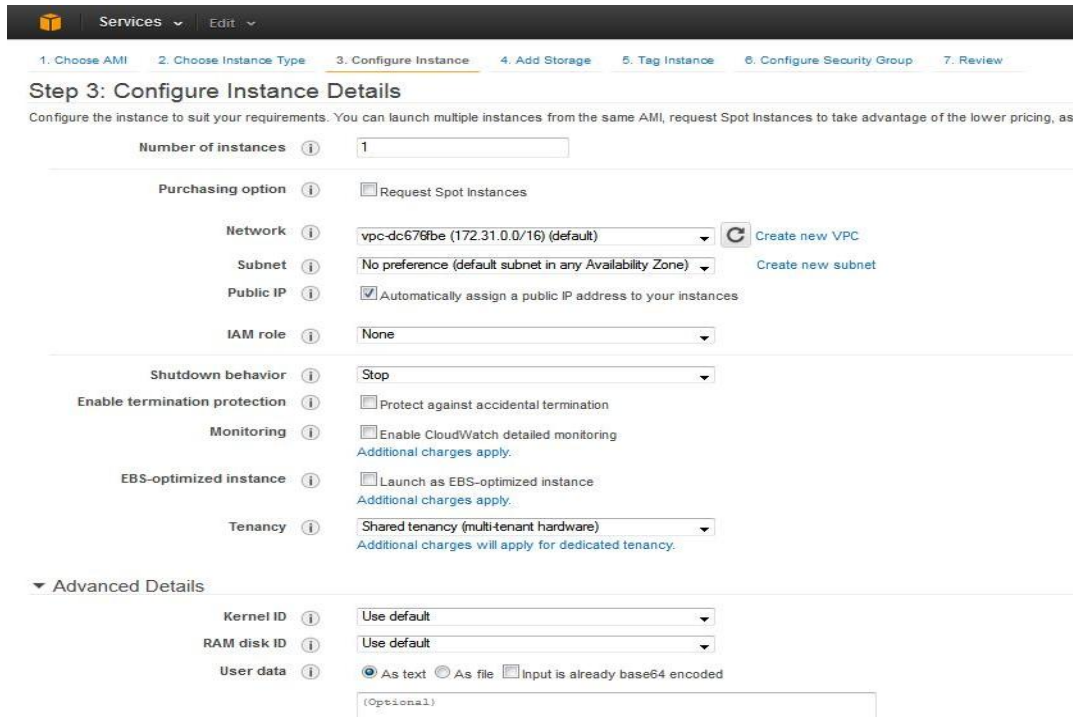


Figure 10: AWS Configure Instance Details

The amount of storage required for the system can be selected on the next page. The default option of 8 GB is enough for the project.

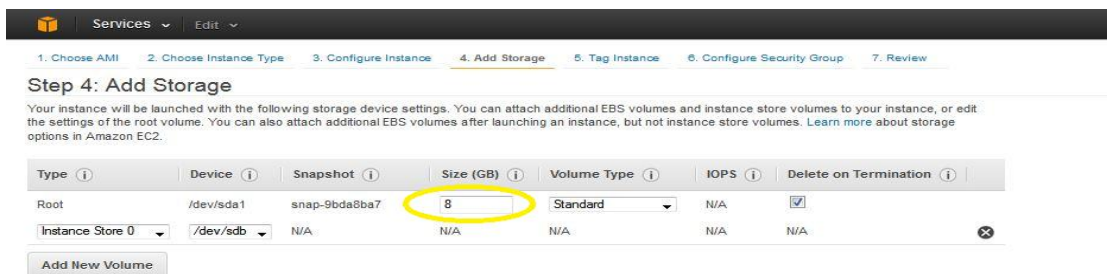


Figure 11: AWS Add Storage

We can provide a name to our machine. It is helpful to distinguish when we have multiple machines running on cloud.



Figure 12: AWS Tag Instance

The following page provides the option to configure the network of the machine. We strictly require port 22 to be open for the SSH protocol. as we can only access the machine using the SSH client as the machine will be running over cloud. Remaining ports can be selected as per choose considering in mind the security issues.

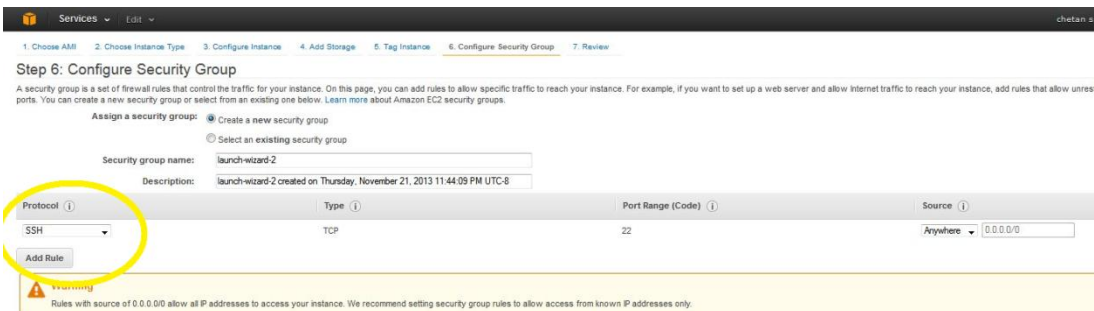


Figure 13: AWS Configure Security group

After clicking on the next button we are done with the setup of our machine. the following page will ask us to give a final review to our application before we proceed ahead.

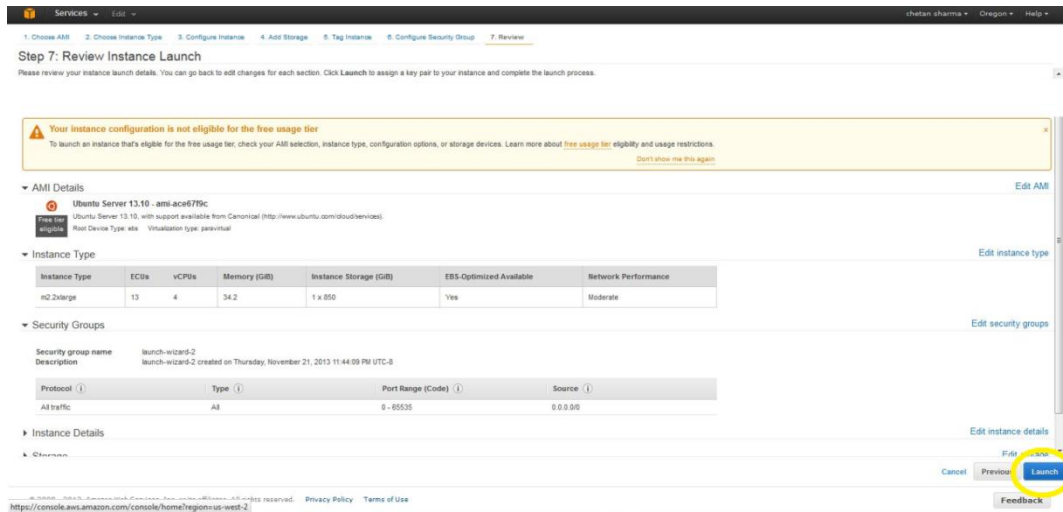


Figure 14: AWS Review Launch Instance

After reviewing the details of the machine configuration successfully. We can proceed with the launch button. As the machine will be running on the cloud ,therefore for the purpose of security it will ask us to create a new key value pair, if one doesn't exist.

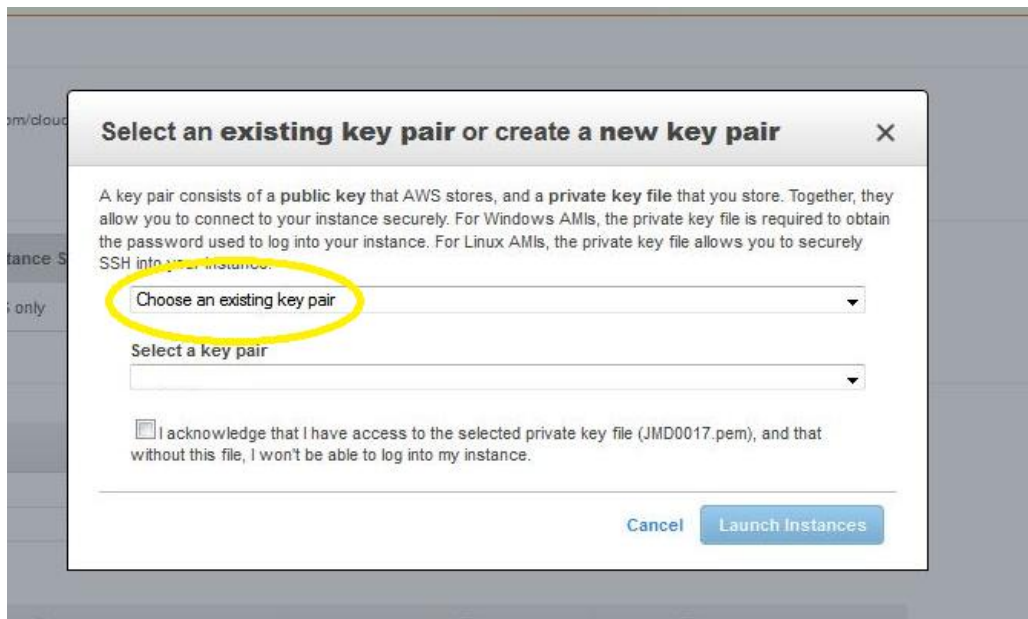


Figure 15: AWS Key Pair

Choose the first option for generating a new key value pair and provide a name to it.

A download key pair option will appear. click to download it and keep it in a safe place, as u will not be able to generate a duplicate key. Therefore, it has be keep safe for successful access to the machine.

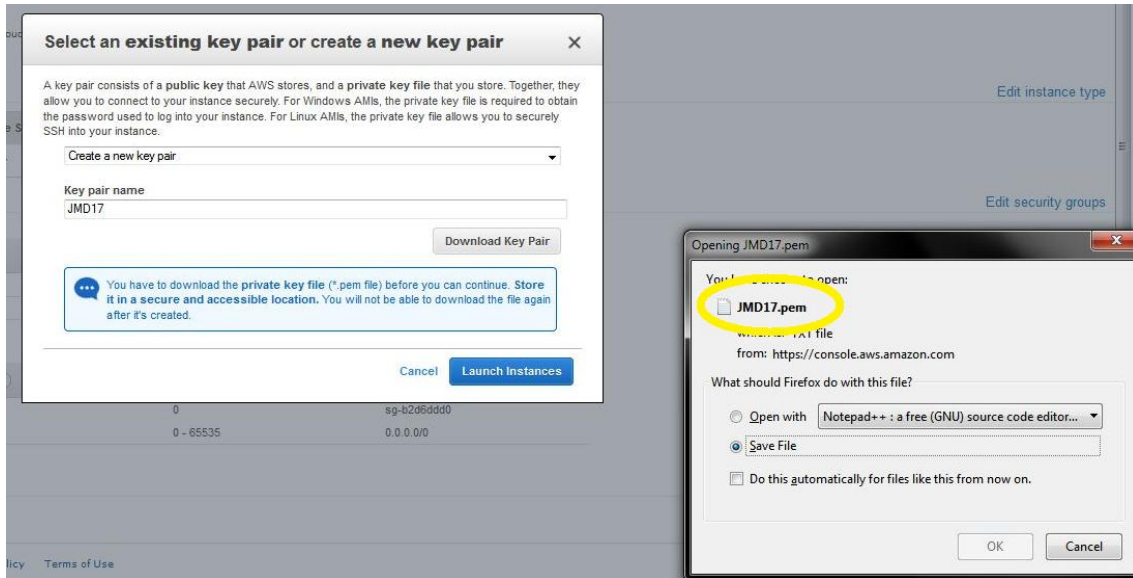


Figure 16: AWS new key pair

Click on the launch button. a successful launch page will appear after the instance has been started.

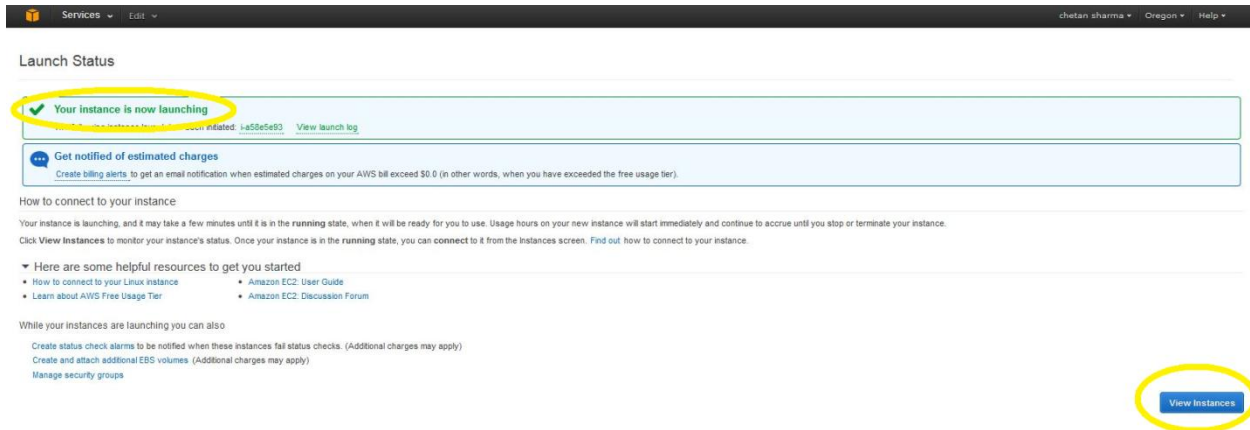


Figure 17 :AWS Launch Status

view instances button will display the console where all the machines created by this account is displayed .A particular machine information and control can be taken by selection the checkbox next to it.

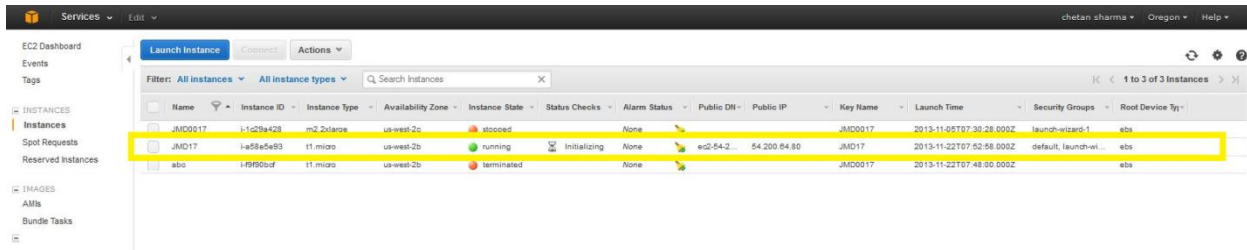


Figure 18: AWS Management console

Tabs below provide the detail configuration information for the machine selected above.

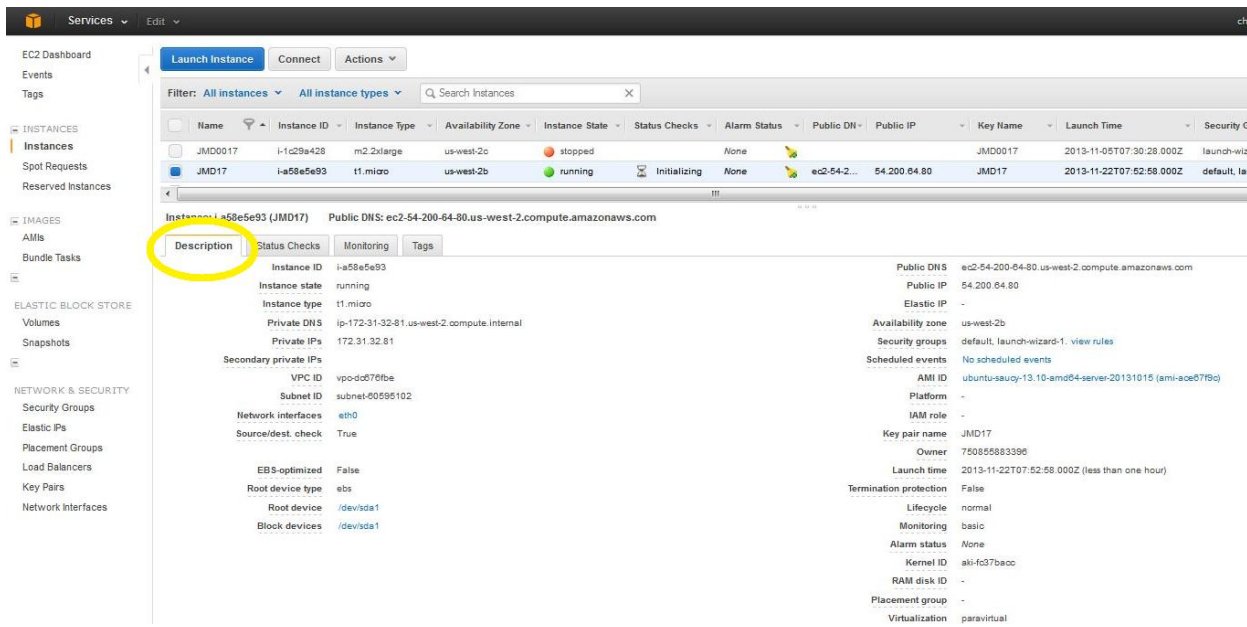


Figure 19: AWS Instance Description

Also, the system can be easily monitored by selecting the 'monitoring' tab next to the information tab. It display a series of graphs representing the current state of the system. like number of disk reads, network usage, ram usage, storage usage etc.

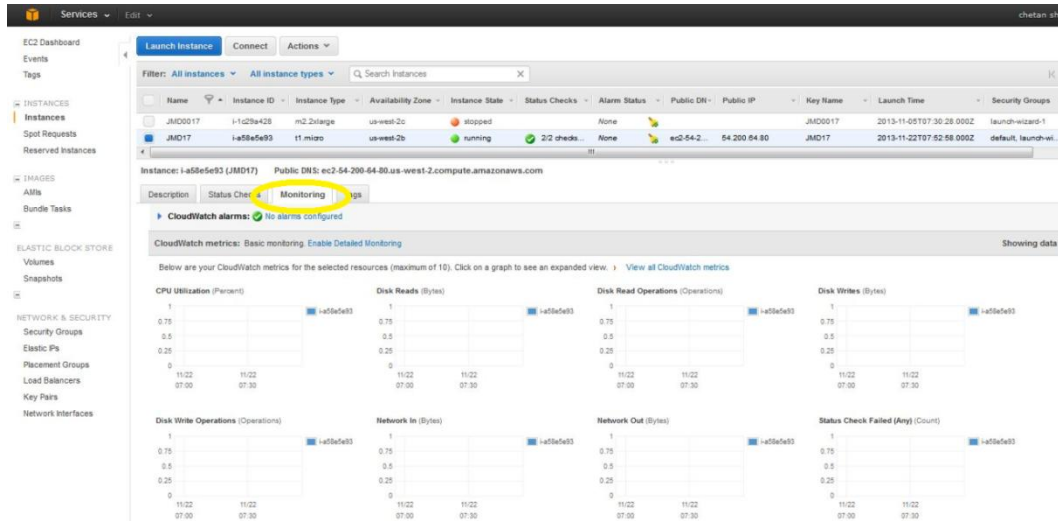


Figure 20: AWS Instance Monitoring

To start the machine, select the corresponding machine and click on connect.

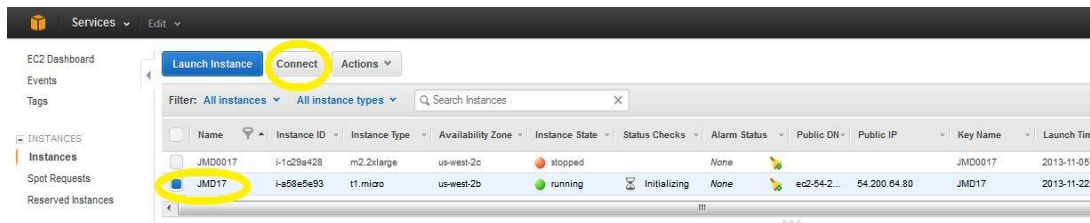


Figure 21: AWS Instance Connect

The following page will provide the information to connect to the machine using the IP address. As the ip address is assign at the time of system start, therefore there will be different IP address allocate to the machine every time we restart the instance.

There two options provided. we can either connect through the ssh client like putty, or through our web browser by selecting the second option. select the "a standalone ssh client" and copy the ip address.

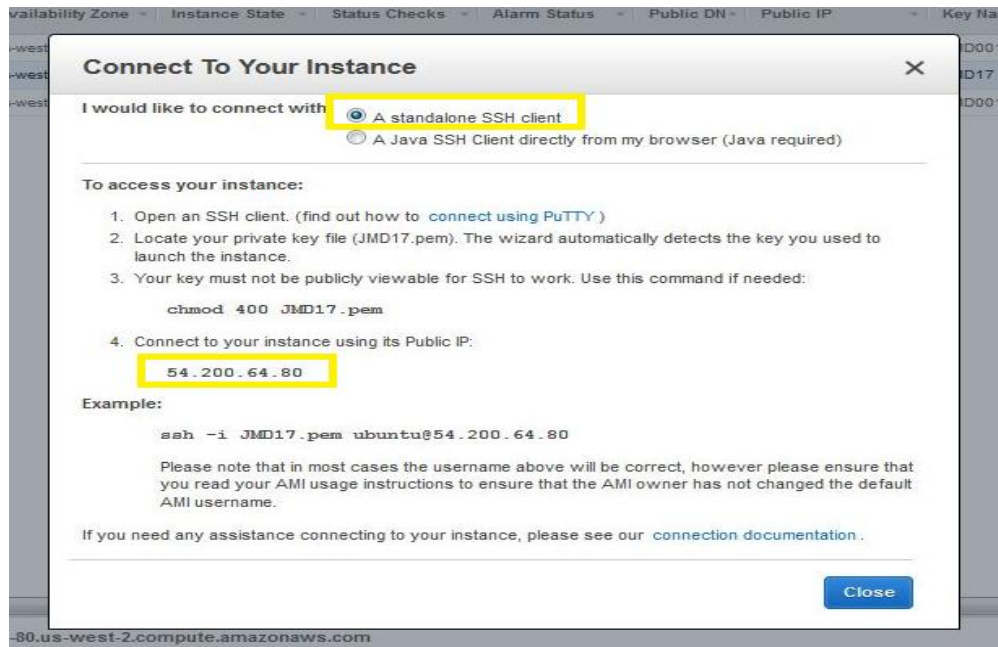


Figure 22: AWS Instance Connection details

To connect through putty. use the ip address provided. Also, as we have selected the security through key pairs, therefore path to the private key download has to be provided to the ssh client.

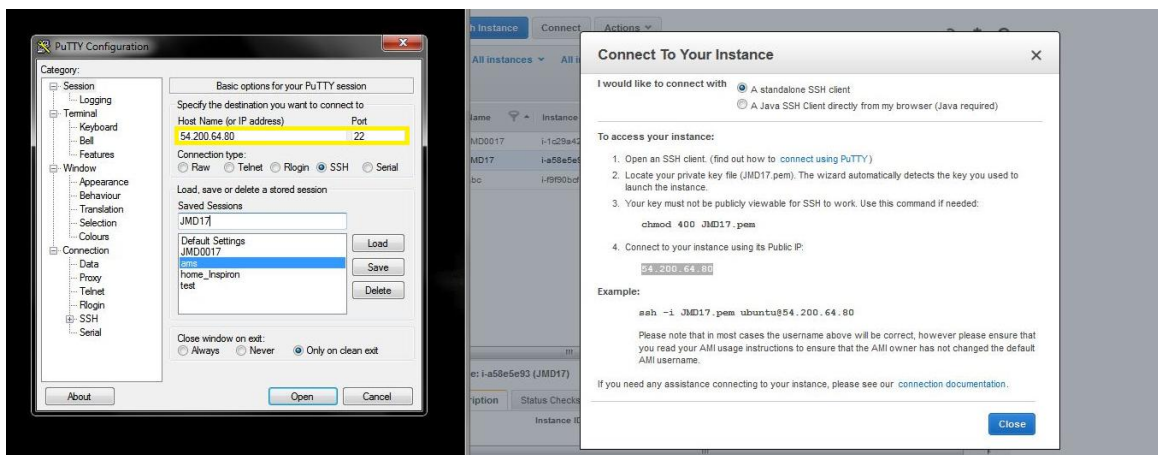


Figure 23: AWS Instance Putty Connection

Traverse through the category structure presented on the left of the putty.

Connection -> SSH-> AUTH . It will display on the right, option to provide the path of the private key for the given ip address.

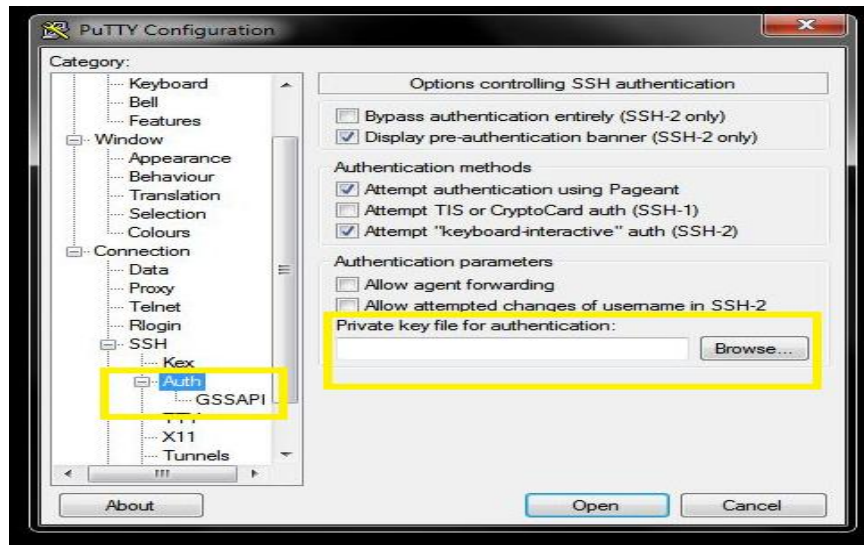


Figure 24: AWS Instance Putty Private Key Setup

As the private key generated was in the .pem format we need to download use the putty key generator to convert the private key file from .pem format the .ppk format. It can be achieved as per the following steps:

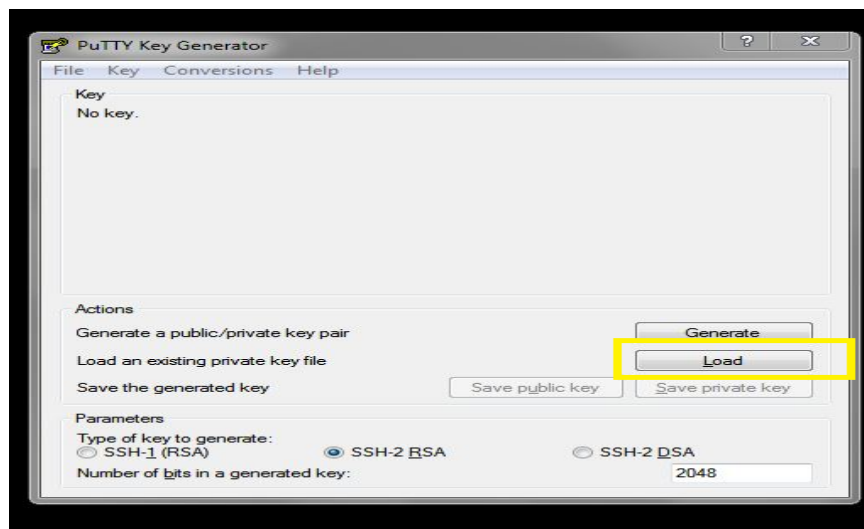


Figure 25: AWS Load .pem private key

Load the .pem private key file using the load option as shown in figure.



Figure 26: AWS Rendering .PEM file

After successfully rendering the .pem file. it will display the certificate in the text box as shown below.

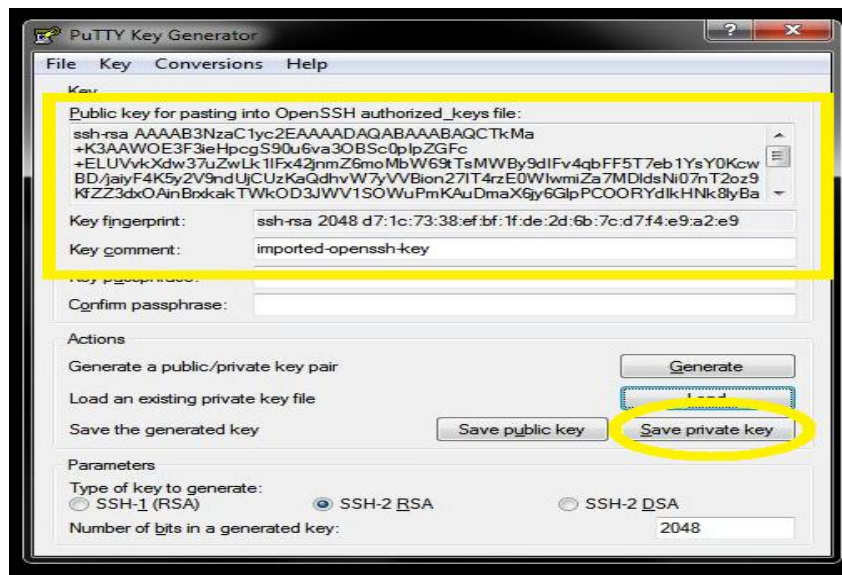


Figure 27: AWS Save private key

Now, to convert it to the .ppk format. Select the save option and choose the format as .ppk.

Select yes when prompted for permission. it will convert the .pem file and save the file in the .ppk format.



Figure 28: AWS .PPK file generation

The generate .ppk file will be load in putty and click open to connect to the machine on AWS.

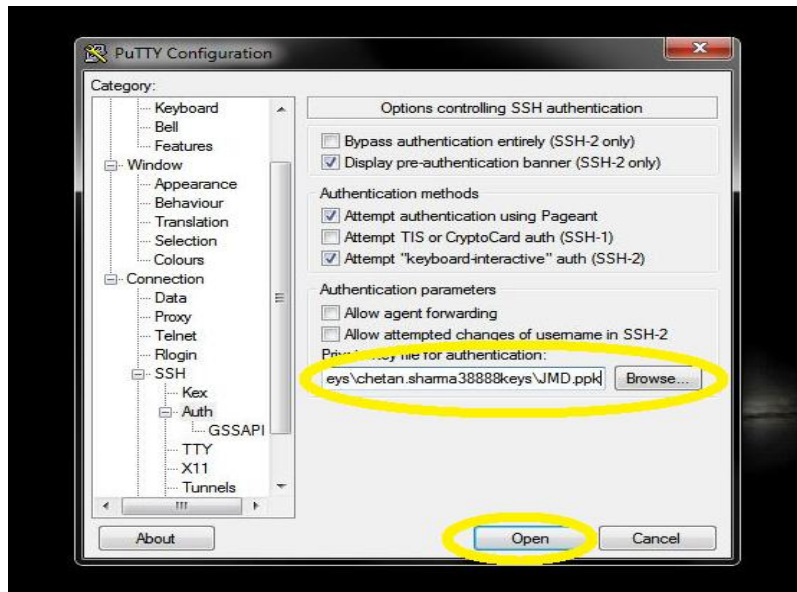
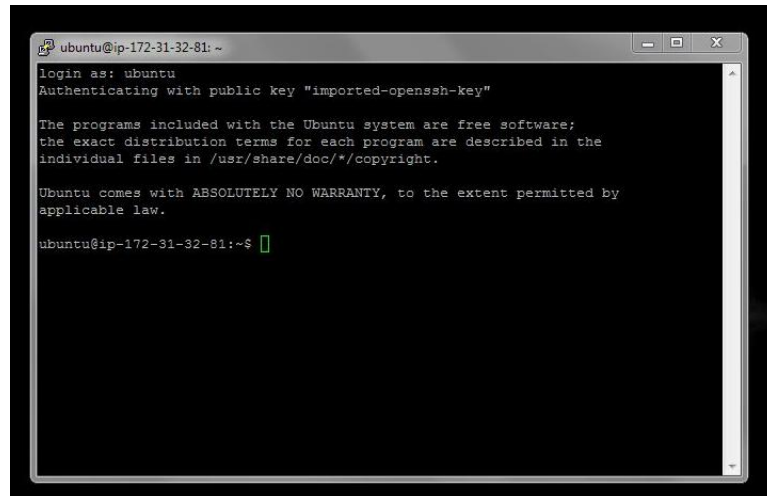


Figure 29: AWS Putty Private Key Setup

When connecting for the first time, the putty will prompt for the permission to connect for the security reasons. click yes and proceed to the login terminal of the machine.

enter the username : ubuntu

By default, the username of the ubuntu Os is set to ubuntu .The login will not ask for the password authentication as the authentication was done trough public private key pair.



```
ubuntu@ip-172-31-32-81: ~  
login as: ubuntu  
Authenticating with public key "imported-openssh-key"  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
ubuntu@ip-172-31-32-81:~$
```

Figure 30: AWS Connected using putty

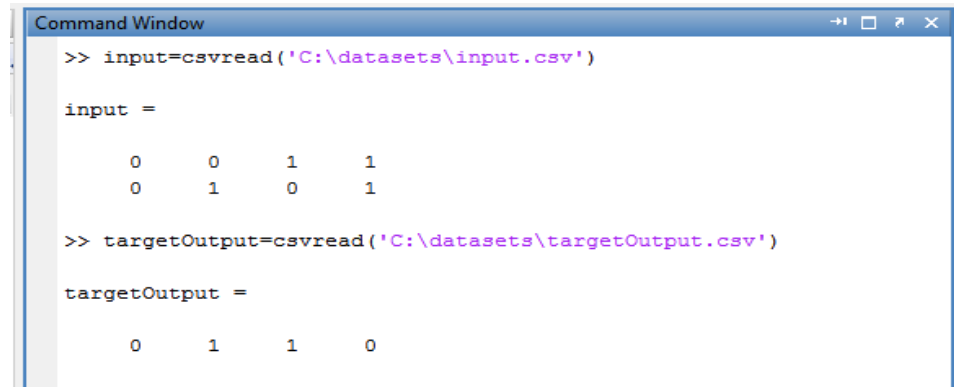
After displaying the command shell the Ubuntu Setup over AWS has been completed successfully.

4. Matlab neural network toolbox

Following are the steps to demonstrate the working of the neural network toolbox in matlab. To start with, we generate some dummy training dataset

input : matrix containing the input vectors.

targetoutput : matrix containing the corresponding desired outputs.



```
Command Window
>> input=csvread('C:\datasets\input.csv')

input =

     0     0     1     1
     0     1     0     1

>> targetOutput=csvread('C:\datasets\targetOutput.csv')

targetOutput =

     0     1     1     0
```

Figure 31: Matlab: Sample Dataset

We can start the neural network tool using the nnstart command. It will provide us with four options to select upon depending upon the type of training we are going to be required.

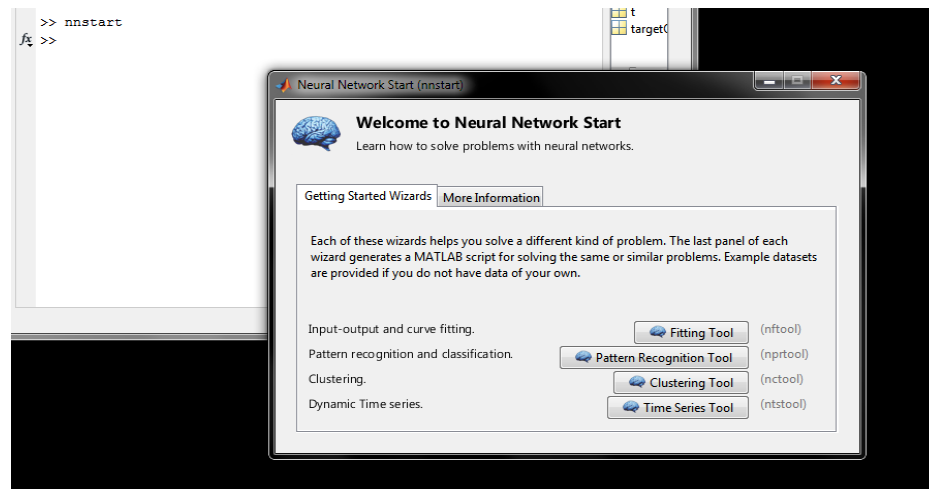


Figure 32: Matlab: Neural Network toolbox

Select the pattern recognizing tool. It will display the following window containing the information about the type of training tool we have selected. Click next to proceed with the network construction

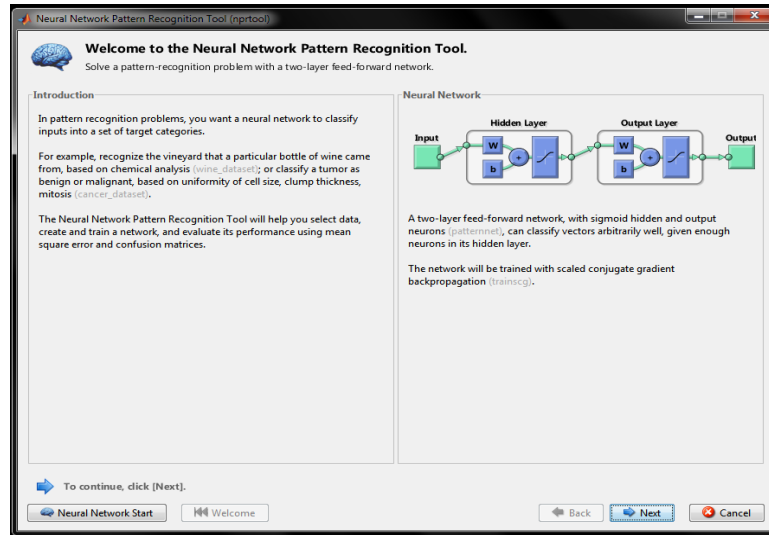


Figure 33: Matlab: Pattern Recognition tool

Select the input and target output required to train the network. From the drop down menu select the dummy input and target output matrices created earlier.

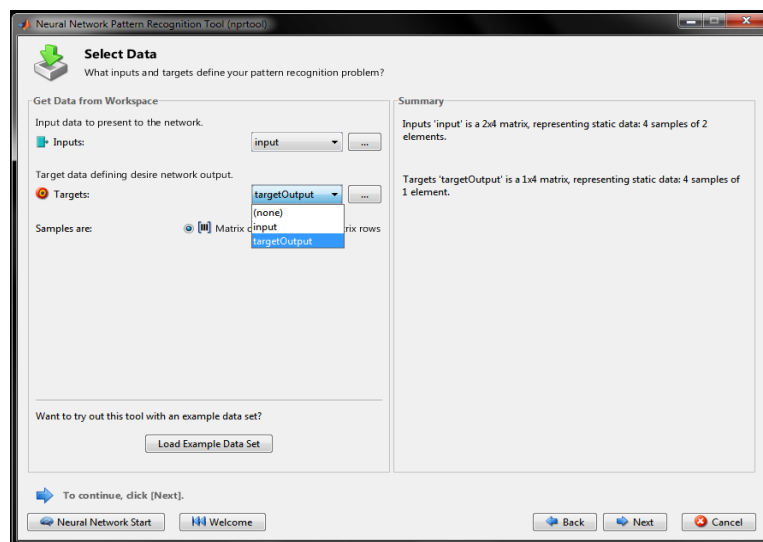


Figure 34: Matlab: select Data

The next window will provide the options to divide the training dataset into three parts. value entered randomly divide the training dataset on percentage basis.

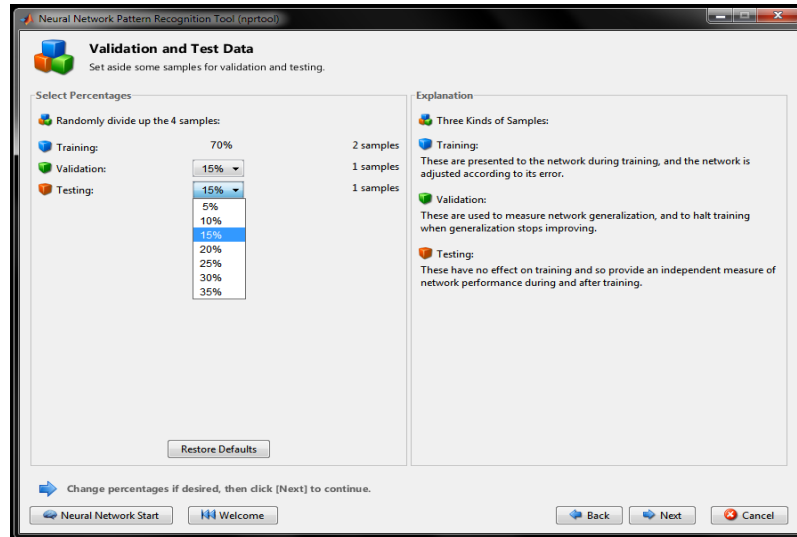


Figure 35: Matlab: Validation and test data

The following window will allow the user to enter the number of hidden neurons required for the network. The input layer and output layer neurons will be automatically set from the training dataset provided earlier.

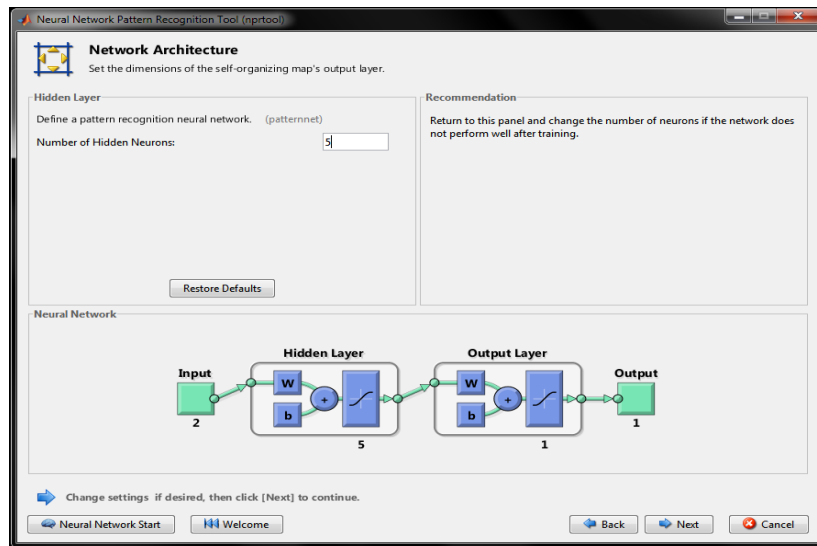


Figure 36: Matlab: Network Architecture

The neural network has been constructed successfully. Clicking on the train button will start the network training as shown in the next figure.

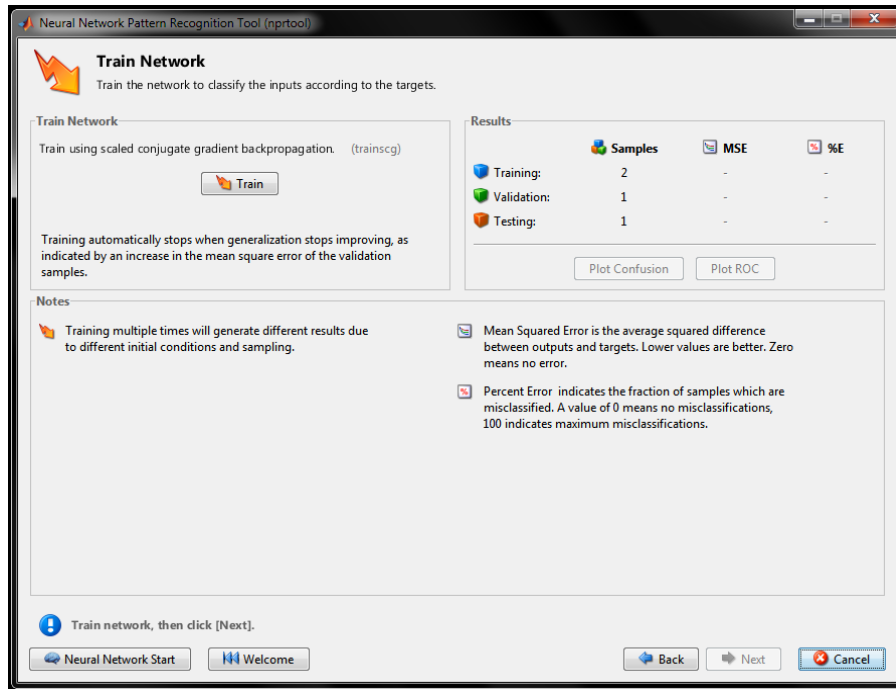


Figure 37: Matlab: Train Network

The training window shows the following information :

- The network structure with number of neurons in each layer and the transfer functions used.
- The training algorithms along with the data division parameters.
- The status of the learning phase of the network
- Options to plot the different training graphs during and after learning

There are two ways for the training to be completed. either can stop the training in between by clicking on the stop training button at any point of time or when the training gets automatically stop because any one of the network learning parameter like number of iterations, mean square error, minimum gradient ,number of validation stops ,etc. has been reached.

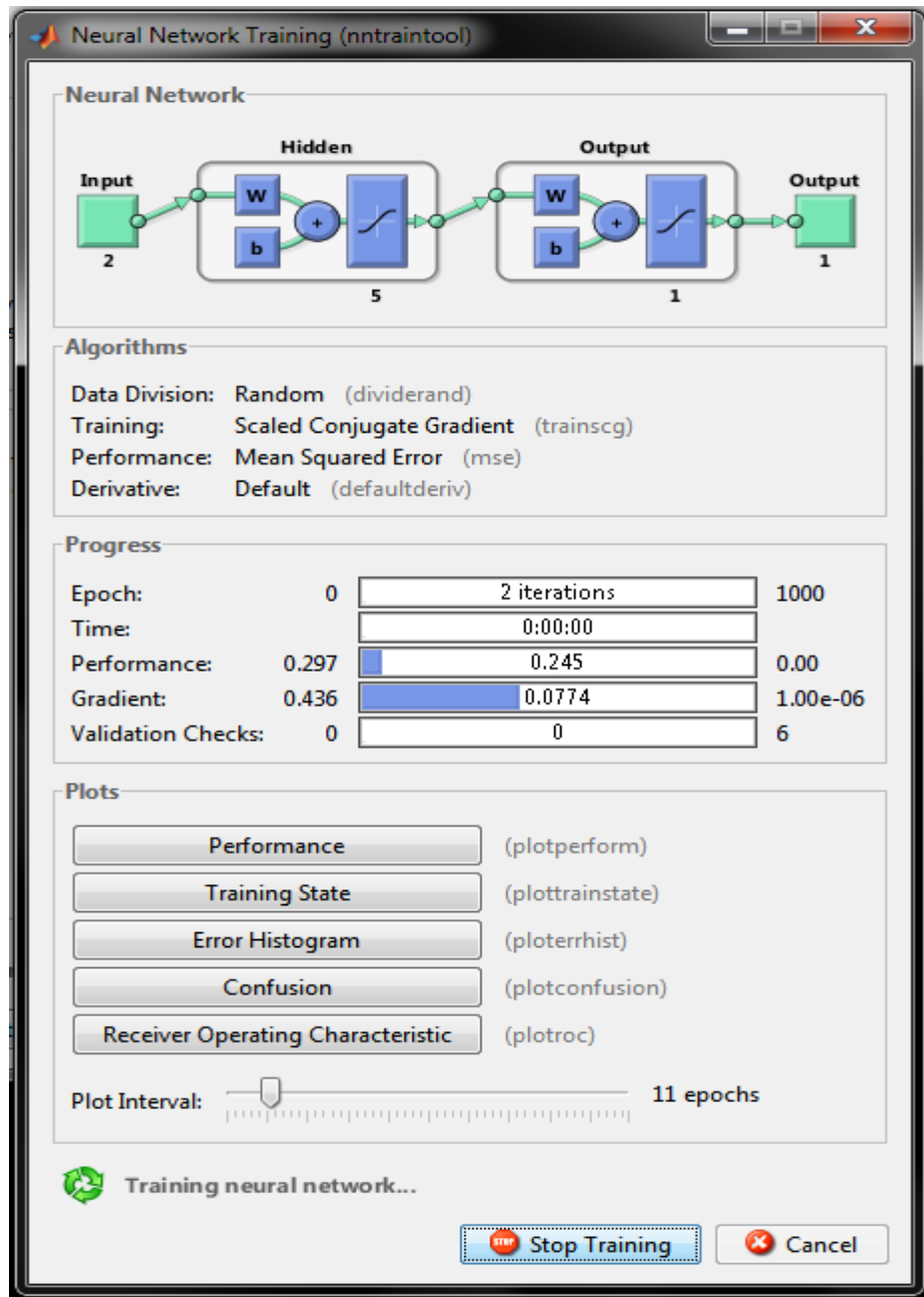


Figure 38: Matlab: Neural Network Training phase

As shown in figure below, the training has been completed as the number of validation stops which has been set to 6 during network initialization has been reached while training was in progress.

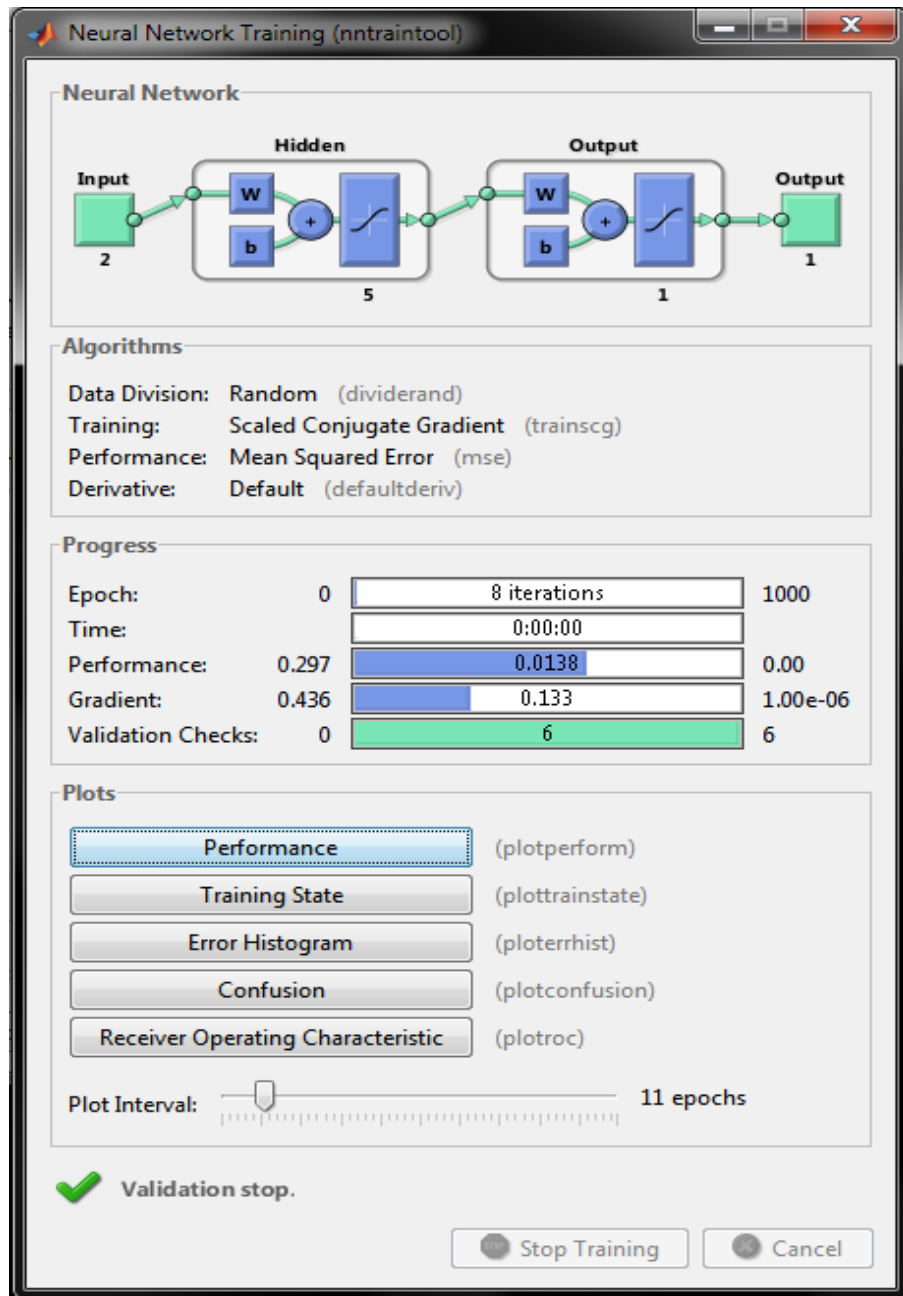


Figure 39: Matlab: NN Training completed

After training has been completed, the following window will show the option to retrain the network from start.

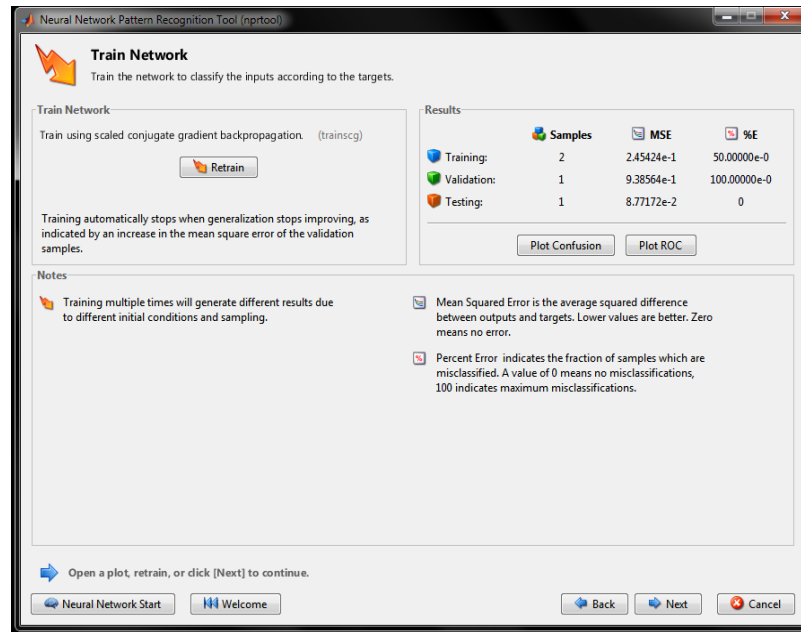


Figure 40 : Matlab: Retrain Network

If the network performance is not as per the desired results this option can be selecting with the options to increase the network size or with increased training dataset.

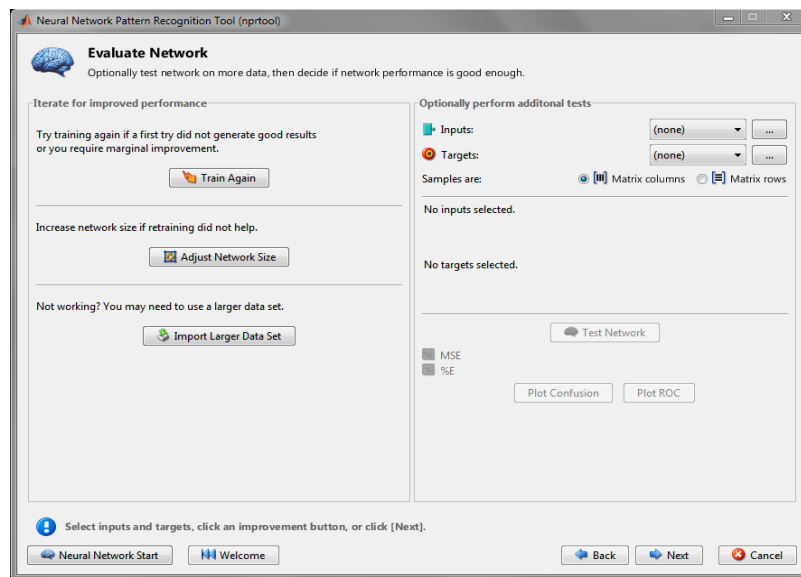


Figure 41 : Matlab: Evaluate Network

After the training has been successfully completed the next window will help save the results.

Either choose from the options given to save parts of the training results or the entire training results can be saved by checking in the check box "save all".

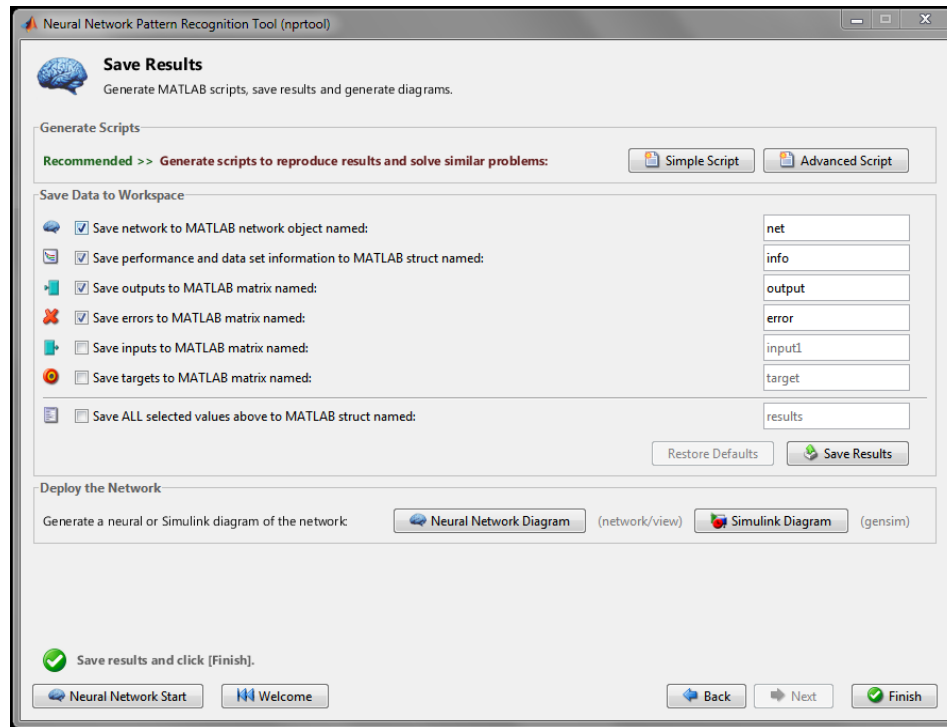


Figure 42: Matlab: Save results

The save option will save the results on the current workspace.

as shown in the figure below. the variable "net" has been saved to the current workspace.

```
>> whos
      Name                Size                Bytes   Class      Attributes

      ans                  1x11                22    char
      input                2x4                  64    double
      net                   1x1                 63022  network
      targetOutput         1x4                   32    double
```

f >> |

Figure 43: Matlab: Current Workspace

we can save the current trained network for future use in the file system using the save command as shown in the figure.

```

>> whos
  Name          Size          Bytes  Class      Attributes

  ans           1x11           22    char
  input         2x4            64    double
  net           1x1           63022  network
  targetOutput  1x4            32    double

>> save('xor_net')
fx >> |

```

Figure 44: Matlab: Saving workspace

The previously saved network can be reloaded into the current workspace using the load command as shown in the figure below.

```

>> load('xor_net')
>> whos
  Name          Size          Bytes  Class      Attributes

  ans           1x11           22    char
  input         2x4            64    double
  net           1x1           63022  network
  targetOutput  1x4            32    double

fx >> |

```

Figure 45: Matlab: Load Workspace

The trained network "net "is used to do predictions for the dataset "inputdata" using the following command net("inputdata"). the results can be seen in the figure below.

```
>> targetOutput
targetOutput =
    0     1     1     0
>> prediction = net(input)
prediction =
    0.0000    1.0000    1.0000    0.0000
fx >> |
```

Figure 46: Matlab: Predicting Output

As we can see the predicted output matches the desired output. The training has been completed successfully.

5 IBM WATSON

5.1 Data Processing

5.1.1 Dataset Design :

Datasets provided contains the following two files :

Training dataset :

It contains the data used to train the network. Following are characteristics of this dataset.

- 1) Total number of rows: 2,39,944
- 2) Each row contains one input vector.
- 3) A row represented with the following format:

Input vector ID, Input vector, Target output

<i>Id</i>	<i>Input Vector</i>	<i>Target Output</i>
1,844220.0,0.9472554922103882,0.0,	1.0,1.0	false
2,844220.0,0.0,0.9472554922103882,	0,1.0,1.0	false
3,844220.0,0.9472554922103882,0.0,	0.0,0.0	true
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮

Figure 47: Training Dataset

Where,

Input vector id: It is an integer which uniquely identifies the input vector and its target output

Input vector: A 319-dimensions vector to be used for the purpose of input training data.

Target output : The desired output in the form of TRUE or FALSE for the corresponding input vector.

Evaluation data set :

It contains the data used to evaluate the machine which was trained using the training dataset earlier. Following are characteristics of this dataset.

- 1) Total number of rows: 51,906
- 2) Each row contains one input vector.
- 3) A row represented with the following format:

Input vector ID, Input vector

<i>Id</i>	<i>Input Vector</i>
451868	,852517.0,0.0,0.9472554922103882,0.0,..... 1.0,1.0
451869	,852517.0,0.0,0.0,0.9472554922103882,.....0,1.0,1.0
451870	,852517.0,0.0,0.9472554922103882,0.0,.....0.0,0.0
⋮	⋮

Figure 48: Evaluation Dataset

where, *Input vector id*: It is an integer which uniquely identifies the input vector

Input vector: A 319-dimensions vector used for the purpose of generating predictions to evaluate the performance of the machine trained using training dataset.

5.1.2 Input data processing methods :

The dataset files provided were in the csv (comma separated values) format, therefore, the datasets can be loaded into the matrix using the *csvread* command as follows:

The following Matlab command reads the CSV files(containing only numeric values) into the matrix P:

```
P = CSVREAD("filename",row,column,csvrange);
```

where,

filename: The file to be read into matrix.

row : This argument is optional and when provided, it reads the csv file starting from the row number provided by skipping the previous ones.

col : This argument is optional and when provided, it reads the csv file starting from the column number provided by skipping the previous ones.

csvrange: This argument is optional and when provided, It reads only the range specified in this argument.

Prepare and process the training input data :

The training dataset file contains both the input vectors and the corresponding target outputs along with the vector IDs. Two training matrices will be constructed from this single file, the input data matrix and the target output matrix.

As the training dataset contains the target output in the form of True/False(and not in numeric form) therefore, the training dataset has to be processed before we can load it using *csvread* command. This can be easily done by following steps:

- 1) Open a file in a text editor and use *find and replace* option to convert all the "True" alphabetical words into numerical value "1" and "False" with "0" or vice-versa.

2) Using `csvread` command load the file into two matrices as follows:

```
p=csvread("training_dataset_filename",0,1,(0,1,239944,319))
```

```
t=csvread("training_dataset_filename",0,319,(0,319,239944,320))
```

Here, input data matrix *p* will be created by loading file and skipping the first column, which are Vector ID's, and last columns, which are target outputs.

The target output matrix *t* will be created by loading only the last column of the file by skipping everything else.

Even though the training dataset is in a plain text format, its size is huge as it contains millions of rows of data, thus it may become bottleneck for some common text editors to process it efficiently. Therefore, another way to process the training input dataset through programming is by creating two csv files, one for each, the input vectors and the target output in the form of 0/1. (See appendix no. for the java code). The resulted files can then be loaded directly using `csvread` command as follows.

```
p=csvread("training_dataset_filename")
```

```
t=csvread("training_dataset_filename_2")
```

Process the evaluation data :

The evaluation dataset contains the input vector ID and the input vector. As it contains only numeric data, unlike training input dataset which also had the alphabetical data in the form of target output, therefore, it can be directly loaded into evaluation input matrix, without having the need of pre processing, using the `csvread` command as follows

```
eval_p=csvread("evaluation_dataset_filename",0,1)
```

Here, evaluation input data matrix *eval_p* will be created by loading file and skipping only the first column, which are Vector ID's.

5.1.3 Output data predicted processing method

Generating Score for the trained Network

The predicted output data file on the evaluation dataset is required to be submitted at the competition website to get the score for the trained machine. The required format of the prediction file is as follows:

True Vector Id's

```
400011
400053
400374
⋮
```

Figure 49: True Vector Id's

- 1) It only contain the ID's for the input vectors generating "True" predictions.
- 2) It should be a text file(.txt).

Predicted output data pre-processing:

The trained neural network is used to predict the output on the evaluation dataset. The output file generated contains the predictions in the form of numbers (generally, decimal numbers ranging from 0 to 1) for the corresponding input vectors in the evaluation dataset. As the prediction file required to be submitted should contain only the true predictions vector Id, therefore there is a need for the preprocessing of the output data file. One way to process the output data file through programming (see appendix for the java code) is as follows:

- 1) As output contains numbers ranging from 0 to 1, set up a threshold limit for the numbers to be considered as *True* predictions.

- 2) Separate out the vector Ids generating number in the *True* predictions threshold limit .
- 3) Save these true predictions vectors Id's in a .t

5.2 Implementation

5.2.1 Gradient descent back-propagation

Algorithm :

Back-propagation is a gradient descent algorithm. It's an algorithm for taking one training case and computing efficiently for every weight in the network depending upon how the error will change on that particular training case as you change the weight. We can compute how fast the error changes as we change a hidden activity on a particular training case. So instead of using activities of the hidden units as our desired states, we use the error derivatives with respect to our activities

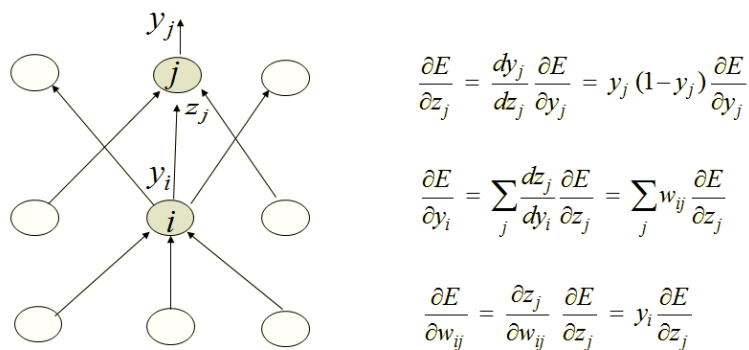


Figure 50: Back-propagation

The core of back propagation is taking error derivatives in one layer and from them computing the error derivatives in the layer that comes before that as shown in the figure 2.1

Configuration :

Learning rate = 0.01

Number of hidden layer neurons = 639

Training function used = traingd

Transfer function = tansig - tansig

Results:

Table 1: Gradient descent back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.014	23

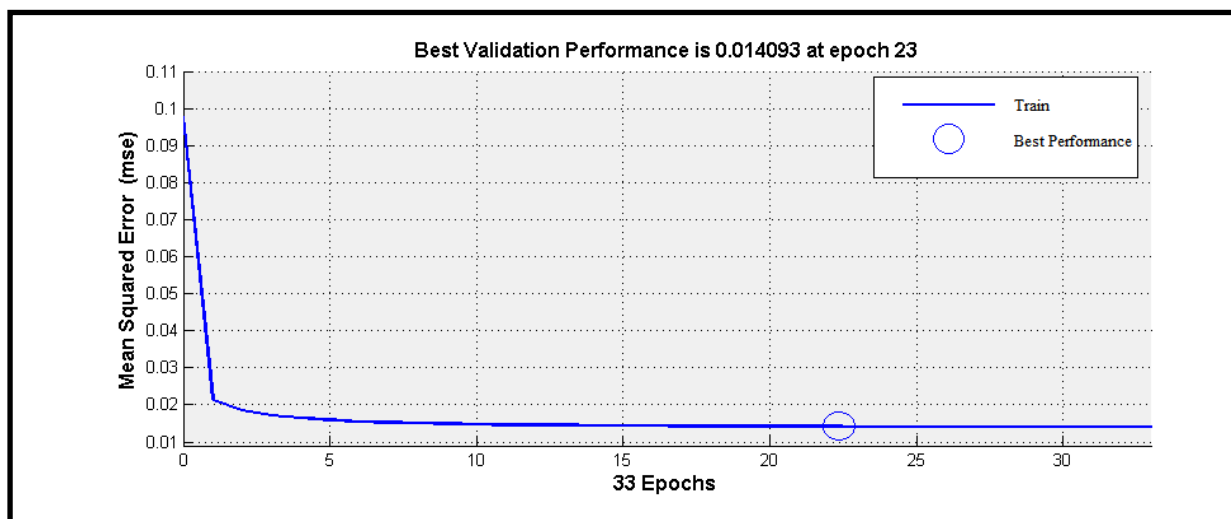


Figure 51: Gradient descent back-propagation Performance Graph

The graph shows the performance curve of the gradient descent algorithm measured using mean square error (mse). Starting with the initial random weights of the network the learning curve shows the good descent within first few iterations as we can see a steep drop in the mse from 0.09 to 0.02 within first two iterations over the entire training dataset. The training continued to perform better for few more iterations only and finally became negligible/unaccountable after 23 passes over the entire training dataset. The best mse able to reached by the network is 0.014 within 23 epochs.

5.2.2 Resilient back-propagation

Algorithm :

The motivation behind this approach is that the magnitude of the gradient can be very different for different weights & can change during learning. Therefore selecting a single global learning rate will not be easily possible. To come out with a solution we can modify our stochastic descent algorithm in such a way that it will now only consider the sign of the partial derivative over all patterns and thus it can be easily applied on different weights independently. The algorithm is as follows

- (1) If there is change in sign of the partial derivatives from previous iteration, then weight will be updated by a factor of η^- .
- (2) If there is no change in sign of the partial derivatives from previous iteration, then weight will be updated by a factor of η^+ .

Each weight is changed by its own update value, in the opposite direction of that weight's partial derivative. This method is one of the fastest and memory efficient weight update mechanisms as compared to standard steepest descent algorithm

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 639

Training function used = trainrp

Transfer function = tansig - tansig

Delta_increase = 1.2 .It determines the Increment to the weight change

Delta_increase = 0.5 .It determines the Decrement to the weight change

Delta_initial = 0.07 .it determines the initial weight change

Delta_Maximum = 50 . it determines the maximum amount by which the weights can be updated

Results:**Table 2: Resilient back-propagation results**

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.0145	20

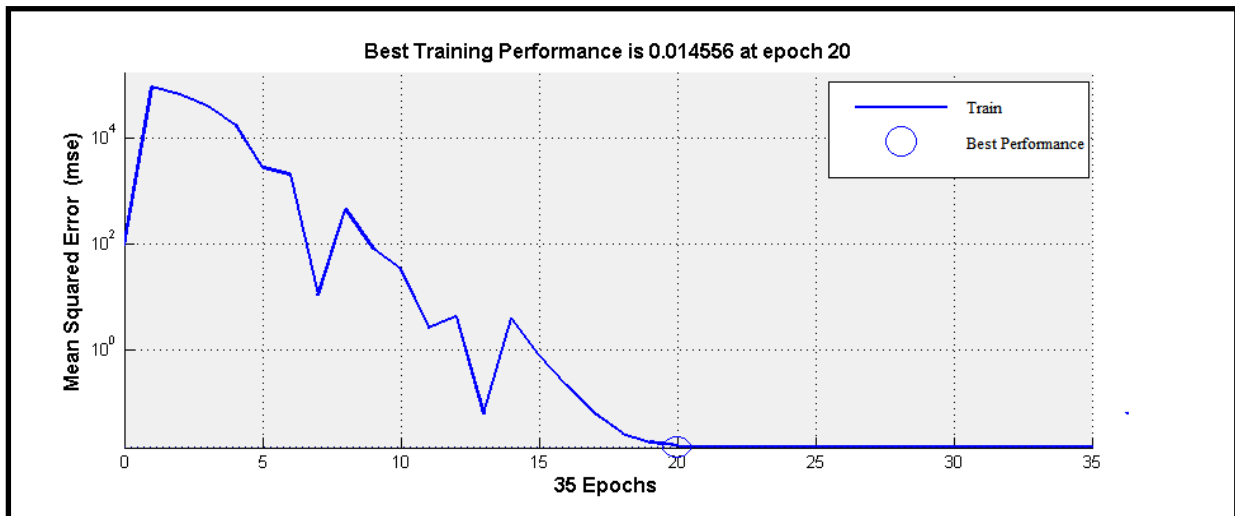


Figure 52: Resilient back-propagation Performance Graph

Figure shows the logarithmic performance curve of the resilient back-propagation algorithm measured using mse of the network with respect to the iterations over the training dataset. At start, with initial random weights, the training performing was getting worse, instead of reducing the error difference between the output predicted and the desired output, the training was updating weights resulting in increased mse for initial iterations producing steep slope upwards for the first two iterations. Compared to gradient descent, the results are completely opposite for the initial learning phases of the training. The graph shows few spikes generated due to the policy of updating weight by $\eta+$ or $\eta-$ only. The best mse able to reach by the network was 0.0145 in 20 epochs only. although the training was fast as compared gradient descent but had a decreased in mse from 0.014 to 0.0145.

5.2.3 Scaled conjugate gradient back-propagation

Algorithm:

The basic back-propagation algorithm adjusts the weights in the steepest descent direction where as in scaled conjugate gradient descent a search is performed based on conjugate directions. It not perform a line search at each iteration. This algorithm is proved to be faster than the basic gradient descent back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons =639

Training function used = trainscg

Transfer function = tansig - tansig

sigma = $5.0e-5$.It determine change in weight for second derivative approximation

lambda = $5.0e-7$. The parameter for regulating the indefiniteness of the Hessian

Results:

Table 3: Scaled conjugate gradient back-propagation results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.010	22

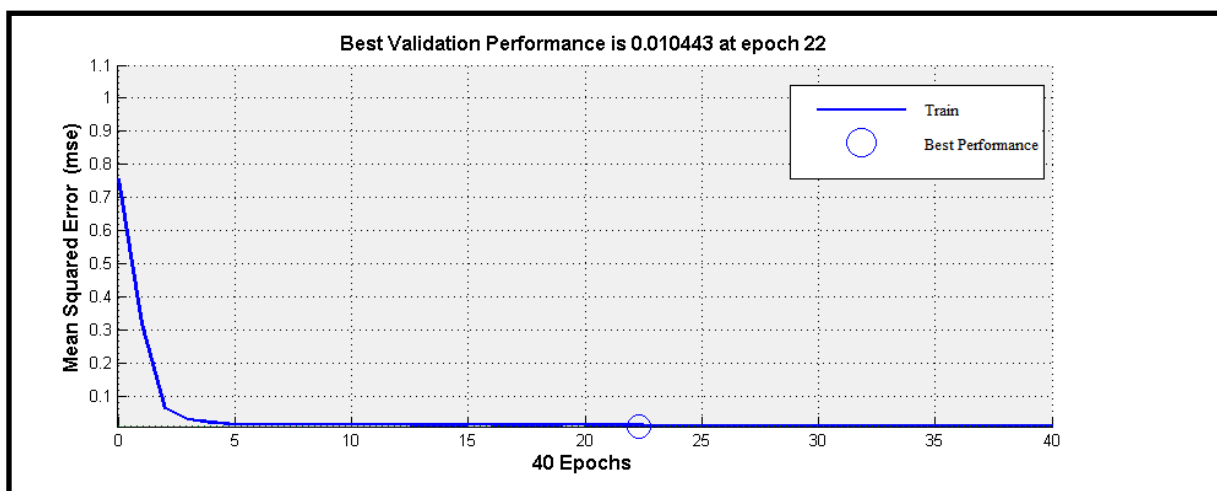


Figure 53 : Scaled conjugate gradient back-propagation Performance Graph

Figure shows the performance curve of the scaled conjugate algorithm using mse with respect to the number of iterations over the training dataset. During the initial learning phase of the training the performance was very similar to that of gradient descent algorithm generating a steep slope down within first two iterations of the training but outperforms it later on by reaching the best mse of 0.010 as compared to 0.014 from gradient descent within 22 passes over the training dataset.

5.2.4 Momentum back-propagation

Algorithm :

Unlike in mini-batch learning, in Momentum method we use the change in gradient to update the velocity and not the position of weight particle. In this method, we take the fraction of the previous weight updates and add that to the current one. The main idea behind doing this is to improve the efficiency of the neural network by preventing the system to converge to a local minimum or to converge to a saddle point.

Considering that we have taken very high momentum parameter which may help us speed up the convergence rate of the neural network, but this may also incur the risk of overshooting the momentum and making system unstable generating more function hikes. On the other hand, if we take momentum parameter very small, it may happen that neural network learns very slowly. The tradeoff between high and low value have to be kept in mind before choosing the momentum magnitude for training.

Configuration:

Learning rate = 0.01

Number of hidden layer neurons =639

Training function used = traingdm

Transfer function = tansig - tansig

Momentum = 0.9 .it determines the momentum constant.

Results :**Table 4: Momentum back-propagation Results**

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.014	2

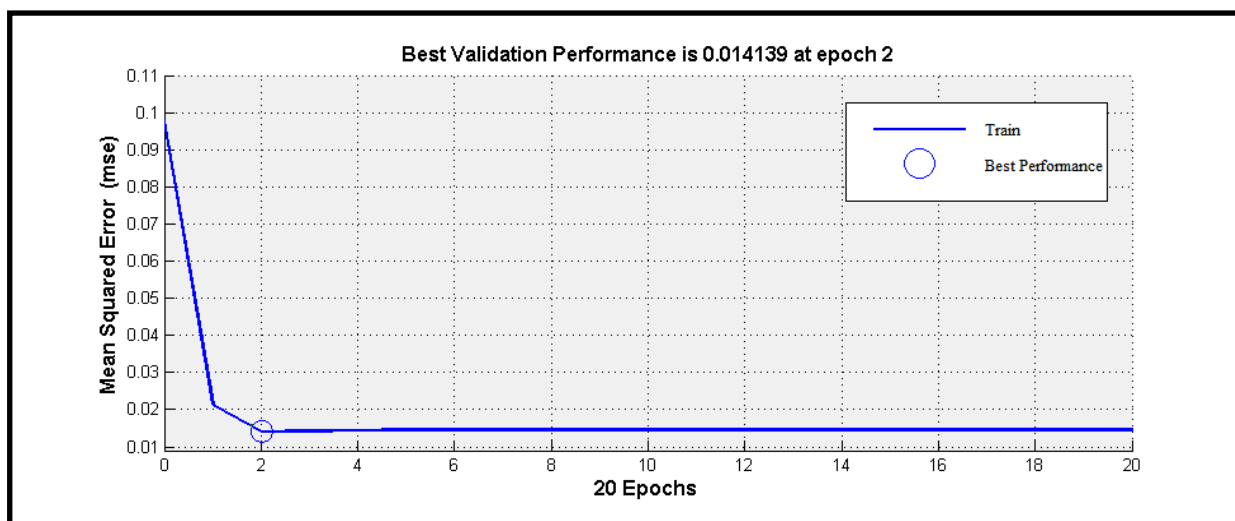
**Figure 54 : Momentum back-propagation Performance Graph**

Figure shows the performance curve of the momentum back-propagation using mse with respect to the number of iterations over the training dataset. The Steep drop in mse showing by the graph for the first two iterations is similar to that of gradient descent and SCG but it differs at a point that the best mse of 0.014, which is similar to that SCG and not too far from gradient descent, was achieved in first two epochs only. Although there was no any further/accountable decrease in network mse over the next few iterations continuously.

5.2.5 Adaptive Learning Rate back-propagation

Algorithm :

We have used multi layer neural network, hence, we need to consider that there is a wide variation on what will be the suitable learning rate at each corresponding layer i.e. the respective gradient magnitudes at each layer are generally different.

This situation motivates to use a global learning rate responsible for each weight update. With adaptive learning method, the trial and error search for the best initial values for the parameters can be avoided. Normally the adaption procedures are able to quickly adopt from the initial given values to the appropriate ones.

Also, the amount of weight that can be allowed to adapt depends on the shape of the error surface at each particular situation. The values of the learning rate should be sufficiently large to allow a fast learning process but also small enough to guarantee its effectiveness. If at some moment the search for the minimum is being carried out in the ravine, it is desirable to have a small learning rate, since otherwise the algorithm will oscillate between both sides of the ravine.

configuration:

Learning rate = 0.01

Number of hidden layer neurons = 639

Training function used = traingda

Transfer function = tansig - tansig

learning rate _increase = 1.05 .It determines the increase in learning rate.

learning rate _decrease = 0.7 .It determines the decrease in learning rate.

Results :**Table 5: Adaptive Learning Rate back-propagation Results**

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.015	28

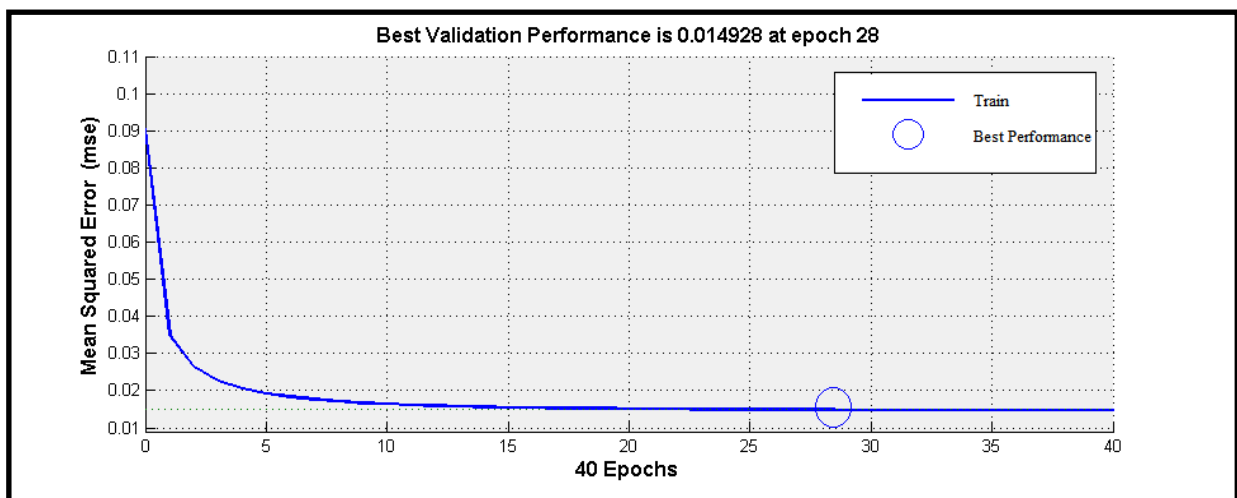
**Figure 55 : Adaptive Learning Rate back-propagation Performance Graph**

Figure shows the performance curve of the adaptive learning back-propagation using mse with respect to the number of iterations over the training dataset. Following the similar pattern of mse curve as earlier algorithms the reduction in mse showed a steep slope within the first few iterations over the dataset but it differs at a point that it was able to reach the best mse of approx. 0.015 in 28 iterations. which by far is the highest mse up till now requiring largest number of iterations.

5.2.6 Momentum and Adaptive Learning Rate back-propagation**Algorithm:**

It combines both the momentum back-propagation and adaptive learning rate strategy to train the network.

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 639

Training function used = traingda

Transfer function = tansig - tansig

Momentum = 0.9 .it determines the momentum constant.

learning rate _increase = 1.05 .It determines the increase in learning rate.

learning rate _decrease = 0.7 .It determines the decrease in learning rate.

Results :

Table 6 Momentum and Adaptive Learning Rate back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.015	3

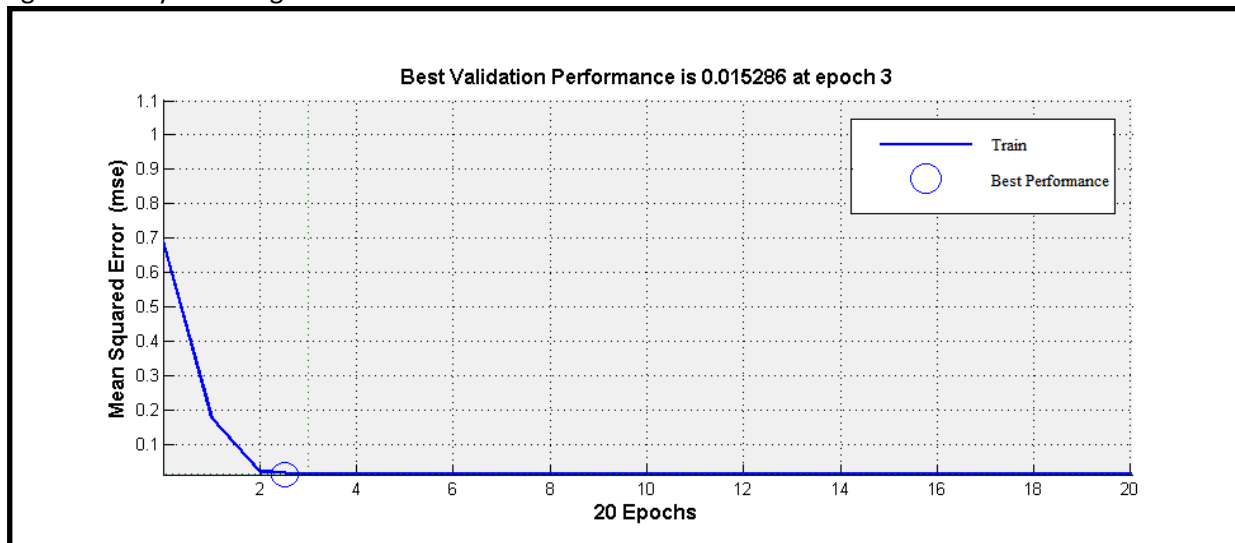


Figure 56 : Momentum and Adaptive Learning Rate Performance Graph

Figure shows the performance curve of the momentum back-propagation and adaptive learning rate strategy using mse with respect to the number of iterations over the training dataset. The graph is almost similar to that of the adaptive learning rate algorithm reaching the best mse of 0.015 but the current method was much faster than the adaptive learning alone as the number of iterations reduces from 28 to 3 only.

5.2.7 Method I: Non zero vector dimensions/features only

Implementation :

One of the methods to improve performance can be done by preprocessing the data such that the columns which remain zero throughout the entire training dataset can be safely removed from the raw dataset. The constant columns do not provide any meaningful information in classification of the input vectors. Removing such columns helps reduce the dimensionality in the training input vectors, thus helping in reducing the size of the neural network due to a reduction in the number of neurons required in the hidden layer of the network.

In the IBM data set, after removing the constant columns of zero values from the entire data set, the dimensionality of the input vectors reduced from 319 to 261 only, which eventually helped reduce the network size from having 639 neurons in the hidden layer to 523 neurons only.

Results :

Table 7: Non-zero vector dimensions/features Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.068	39

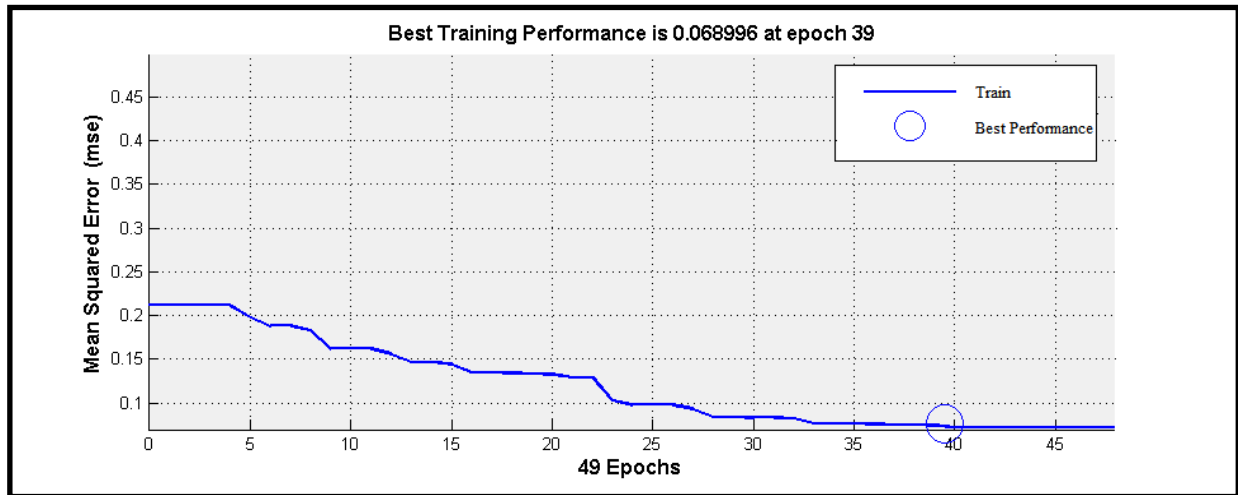


Figure 57 : zero vector dimensions/features Performance Graph

The results shows a gradual decrease in the mse of the network. which at start look very promising due to the reduced network size but the training efficiency instead become worse requiring approx 40 iterations to reached an mse of 0.007 which is highest among all the previous algorithms discussed with full dataset.

5.2.8 Method II: Four times more true vectors.

Implementation :

Another approach to preprocess the data to help increase the efficiency and accuracy of

the network is increasing the ratio of minorities in the raw training data by duplicating the vectors.

for example, in IBM data set only 1.47 % of the input row vectors contains the target output of True .therefore , tried with boosting the true input row vectors 4 times by duplicating the data.

Results :

Table 8: four times more true vectors Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.026	47

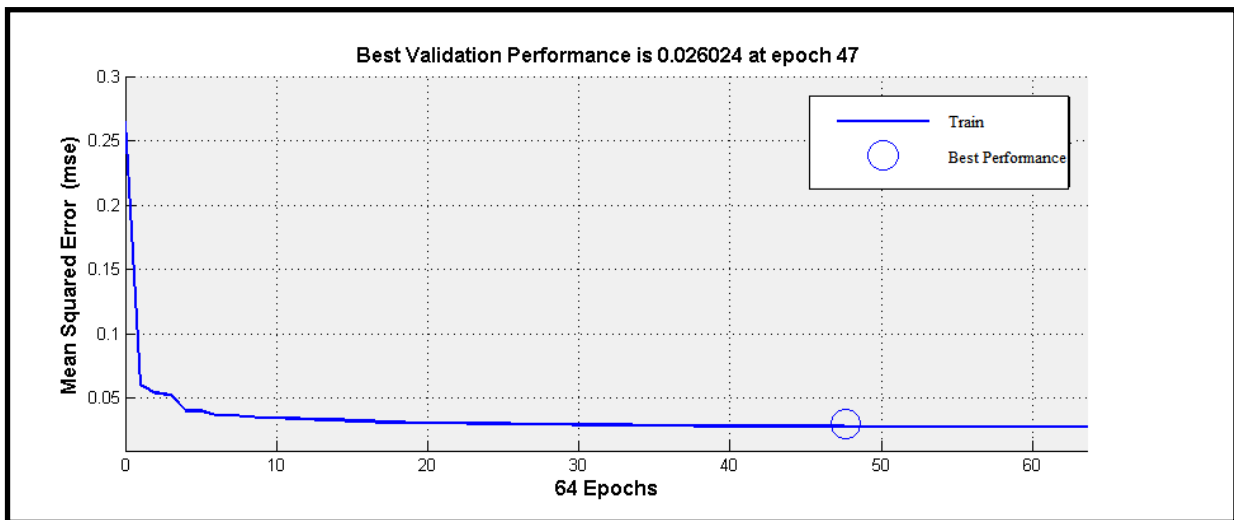


Figure 58 : four times more true vectors Performance Graph

The Performance was much better compared to method 1 approach but the results were not as close to those of training algorithms discussed earlier. The best mse able to reached was 0.026 in 47 epochs before the update in mse started became negligible/unaccountable for learning further learning.

5.2.9 Method III: Removed all the Constant columns

Implementation

Removed all the columns which were constant in the entire training dataset.

constant columns(features) removed : 72

features remaining : 247

Results :

Table 9: Removed all the Constant columns

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.0243	24

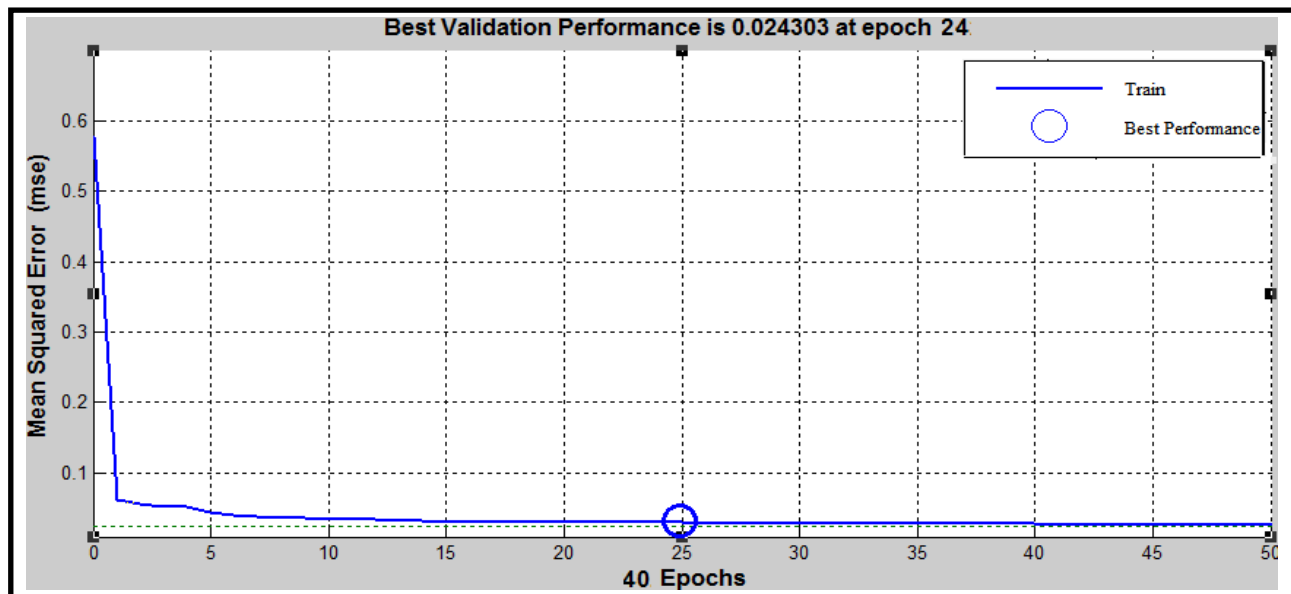


Figure 59 : Removed all the Constant columns

5.2.10 Method IV : Removing all the true vector constant columns

Implementation

First found all the rows which were true, then search for all the columns which are constant in these true rows set only, which may or may not be constant in the entire training dataset. Now removed all these constant columns from the entire training dataset.

constant columns(features) removed : 78

features remaining : 241

Results :

Table 10: Removing all the true vector constant columns

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.0246	34

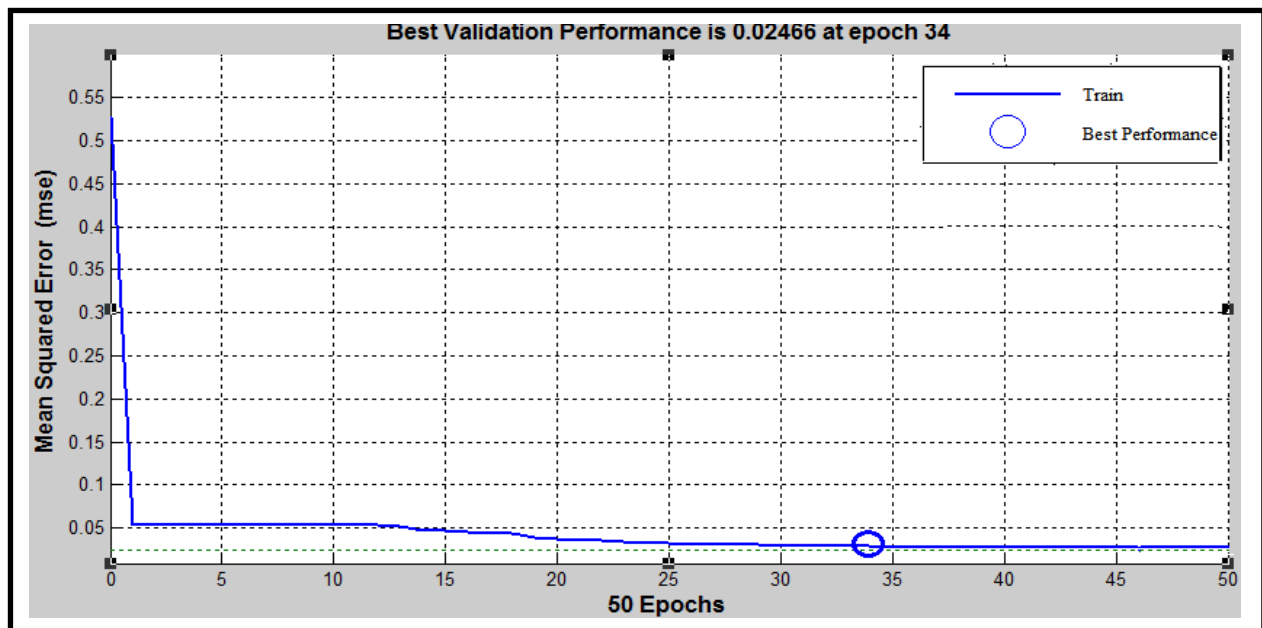


Figure 60: Removing all the true vector constant columns

In this case, The training was fastest as the number of constant columns further increased by 5 more columns as compared to the case 2 above, But the network didn't able to improve the score. As we were not sure whether to provide those extra 5 columns more priority over other columns or just discard them completely. and we tried by proceeded with the option of discarding them.

Comparison and Conclusion

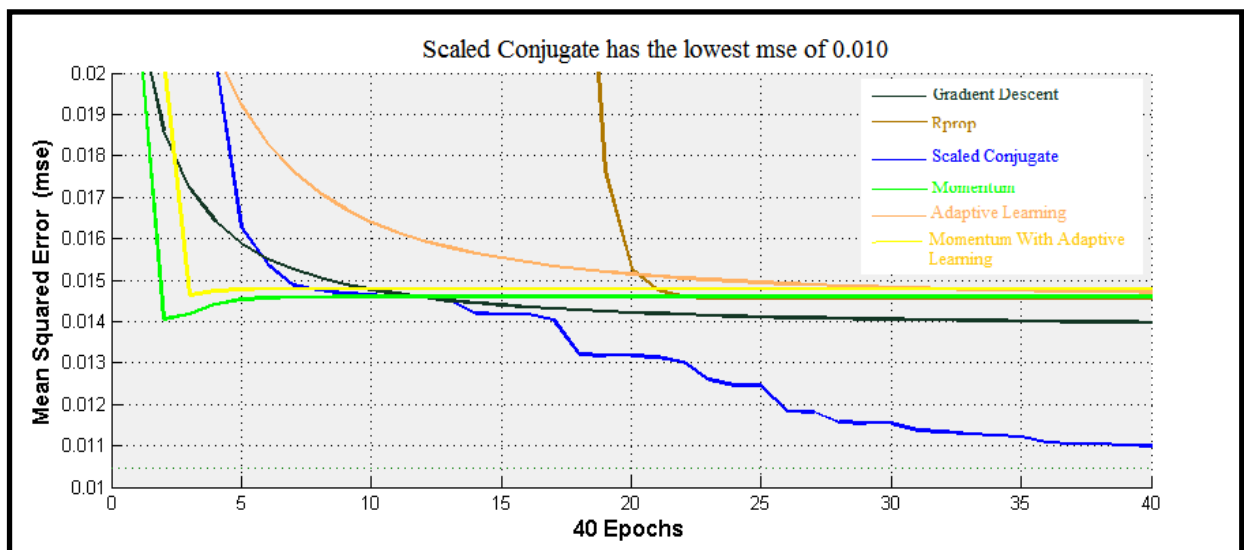


Figure 61 : Comparison and Conclusion Graph

The figure shows the comparison of different learning algorithms used with respect to the number of iterations. Total number of iterations done are 40. During the initial phase of the training the learning curve is almost similar for most of the algorithms except resilient back propagation which generate the slope completely opposite to the others. To some extent the combined approach of momentum and adaptive learning also takes time to converge during the initial learning stages . The momentum algorithm converges fastest among all within two iterations but had a learning rate of 0.014. As it can be easily seen in the graph, the scaled conjugate gradient has the best performance in terms of accuracy in prediction by having the least mse of 0.01 only.

6 INNOCENTIVE

6.1 Data Processing

6.1.1 Dataset Design :

Following were the details of the InnoCentive project challenge datasets available for training and prediction . The dataset is generated on the background of the game of trust which is played between two people , not necessary to be a friend or known to each other, judging whether the person should be trusted or not based on the promises which he has made and actions he has took in fulfilling of those promises all as a part of a game.

Going through the challenge details found out there are two types of dataset available:

1 RAW dataset : it sizes ranges from 26 gb to 60 gb. the size is huge because it contains lots of readings from ECG and other electronic instruments to capture the person biological and physical readings during play

2 PROCESSED dataset : size is of few kb's only.

The processed dataset is derived from the raw dataset and contains the data in the format suitable for statistical, mathematical, cognitive or AI analysis. It contains the data in the form features. there are approx 100 features and few hundred rows in the training dataset.

The processed dataset completely resembles the raw dataset and the competition will only consider the final score irrespective of which dataset set has been used for generating the results. Therefore downloaded all the Processed datasets for analyzing ,training and final evaluation purpose.

Analysis of the processed datasets provided on the challenge dashboard.

There are five types of data sets available:

- 1) INSTINCT_Cortisol
- 2) INSTINCT_Mayer
- 3) INSTINCT_NEO
- 4) INSTINCT_Oxytocin
- 5) INSTINCT_Feature_Matrices

Although the first four dataset has been provided for our evaluation and training purpose as these datasets were also derived as a byproduct from the raw datasets preprocessing but it is not known if that information will be helpful in predicting the results or not. The information containing in this matrices dataset is not included in the feature matrices dataset, which is going to be used for machine learning.

Training dataset :

It contains the data used to train the network. Following are characteristics of this dataset.

- 1) Total number of rows: 431
- 2) Each row contains one input vector with 109 features .
- 3) A row represented with the following format:

Input vector ID, constant , Target output, Input vector

	<i>Id</i>	<i>Constant</i>	<i>Target Output</i>	<i>Input Vector</i>
1	73,1,2,7	b	Exact amount promised	,high,7,5.25,0.83333,0.57143,0.57143,776.0714111,243.4980927,244.0
2	73,1,2,8	b	Exact amount promised	,high,7,5.5,0.91667,0.5,0.64286,756.8734131,470.1168823,205.85751
3	73,1,2,9	b	Exact amount promised	,medium,9,5.5,0.91667,0.5,0.78571.750.8125.185.1594391.229.434310
	⋮			⋮

Figure 62: Training Dataset

Where,

Input vector id: It is an integer which uniquely identifies the training input vector and its target output

Input vector: A 109-dimensions vector to be used for the purpose of training input data.

Target output : The desired output in the form of whether a person should be trust or don't' trust for the corresponding input vector.

Constant : is a part of training input vector.

Evaluation data set :

It contains the data used to evaluate the machine which was trained using the training dataset earlier. Following are characteristics of this dataset.

- 1) Total number of rows: 215
- 2) Each row contains one input vector of 109 features.
- 3) A row represented with the following format:

Input vector ID, constant, Input vector

	<i>Id</i>	<i>Constant</i>	<i>Input Vector</i>
1	22,2,1,2	b	, medium, 5, 5.5, 0.75, 0.92857, 0.85714, 625.6911011, 509.5175476, 140.56
2	22,2,1,3	b	, high, 7, 5.75, 0.58333, 0.78571, 0.5, 624.6165771, 757.503418, 174.37849
3	22,2,1,4	b	, high, 10, 4.5, 0.75, 0.92857, 0.85714, 620.0880737, 1501.082764, 90.1083
	⋮	⋮	⋮

Figure 63: Evaluation Dataset

where,

Input vector id: It is an integer which uniquely identifies the evaluation input vector

Input vector: A 109-dimensions vector used for the purpose of generating predictions to evaluate the performance of the machine trained using training dataset.

Constant : is a part of training evaluation input vector.

6.1.2 Input data processing methods :

The dataset files provided were in the csv (comma separated values) format, therefore, the datasets can be loaded into the matrix using the `csvread` command.

Prepare and process the training input data :

The training dataset file contains both the input vectors and the corresponding target outputs along with the vector IDs. Two training matrices will be constructed from this single file, the input data matrix and the target output matrix.

Apart from three feature columns,

i.e. *column 5, column 6 and column 7*

which are string representations, all of the remaining features are in the format of numbers.

As the training dataset contains the target output in the form of string representations (and not in numeric form) therefore, the training dataset has to be processed before we can load it using `csvread` command.

<i>Id</i>	<i>Target Output</i>	<i>Input Vector</i>
1	73,1,2,7	1
2	73,1,2,8	0
3	73,1,2,9	0
⋮	⋮	⋮
⋮	⋮	⋮

Figure 64: Input data processing methods

1] column 6 data will constitute the desired output of the training dataset. Following will be the column 6 representations into number data type :

Column 6 : It has the following ranges:

- Exact amount promised,* -----TRUST-----> 1
- More than promised* -----TRUST-----> 1
- Promise not fulfill able* -----DO NOT TRUST-----> 0
- Less than promised* -----DO NOT TRUST-----> 0

As per the instructions mentioned in the challenge details: The player action is considered to be entitled for TRUST classification if and only if either of the following two conditions satisfies:

condition 1: The player(which can be a friend or stranger in the game) has provided the exact amount promised

condition 2: the player has provided the amount which is more then what has been initially promised by him.

Therefore, in dataset, all the strings representing above two conditions are replaced with (Trust) 1 and the remaining string representations with (Don't Trust) 0.

2] column 7 data string data type representations into number data type :

Column 7 : the column 7 has the following ranges:

<i>High</i>	----->	100
<i>Medium</i>	----->	50
<i>Low</i>	----->	0

3] The Unique row vectors Id's are constituted with the subset containing first four columns of the dataset.:

unique row vector ID's : [PID, SESSION, BLOCK, ROUND]

4] Column 5 is constant throughout the entire training dataset ,therefore it's contribution to the training is none. therefore, it will be removed from the training datasets.

Using `csvread` command load the file into two matrices as follows:

```
p=csvread("training_dataset_filename")
```

```
t=csvread("desired_output_dataset_filename")
```

where, `p`= input to the network.

`t` = desired output of the network.

Process the evaluation data :

Evaluation dataset also contains the input vectors in the form of string representations (and not in numeric form) therefore, the evaluation dataset has to be processed before we can load it using `csvread` command.

	<i>Id</i>	<i>Input Vector</i>
1	22,2,1.2,,	50, .5, 5.5, 0.75, 0.92857, 0.85714, 625.6911011, 509.5175476, 140.56
2	22,2,1.3,,	100, 7, 5.75, 0.58333, 0.78571, 0.5, 624.6165771, 757.503418, 174.37849
3	22,2,1.4,,	100, 10, 4.5, 0.75, 0.92857, 0.85714, 620.0880737, 1501.082764, 90.1083
⋮	⋮	⋮

Figure 65: Process the evaluation data

Here , we only have one column , i.e *column 7* which is in string representations , all of the remaining features are in the format of numbers. therefore, like in training dataset we will convert the column 7 data representations into number data type :

Column 7 : the column 7 has the following ranges:

High -----> 100

Medium -----> 50

Low -----> 0

Like in training dataset, evaluation dataset's Unique row vectors Id's are also constituted with the subset containing first four columns of the dataset.:

unique row vector ID's : [pID, SESSION, BLOCK, ROUND]

Similarly, like in training dataset, column 5 is constant throughout the entire evaluation dataset, therefore its contribution to the prediction is none. therefore, it will be removed from the evaluation datasets.

The evaluation dataset is loaded using the *csvread* command as follows

```
eval_p=csvread("evaluation_dataset_filename")
```

6.1.3 Output data predicted processing method

Generating Score for the trained Network

The predicted output data file on the evaluation dataset is required to be submitted at the competition website to get the score for the trained machine. The required format of the prediction file is as follows:

Vector Id's Predictions

1	test, , , ,
2	99, 2, 1, 4, , Trust
3	150, 2, 2, 7, Trust
4	162, 2, 2, 8, Don't trust
5	124, 2, 2, 7, Trust
6	162, 2, 2, 9, Don't trust
7	25, 1, 2, 10, Trust
8	93, 1, 1, 4, Trust
⋮	⋮
⋮	⋮
⋮	⋮

Figure 66: Predicted Vector Id's

- 1) First line should contain "test,,,"
- 2) It only contains the ID's for the evaluation vectors.
- 3) Id's should be followed by predictions TURST or DON'T TRUST
- 4) It should be a text file(.txt). which should be zipped in .zip /.rar format.

Predicted output data pre-processing:

The trained neural network is used to predict the output on the evaluation dataset. The output file generated contains the predictions in the form of numbers (generally, decimal numbers ranging from 0 to 1) for the corresponding input vectors in the evaluation dataset. As the prediction file required to be submitted should contain only the TRUST and DON'T TRUST values along with their predictions vector Id, therefore there is a need for the preprocessing of the output data file.

After training was completed and predictions were calculated on the evaluation data-set. combined the predicted classes as

≤ 0.5	----->	DONT' TRUST
> 0.5	----->	TRUST

6.2 Implementation

6.2.1 Gradient descent back-propagation

Configuration :

Learning rate = 0.01

Number of hidden layer neurons = 219

Training function used = traingd

Transfer function = tansig - tansig

Results:

Table 11: Gradient descent back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.198	517

The graph shows the performance curve of the gradient descent algorithm measured using mean square error (mse). Starting with the initial random weights of the network the learning curve shows the good descent within first few iterations as we can see a steep drop in the mse within first few iterations over the entire training dataset. The training continued to perform better for few more iterations only and finally became negligible/unaccountable after 517 passes over the entire training dataset. The best validation performance mse able to reached by the network is 0.198 within 517 epochs.

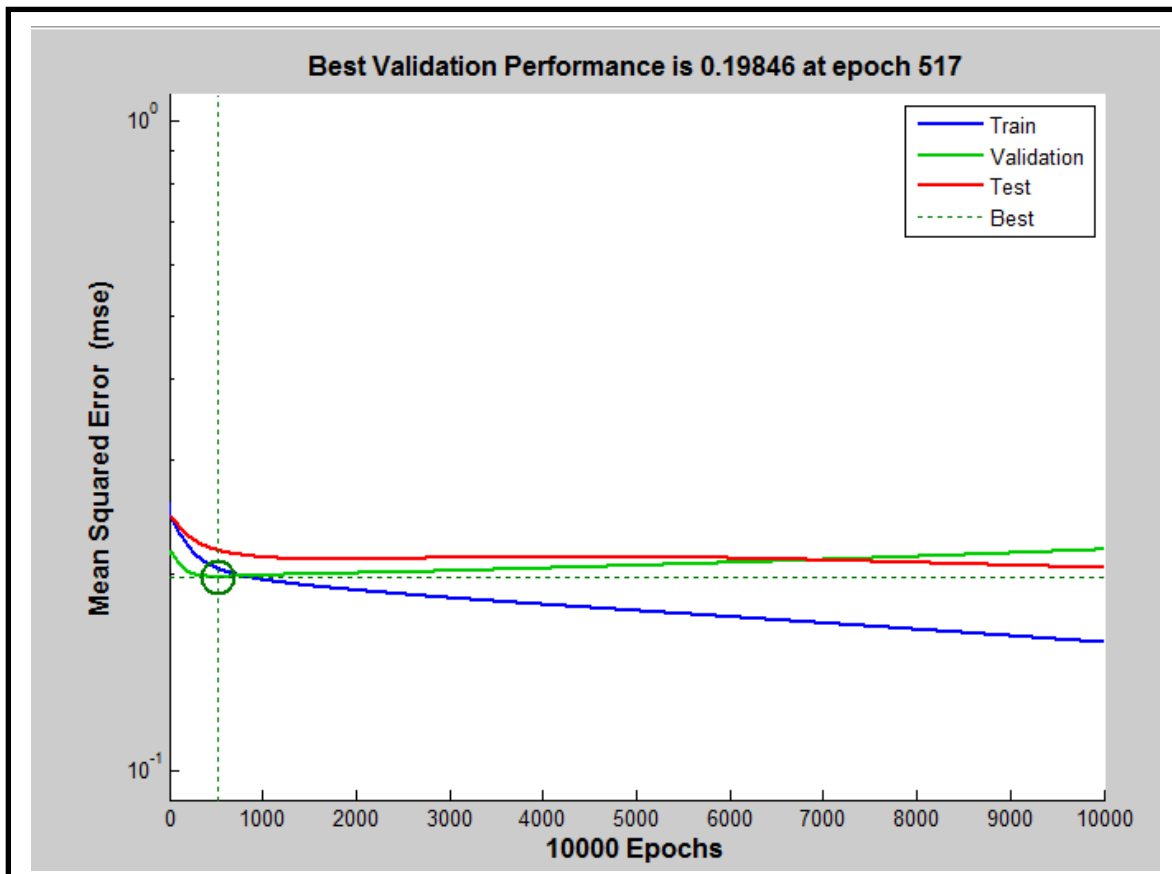


Figure 67: Gradient descent back-propagation Performance Graph

6.2.2 Resilient back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 219

Training function used = trainrp

Transfer function = tansig - tansig

Delta_increase = 1.2 .It determines the Increment to the weight change

Delta_decrease = 0.5 .It determines the Decrement to the weight change

Delta_initial = 0.07 .it determines the initial weight change

Delta_Maximum = 50 . it determines the maximum amount by which the weights can be updated

Results:

Table 12: Resilient back-propagation results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.20	66

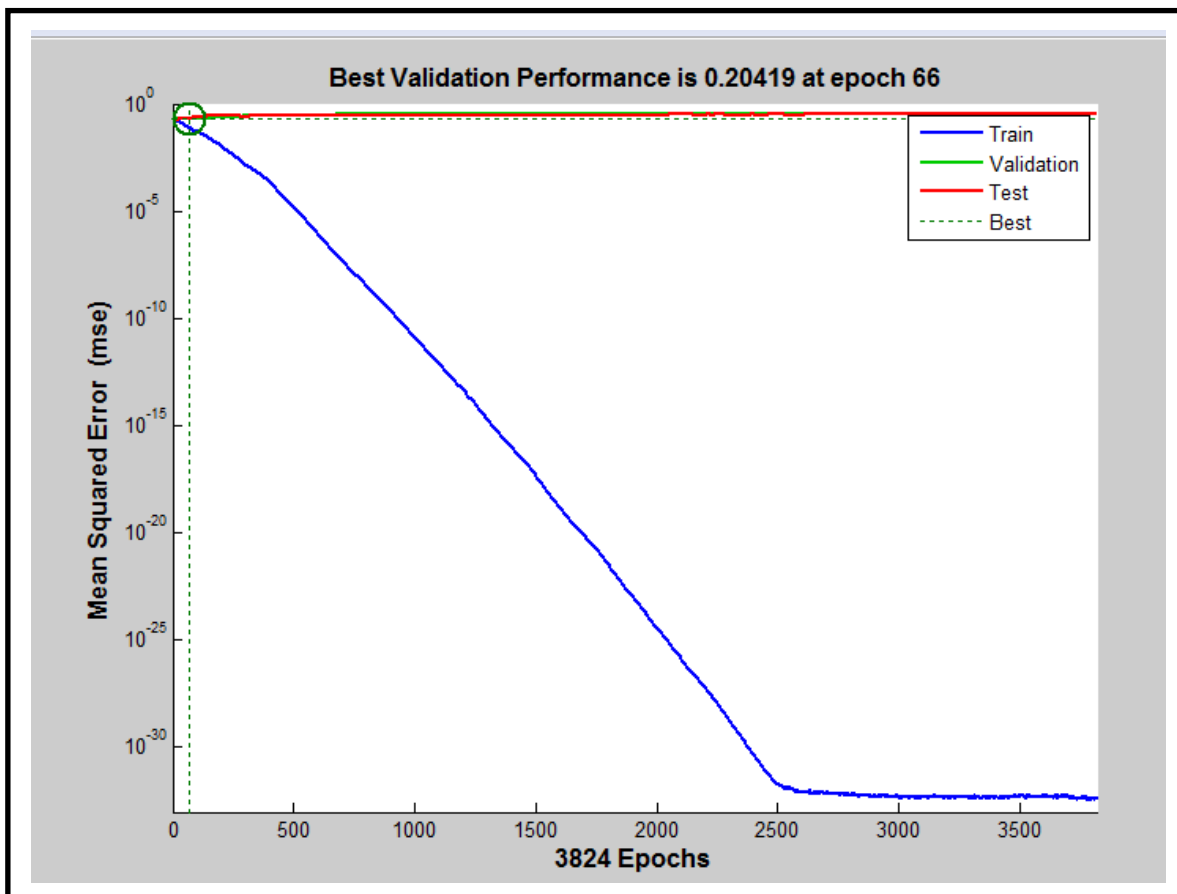


Figure 68: Resilient back-propagation Performance Graph

Figure shows the logarithmic performance curve of the resilient back-propagation algorithm measured using mse of the network with respect to the iterations over the

training dataset. At start, with initial random weights, the training performance curve steadily keeps on declining up-till 2500 iterations. and later on the training become almost negligible. The validations performance is best at 0.204 which comes at much faster rate as compared to gradient descent algorithm.

6.2.3 Scaled conjugate gradient back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons =219

Training function used = trainscg

Transfer function = tansig - tansig

sigma = $5.0e-5$.It determine change in weight for second derivative approximation

lambda = $5.0e-7$. The parameter for regulating the indefiniteness of the Hessian

Results:

Table 13: Scaled conjugate gradient back-propagation results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.192	43

Figure shows the performance curve of the scaled conjugate algorithm using mse with respect to the number of iterations over the training dataset. During the initial learning phase of the training the performance is very different as compared to the gradient descent algorithm. It shows a gradual slope for first few hundred iterations followed by a steep slope down from 1000th to 1300th iteration and outperforms the gradient descent and resilient back propagation algorithms. by reaching the best validation performance mse of 0.192 within much faster time of 43 iterations only.

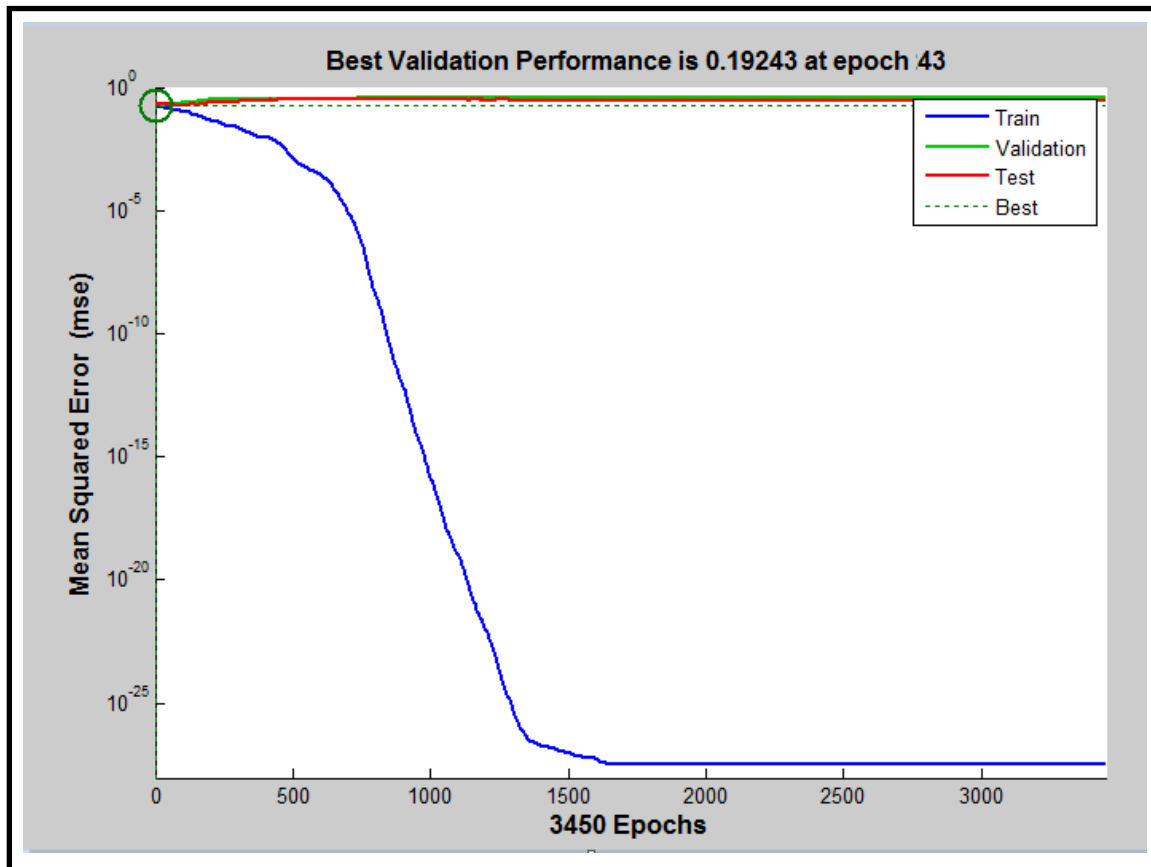


Figure 69 : Scaled conjugate gradient back-propagation Performance Graph

6.2.4 Momentum back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 219

Training function used = traingdm

Transfer function = tansig - tansig

Momentum = 0.9 .it determines the momentum constant.

Results :

Table 14: Momentum back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.201	868

Figure shows the performance curve of the momentum back-propagation using mse with respect to the number of iterations over the training dataset. The drop in mse showing by the graph for the first few iterations is similar to that of gradient descent and Scaled conjugate gradient but later on the slop becomes almost flat. similar is the slope of validation performance and test performance of the network . Although there was no any further/accountable decrease in network mse over the next few iterations continuously, the performance of the network is almost similar to the resilient back propagation algorithm with best validation mse achieved is 0.20

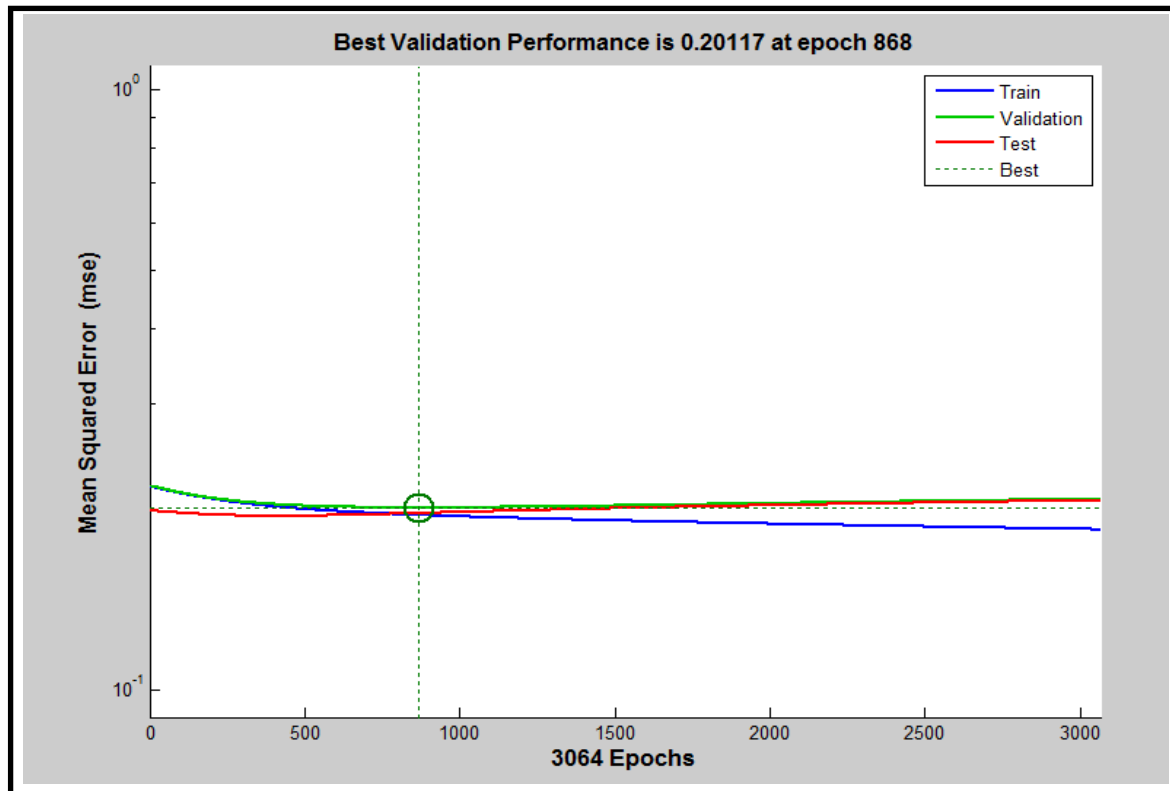


Figure 70 : Momentum back-propagation Performance Graph

6.2.5 Adaptive Learning Rate back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 219

Training function used = traingda

Transfer function = tansig - tansig

learning rate _increase = 1.05 .It determines the increase in learning rate.

learning rate _decrease = 0.7 .It determines the decrease in learning rate.

Results :

Table 15: Adaptive Learning Rate back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.22	70

Figure shows the performance curve of the adaptive learning back-propagation using mse with respect to the number of iterations over the training dataset. The curve contains ripples due to the nature of adaptive learning rate where the learning rate of the network is either increased or decreased depending on the learning threshold limit set for the network. this continuous change in learning rate resulted in back and forth movement of the curve. similar is the case with test and validations performance curves. The best validation performance the network able to reach was 0.223 in 70 iterations which is the highest value up till now.

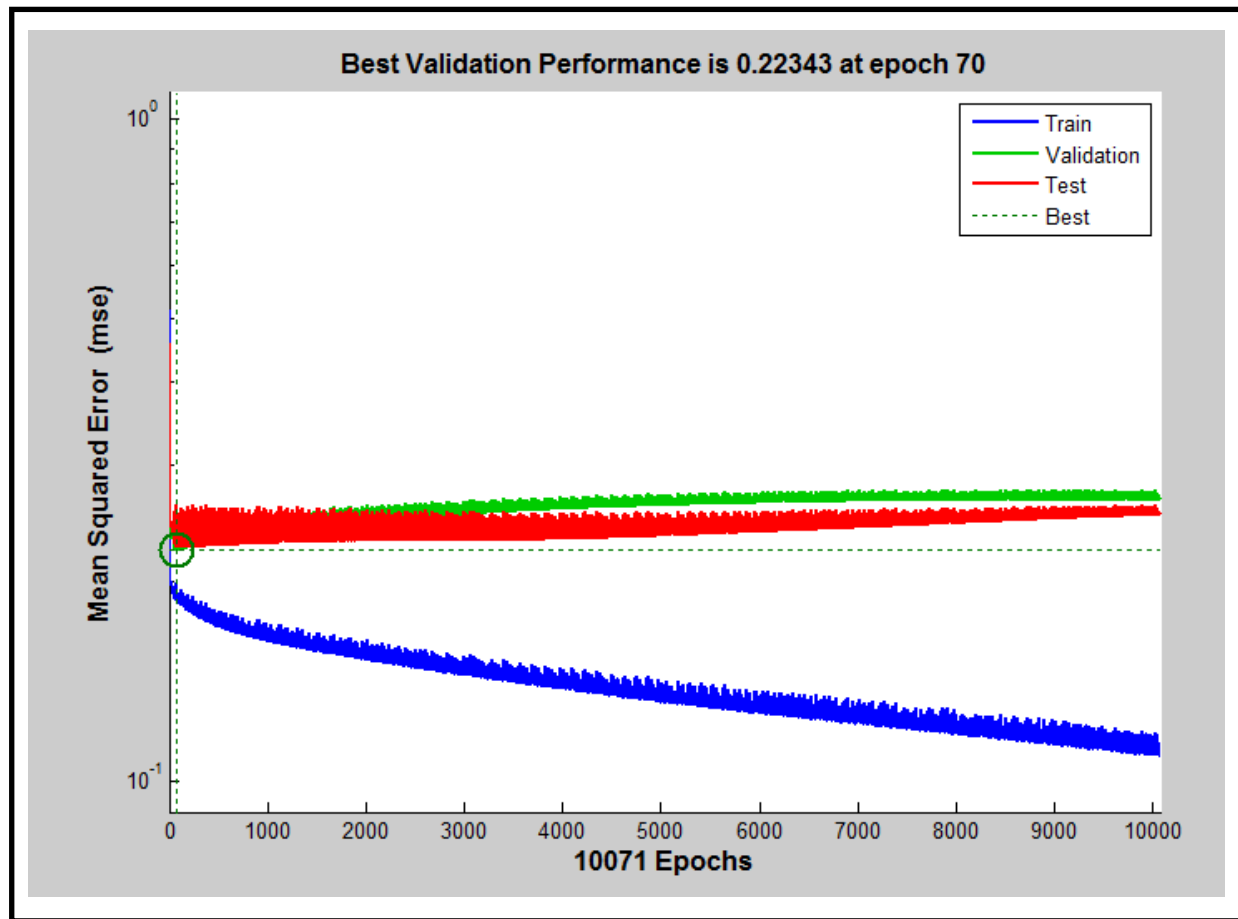


Figure 71 : Adaptive Learning Rate back-propagation Performance Graph

6.2.6 Momentum and Adaptive Learning Rate back-propagation

Configuration:

Learning rate = 0.01

Number of hidden layer neurons = 639

Training function used = traingdx

Transfer function = tansig - tansig

Momentum = 0.9 .it determines the momentum constant.

learning rate _increase = 1.05 .It determines the increase in learning rate.

learning rate `_decrease = 0.7` .It determines the decrease in learning rate.

Results :

Table 16 Momentum and Adaptive Learning Rate back-propagation Results

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.22	98

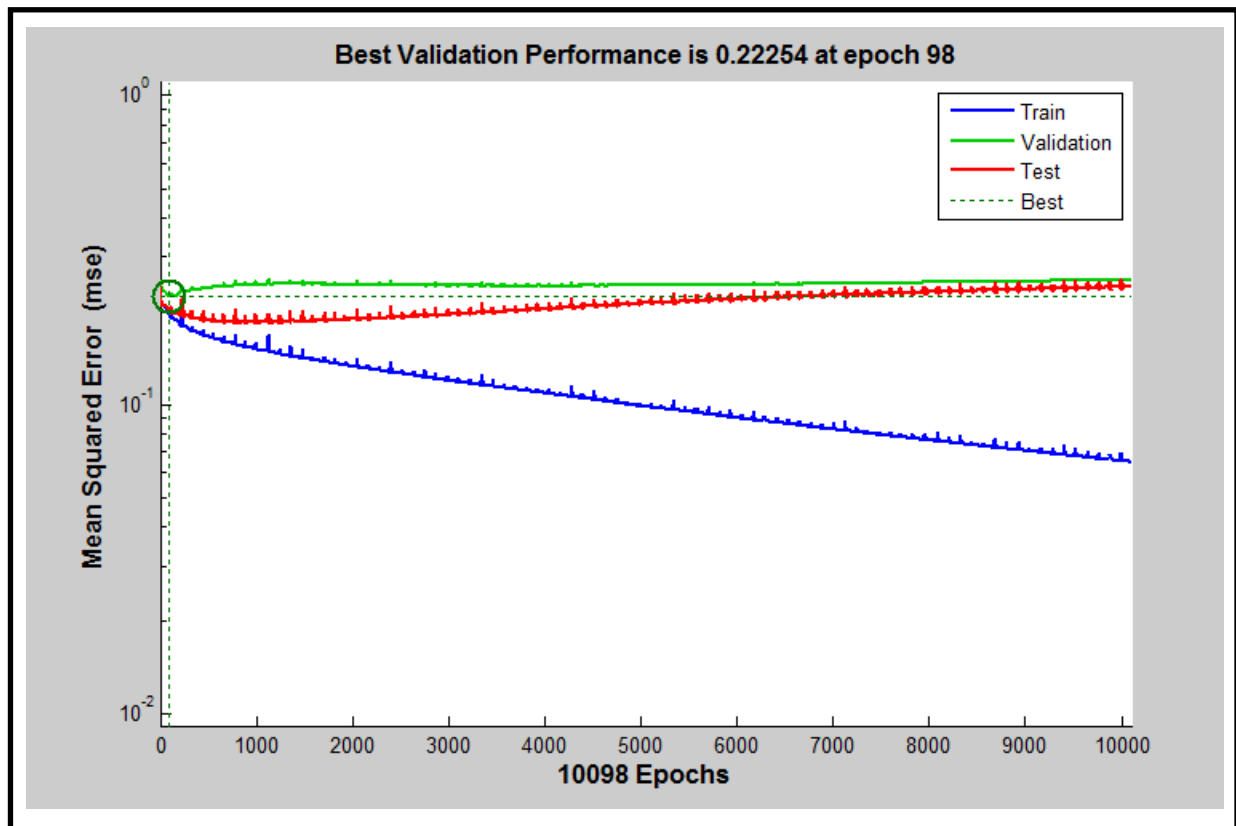


Figure 72 : Momentum and Adaptive Learning Rate Performance Graph

Figure shows the performance curve of the momentum back-propagation and adaptive learning rate strategy using mse with respect to the number of iterations over the training dataset. The graph is almost similar to that of the adaptive learning rate

algorithm reaching the best validation performance mse of 0.22 but with a higher number of iterations comparatively

6.2.7 Method I: Four classifications in desired output vector

Implementation :

One of the method to improve the performance is by preprocessing the data such that , it completely resembles the strings without any loss of information : therefore Convert the column 6 data as shown below:

Column 6 : the column 6 has the following ranges:

<i>Exact amount promised,</i>	<i>-----TRUST-----></i>	<i>1</i>
<i>More than promised</i>	<i>-----TRUST-----></i>	<i>1.5</i>
<i>Promise not fulfillable</i>	<i>-----DO NOT TRUST-----></i>	<i>-1.5</i>
<i>Less than promised</i>	<i>-----DO NOT TRUST-----></i>	<i>1</i>

NOTE:

After training is completed and predictions are made on the evaluation data-set. Combine the predicted classes as follows before submitting the prediction file on Innocentive for generating the score:

Prediction dataset

<i>1 and 1.5</i>	<i>----></i>	<i>Trust</i>
<i>-1 and - 1.5</i>	<i>----></i>	<i>Do Not Trust</i>

Results :

Table 17: Four classifications in desired output vector

Desired	Performance	epochs
---------	-------------	--------

Performance(MSE)	reached	
Less than 0.01	0.324	4

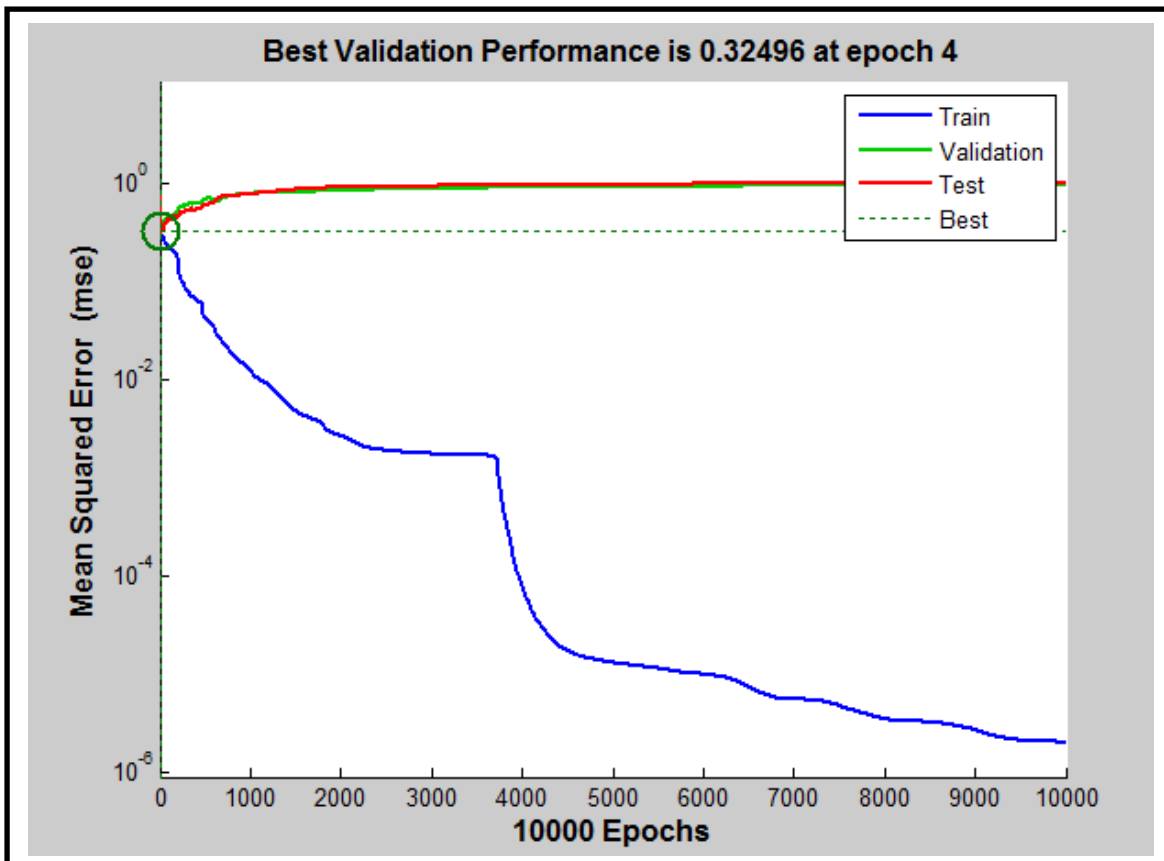


Figure 73 : zero vector dimensions/features Performance Graph

The results shows a increase in the mse of the network. which at start looked very promising due to the steep slope of the curve but the best validation performance reached by the network is 0.324 which is highest among all the previous algorithms up till now.

6.2.8 Method II: Learning with two times more don't trust vectors.

Implementation :

Another approach to preprocess the data to help increase the efficiency and accuracy of the network is increasing the ratio of minorities in the raw training data by duplicating those vectors. for example, in INNOVATE data set only 30 % of the input row vectors containing the desired output of Trust, therefore, boosted the input training data 2x Trust row vectors by duplicating the data.

Results :

Table 18: Learning with two times more don't trust vectors in training data set.

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.219	4

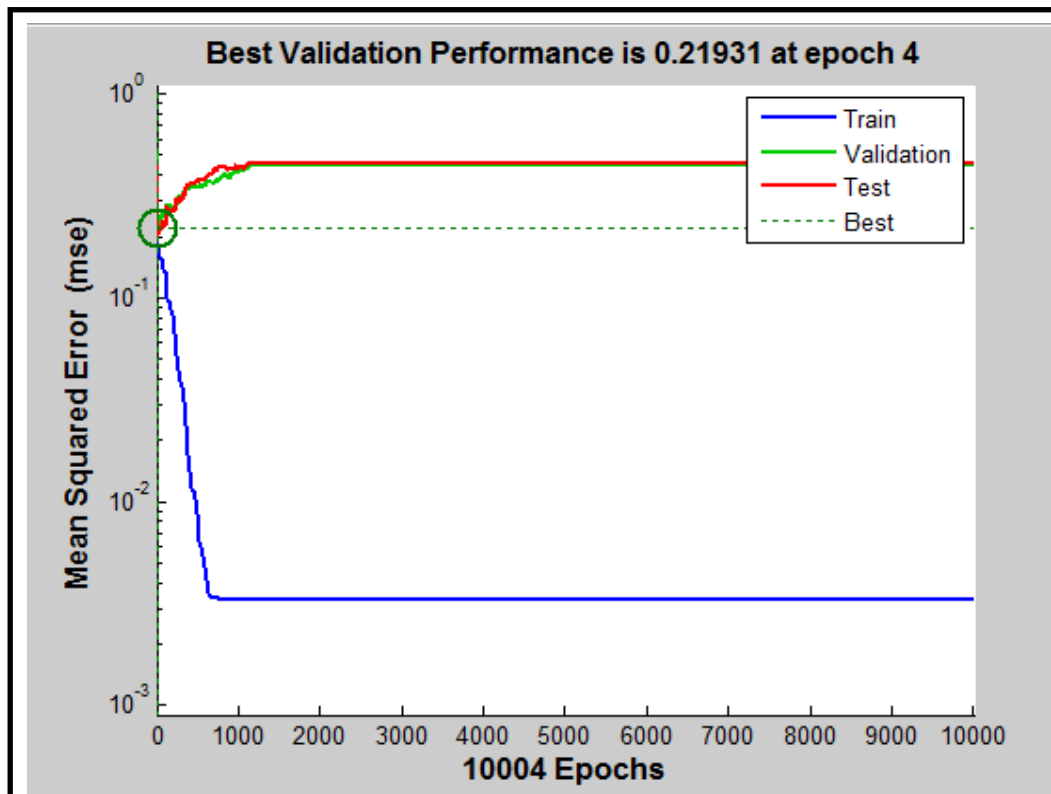


Figure 74 : four times more true vectors Performance Graph

The Performance was much better compared to method 1 approach but the results were not as close to those of training algorithms discussed earlier. The best validation performance mse able to reached was 0.219 in 4 epochs before the update in mse started became negligible/unaccountable for learning further learning. Although, This approach came out to be the fastest one converging in 4 iterations only.

6.2.9 Method III: Variation in threshold limit

Implementation

Up till now, as the network is trained with desired output considering 0 for don't trust and 1 for trust, therefore expected predicted values from the trained network, which falls between 0 and 1, were classified into trust and don't trust predictions using the threshold of 0.5, but looking at the output predicted from the network found out ,two scenarios , which provides possibilities of increasing the threshold from 0.5 to somewhere between 0.6 and 0,65 to get a increase in the number of correct predictions. In scenario one, the output range predicted by the trained network has the least predicted value of approx 0.2 and highest predicted value of 0.99.therefore taking the median for the predicted dataset, which comes out to be $(0.2+0.99)/2 \sim$ approx 0.60.In second scenario, found out the following distribution in the predicted dataset.

Between 0.5 to 0.59	Number of 0(Don't Trust) predictions	Number of 1(Trust) predictions
0.5 to 0.59	20	26
0.6 to 0.69	15	46

therefore, up till now, we were not including these large set of 0(Don't trust) classifications in our final prediction datasets. as we can see in the above table, if we move the threshold to a little higher value say up to approx 0.60 ,the final prediction file will be including this large number of (0)don't trust predictions(20) but along with a cost

of somewhat similar number of 1(trust) predictions(26) also .But as the cost of correct prediction of 0(don't trust) is higher than the cost of correct prediction of 1(trust) values, therefore, the above inclusion of values up till threshold of 0.60 should always be going to increasing the overall score despite having the similar number of correct trust predictions .

As shown in the below table which shows a successful increment in the overall performance of the network boosting the final score to **0.69** by configuring the increment in the threshold value of the network from 0.50 to 0.62.

Results :

Table 19: Variation in threshold limit

Threshold used for classifying True and False	Total number of don't trust rows in the final predictions	Score achieved
0.50	40	0.57
0.68	60	0.63
<u>0.62</u>	52	<u>0.69</u>

6.2.10 Method IV : Focusing on reduced set of training vectors

implementation :

Analyzing the results from various experiments, found one thing similar in the behavior of the trained neural networks both in the training phase and in the prediction phase.

Despite having a good mse the performance of the trained network was not that great. found out validation checks should be kept to a lower number. generated a more better mse this time but still the performance remained almost similar. Analyzing the predicted

values to find out what could have been the reason behind such behavior found out the following distribution of the values in the prediction results.

Predicted values	count
0 (upto e^{-6})	71
1 (from 0.8)	253
<u>0 <-----> 1</u>	<u>115</u>
Total	439

As it can be seen, most of the predictions are able to provide concrete results of either TRUST (1) or DON'T TRUST(0). This was the similar behavior predicted by network with different algorithms and configurations used. having most of the predictions coming out straightly into either true or false category but approx 100 values have a range between 0 and 1 ,which is making the difference in the overall performance of the network.

Therefore, there was a need to try and lookout if there will be increase in the performance by focusing training only on these row numbers .After done with the initial training with complete dataset, say network1, saved the weights and try training network with only considering those subset of rows which were earlier predicting range between 0 to 1, say network_2. Following were the results predicted from final neural network network_2.

Results :

Table 20: Focusing on reduced set of training vectors

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.216	20

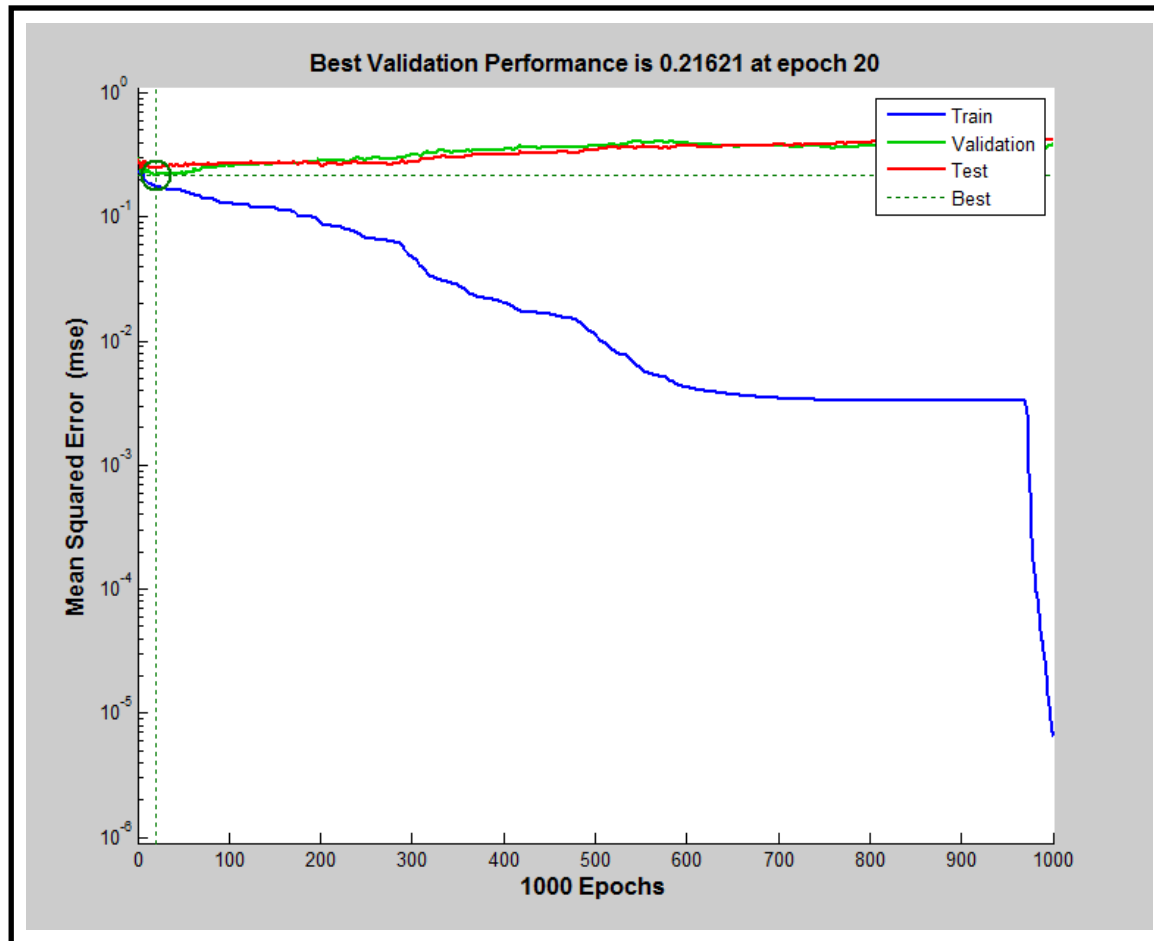


Figure 75 : Focusing on reduced set of training vectors

The best validation mse of 0.21 within 20 iterations achieved by this approach was almost similar to the method iv.

6.2.11 Method V: Increasing the validation and test dataset

implementation :

There is always a tradeoff between the percentage of training data assigned for the training of the network and the test or validation of the network. To improve the ability of the network to accurately predict for the unseen data, increased the test and validation

dataset percentage to 15% each. Following are the analyses of the approaches used, with having the best ones selected among them to generate the score from the Innocentive dashboard.

Training_data (Perf) : 70 %

Validation_data (Vperf) : 15%

Test_data (Tperf) : 15%

Results :

Table 21: Increasing the validation and test dataset percentage

	Configuration	Stop Condition	Performance	Score
1	trainscg__51neurons__tansig-tansig__learnngdm__ epochs500 __validation10000	epochs	best_perf: 0.1746 best_vperf: 0.2162 best_tperf: 0.2455	
2	trainscg__51neurons__tansig-tansig__learnngdm__ epochs1000 __validation10000	epochs	best_perf: 0.1897 best_vperf: 0.1924 best_tperf: 0.1992	0.57
3	trainscg__51neurons__tansig-purelin__learnngdm__ epochs10,000 __validation10,000	epochs	best_perf: 0.1678 best_vperf: 0.2105 best_tperf: 0.2289	
4	trainscg__219neurons__tansig-tansig__learnngdm__ epochs50,000 __validation10,000	validation	best_perf: 0.2040 best_vperf: 0.1845 best_tperf: 0.2051	

6.2.12 Method VI : Revising with important/ stringent vectors

implementation :

One of the way we can increase the performance of the network is by simulating the learning of the neural networks with that of how we as human beings learn, especially

during exam times. once we are done with our entire syllabus we find out the topics which we find hard to remember even after we have went through them while covering the whole book during the first time. Thus, marking out those topics and only go through these shortlisted set of topics instead of entire book to make our self more confident on these stringent /important topics. Similarly , in neural networks we follow the same analogy to try and improve the overall learning capability of the network. The approach used is mentioned below:

Now, there is a need to find out the stringent /important input training vectors, which the network is not able to get trained successfully as compared to other input row vectors. .

one method used to find out the stringent training vectors is to create multiple different networks with various configurations like, different training algorithms, different number of hidden neurons ,different transfer functions, variation in learning rate etc. After training completion, the predictions are made on the complete dataset by all the networks. All the wrong predictions were taken out from all the prediction sets and generated the frequency of each row vector from this wrong prediction datasets.

Following table is generated with the total count of row vector ids grouped together by their frequency of occurrence in the wrong predictions dataset generated through multiple experiments as mentioned above.

Frequency	Stringent input vector count
12	27
11	28
10	13
9	34
8	16
7	3
6	5
5	12
4	4
3	7
2	27

Scenario 1:

In this method, we will start with our base neural network which performed the best among the different algorithmic technique experimented. i.e scaled conjugate algorithm network, which was first trained for the complete dataset similar to the analogy of human beings reading the whole book for the first time. After the first training is complete ,say NN1, . Now, using NN1 as the base network and train the network completely only for the stringent set of training input vector generating a new network say NN2,in an analogy of the human beings doing revision of only the smaller set of most important/stringent topics at the end. Finally, NN2 will be used for the evaluation dataset for generating final predictions. following are the results of the above approach used where NN2 training was stopped after a particular number of iterations /mse to keep a check on the possibility of over training on the stringent vectors.

Results :**Table 22: Revising with important/ stringent vectors**

	Desired Performance (MSE/EPOCHS) on revising dataset	Performance reached on complete dataset	Epochs reached
1	Full	0.6248	446
2	50 iterations	0.5656	50
3	20 iterations	0.5936	20
4	5 iterations	0.5685	5
5	Mse 0.1	0.4684	16
6	Mse 0.135	0.4364	10
7	Mse 0.15	0.4854	6

Following are the results for the sixth network in the above table

Desired Performance(MSE)	Performance reached on revising dataset	epochs
Less than 0.125	0.135	10

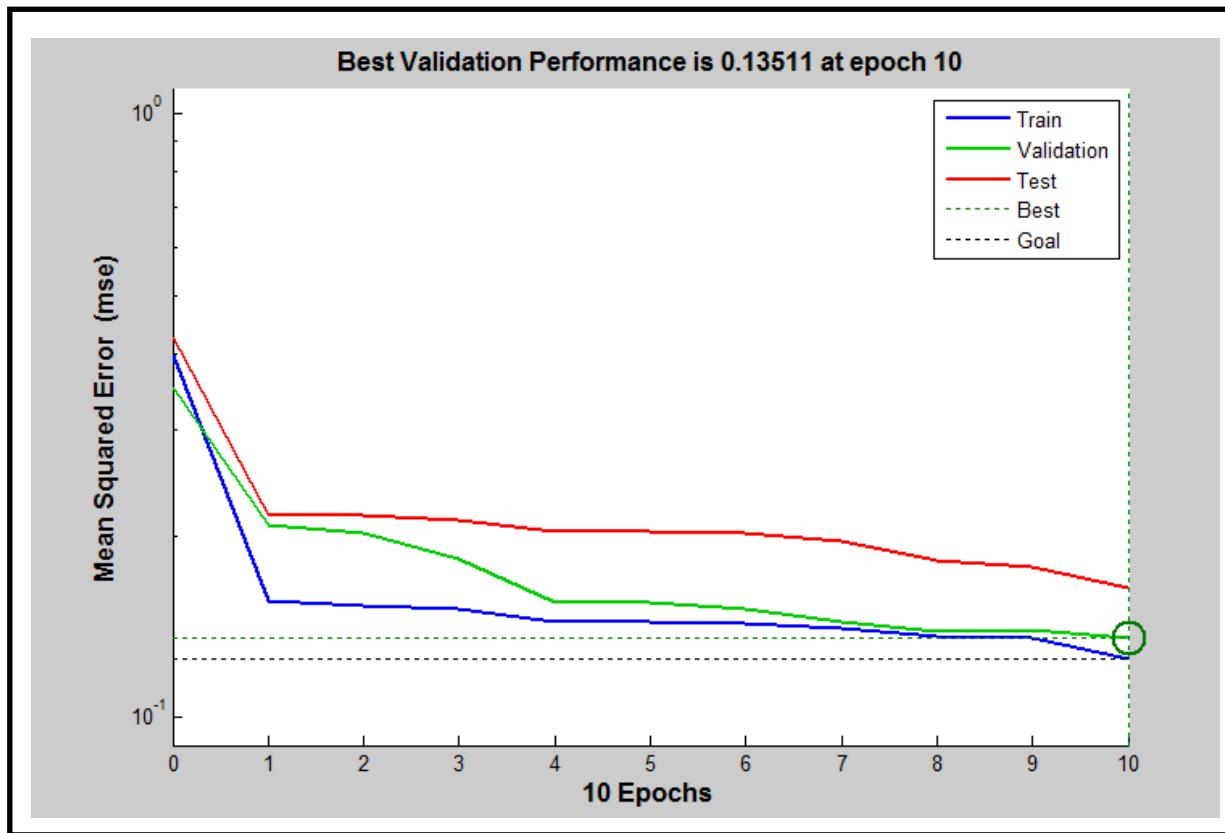


Figure 76 : Revising with important/ stringent vectors scenario 1

The best validation mse of 0.135 within 10 iterations achieved by this approach is the performance of the NN2 network on the revising dataset. The best final performance of the network on the complete dataset is 0.43 in 10 iteration.

Scenario 2 :

In scenario 1, NN2 was generated from NN1 after training on the stringent vectors for a particular amount of iterations /mse to avoid overtraining. Here, in this method, another level of neural network is generated, say NN3, using nn2 as its base network. The concept behind this approach is once the network is retrained for the stringent vectors, it may happen that the network may become biased towards those smaller set of input row vectors. Taking analogy from the human beings by going over the entire book once again after done with revising of the important topics. Finally, NN3 will be used for the evaluation dataset for generating final predictions. Following are the results of the above approach used where the intermediate network NN2 was either trained for the full

number of iterations or stopped after a particular number of iterations /mse to keep a check on over-training on the stringent vectors followed by final NN3 training on complete dataset .

Results :

Table 23: Revising with important/ stringent vectors scenario 2

	Desired Performance (MSE/EPOCHS) on NN1	Desired Performance (MSE/EPOCHS) on NN2	Performance reached on complete dataset	Epochs reached
1	Full	Full	0.1856	11
2	Mse 0.25	Full	0.1915	53
3	Full	20 iterations	0.2722	20

Following are the results for the first network in the above table

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.1856	11

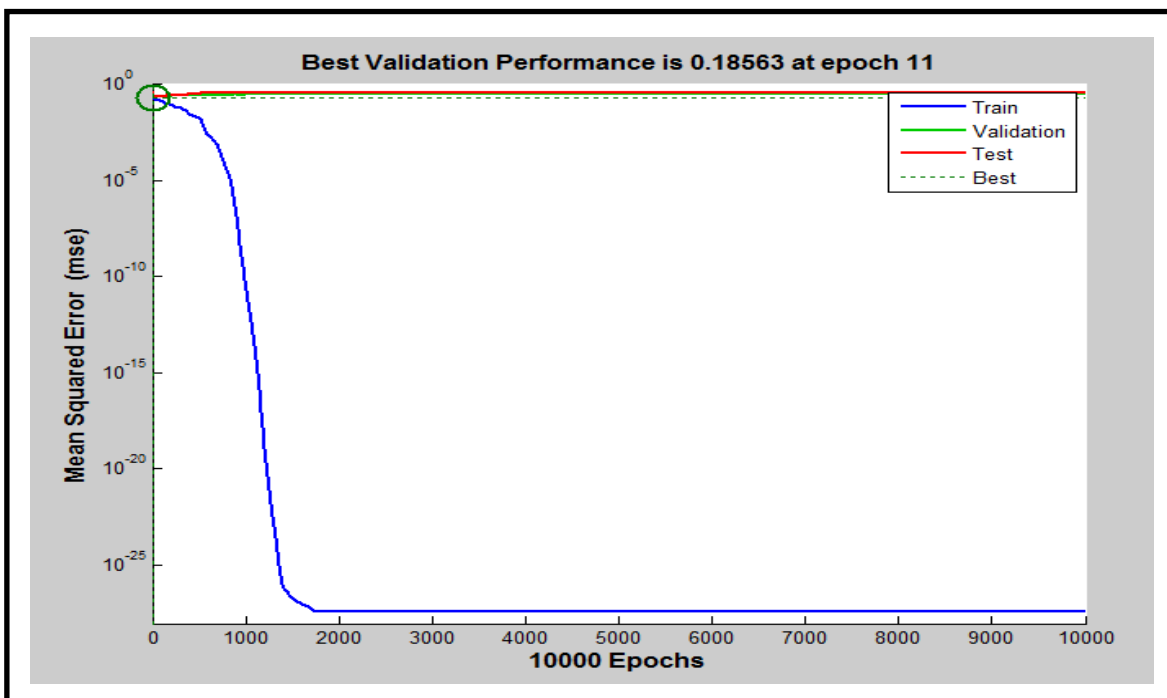


Figure 77 : Revising with important/ stringent vectors scenario 2

The best validation mse of 0.185 within 11 iterations achieved by this approach is the best performance achieved by any network up till now.

6.2.13 Method VII : Excluding Noise from the network

implementations :

There are possibilities that the desired output provided in the training dataset may not be always 100% correct. Such input row vectors are called noise in the training dataset. Presence of such data deviates the network training with false weight updates. Therefore, it is very necessary to remove the noise from the training dataset before feeding it into the network. One of the method to find out the noisy input training vectors is similar to the method vi , i,e create multiple different networks with various configurations and carry out the predictions on the training dataset itself. Analyze and compare the predictions from all the networks, select only those rows numbers whose predictions were constantly fluctuating between Trust and Don't trust. Finally, the highest frequency fluctuation are the strong candidate for the noisy input training vectors.

Following are the results of the training carried out on the dataset exclusive of noisy input row vectors .

Results :

Table 24: Excluding Noise from the network

	Desired Performance (MSE/EPOCHS) on NN1	Desired Performance (MSE/EPOCHS) on NN2	Performance reached on complete dataset	Epochs reached
1	Full	-	0.1955	59
2	Full	Full	0.145	18

Following are the results for the second network in the above table

Desired Performance(MSE)	Performance reached	epochs
Less than 0.01	0.145	18

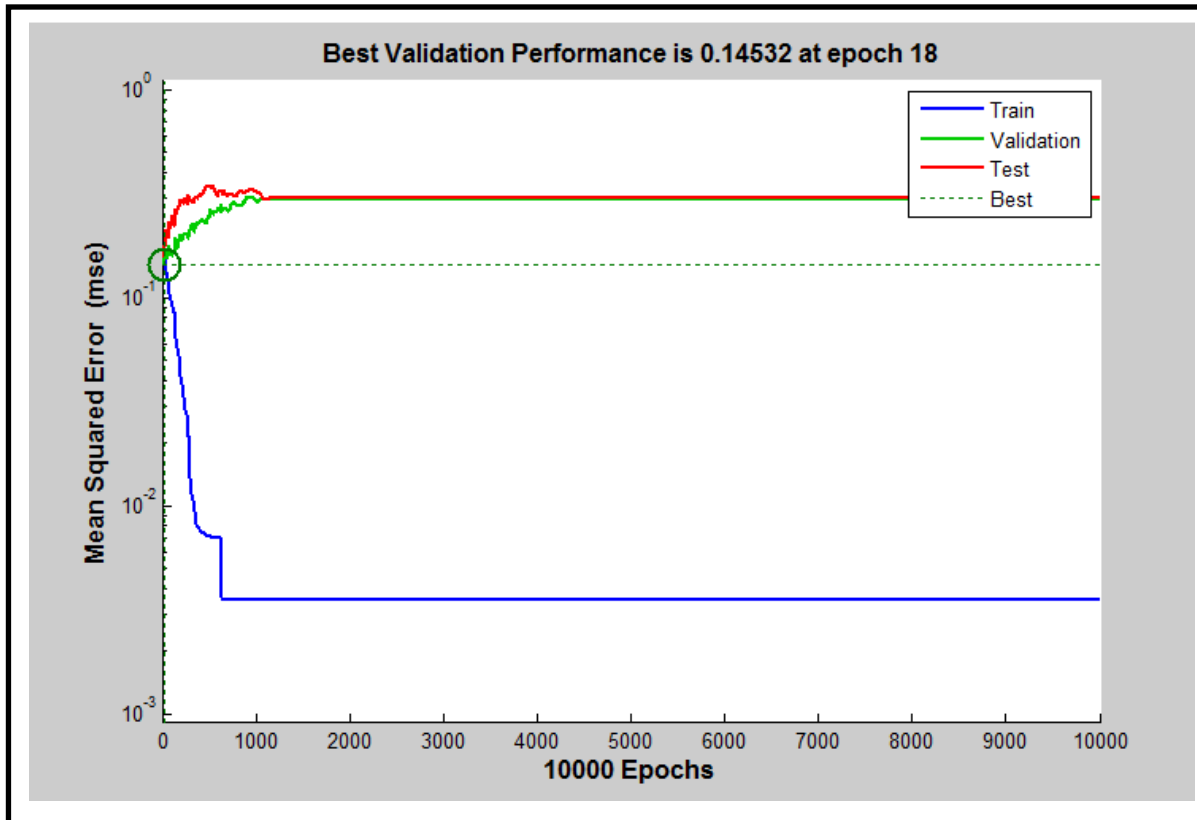


Figure 78 : Excluding Noise from the network

The best validation mse of 0.145 within 18 iterations achieved by this approach crossed the performance achieved by the method vi significantly and become the best network up till now.

6.2.14 Method VIII : Merging method VI and VII

implementations :

This method combines the last two approaches of revision and noise exclusion, As these two analyzing techniques has achieved a significant mse improvement. In this approach, starting with NN1 as an initial trained network, the network is trained with noise exclusive method producing a intermediate trained network NN2. now, NN2 will be used to train with the revising method and generate another neural network NN3. Thereafter, NN3 is fed with the complete dataset for the purpose of generalized training generating NN4. Finally, NN4 is used for the predictions on the evaluation datasets. following are the results of the above approach:

Results :

Table 25: merging method VI and VII

Desired Performance(MSE)	Desired Performance (MSE/EPOCHS) on NN2	Desired Performance (MSE/EPOCHS) on NN3	Desired Performance (MSE/EPOCHS) on NN4	Performance reached	epochs
Less than 0.01	full	full	full	0.127	609

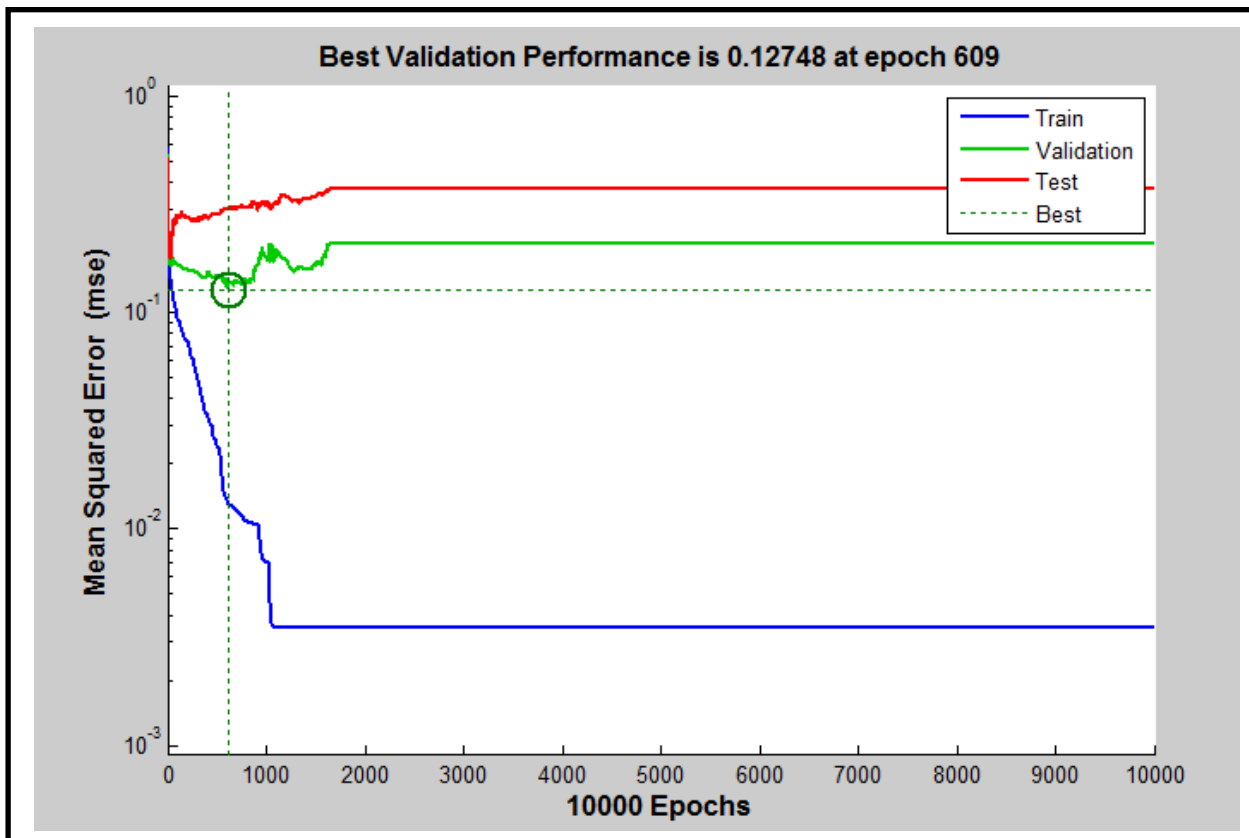


Figure 79 : Merging method VI and VII

The best validation mse of 0.127 achieved by this network was a significant improvement over the last two already best approaches and thus producing the best network up till now.

Comparison and Conclusion

Table 26 : Table of comparison

	Approach	Performance	Epochs
1	Gradient descent back-propagation	0.198	517
2	Resilient back-propagation	0.20	66
3	Scaled conjugate gradient back-propagation	0.192	43
4	Momentum back-propagation	0.201	868
5	Adaptive Learning Rate back-propagation	0.22	70
6	Momentum and Adaptive Learning Rate back-propagation	0.22	98
7	Four classifications in desired output vector	0.324	39
8	Two times more don't trust vectors	0.219	4
9	Variation in threshold limit	0.1924	43
10	Focusing on Reduced set of training vectors	0.216	20
11	Increasing the validation and test dataset	0.1924	43
12	Revising with important/ stringent vectors scenario 1	0.4364	10
13	Revising with important/ stringent vectors scenario 2	0.1856	11
14	Noise exclusion from the network	0.145	18
15	<u>Merging method VI and VII</u>	<u>0.127</u>	<u>609</u>

The table shows the comparison of different learning algorithms and techniques used along with the performance achieved by the network respectively.

Starting with performing different experiments with all the various training algorithms to find out the best algorithm with the lowest possible mse on the training dataset. The results from gradient descent and scaled conjugate gradient descent provided good mse as compared to other training algorithms. The scaled conjugate gradient descent produced the least mse of 0.192 among all the training algorithms ,therefore continued training with more advanced analytical methods using this particular algorithms configurations as the basis for the experiments

In the series of different analytical models ,initially very promising results were achieved with the help of the variation in the threshold technique which helped the score increase from 0.54 earlier to 0.69 on the Innocentive leadership dashboard.

In later stages of the analysis, the technique of retraining the already trained networks for the important rows found very effective in reducing the mse from 0.192 to 0.185.

The exclusion of noise from the network further took mse to one of the best possible value so far achieved.

Finally, as the last two approaches looked very promising in increasing the network performance therefore merge both the techniques resulted in a level 4 neural network which produced the lowest ever mse of 0.127 only. which was much closer to the desired mse of 0.1 .

7. Future Work

As the number of features in the application dataset increase, like in the case of large scale applications , the layers size(neurons) of the artificial neural network needs to be increased to accommodate the increased dimensions of the input dataset. After certain point, the network size becomes so huge that it becomes almost infeasible to be implemented efficiently because of the increased complexity induced due to the exponential growth of the inter-connections among the nodes(neurons) in the network. This phenomenon is generally phrased as the " the curse of dimensionality" in the field of machine learning. Therefore, there is a need to come out with an algorithm to process large dataset efficiently keeping the neural network size considerable small by optimizing the numbers of neurons and the interconnection between them.. The future work will be on optimizing the neural networks. There are several paper published with different approaches to achieve this, but the two most promising ones are genetic algorithms and modular design technique.

8. References

- [1] Anderson, J. R., Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (1983). *Machine learning: an artificial intelligence approach*. Palo Alto, Calif.: Tioga Pub. Co..
- [2] Bishop, C. M. (1998). *Neural networks and machine learning*. Berlin: Springer.
- [3] Chauvin, Y. (1994). *Backpropagation theory, architectures, and applications*. Hillsdale, N.J.: Lawrence Erlbaum Associates.
- [4] Fausett, L. V. (1994). *Fundamentals of neural networks: architectures, algorithms, and applications*. Englewood Cliffs, NJ ; Delhi: Prentice-Hall ; Dorling Kindersley.
- [5] Graupe, D. (2007). *Principles of artificial neural networks* (2nd ed.). New Jersey: World Scientific.
- [6] Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, Mass.: MIT Press.
- [7] Kwon, S. J. (2011). *Artificial neural networks*. New York: Nova Science Publishers.
- [8] Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. Cambridge, Mass.: MIT Press.
- [9] Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill.
- [10] Pfister, M., & Rojas, R. (1993). *Backpropagation algorithms: their characteristics and efficiency*. Berlin: Freie Univ., Fachbereich Mathematik.
- [11] *Proceedings / International Work Conference on Artificial and Natural Neural Networks, IWANN '99*. (1999). Berlin: Springer.
- [12] Reed, R. D., & Marks, R. J. (1999). *Neural smithing supervised learning in feedforward artificial neural networks*. Cambridge, Mass.: MIT Press.
- [13] Schalkoff, R. J. (1997). *Artificial neural networks*. New York: McGraw-Hill.

9. Appendix

9.1 Algorithm to process the predicted output data:

```
/*algorithm to process the predicted output data*/
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class DataPredicted {

public static void main(String [] args) throws Exception{

    InputStream file_input_stream;
    BufferedReader buffer_reader;

    try {

        file_input_stream = new FileInputStream("Predicted_output.txt");
        buffer_reader = new BufferedReader(new InputStreamReader(file_input_stream));
        FileOutputStream file_stream = new FileOutputStream("JMDAUM_outputPredicted_IDs.txt");
        BufferedWriter out = new BufferedWriter(file_stream);

        String L = "";
        Double Threshold_true = 0.7;
        int i =1;

        while (( L = buffer_reader.readLine()) != null)
        {
            L=L.trim();
            if( Double.parseDouble(L)>=Threshold_true)
                out.write((400000+i)+"\n");
            i++;
        }

        out.close();
    }catch(Exception ex)
    {
        ex.printStackTrace();
    }
}
```

9.2 Algorithm to prepare the training input dataset :

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;

public class InputData {
    public static void main(String [] args) throws Exception{
        InputStream file_input_stream;
        BufferedReader buffer_reader;

        try {
            file_input_stream = new FileInputStream("traintgmc.csv");
            buffer_reader = new BufferedReader(new InputStreamReader(file_input_stream));
            FileWriter file_stream = new FileWriter("input_tgmctrain.txt");
            BufferedWriter output = new BufferedWriter(file_stream);
            FileWriter file_stream2 = new
            FileWriter("output_tgmctrain.txt");
            BufferedWriter output2 = new BufferedWriter(file_stream2);

            String L = "";

            while ((L = buffer_reader.readLine()) != null)
            {
                L = L.trim();
                if(L.contains("false"))
                {
                    L=L.replace("false","");
                    output.write(L);
                    output2.write('0');
                }
                else
                {
                    L=L.replace("true","");
                    output.write(L);
                    output2.write('1');
                }
                output.write("\n");
                output2.write("\n");
            }
            output.close();
            output2.close();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
}
```

9.3 Algorithm to double the true vectors in input dataset data :

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;

public class DoubleTrueVectors {
    public static void main(String [] args) throws Exception{

        InputStream file_input_stream;
        BufferedReader buffer_reader;
        InputStream fis_intermdiate;
        BufferedReader br_intermdiate;
        try {
            file_input_stream = new FileInputStream("train_tgmc.csv");
            buffer_reader = new BufferedReader(new InputStreamReader(file_input_stream));

            FileWriter file_stream = new FileWriter("intermediate_tgmctrain.txt");
            BufferedWriter output = new BufferedWriter(file_stream);
            String L = "";

            while ((L = buffer_reader.readLine()) != null)
            {
                if(L.contains("true"))
                    output.write(L+"\n");
            }
            output.close();
            fis_intermdiate = new FileInputStream("intermediate_tgmctrain.txt");
            br_intermdiate = new BufferedReader(new InputStreamReader(fis_intermdiate));
            file_stream = new FileWriter("2xTrueVector_tgmctrains.txt");
            output = new BufferedWriter(file_stream);
            L = "";

            int i = 1;
            while ((L = buffer_reader.readLine()) != null)
            {
                output.write(L+"\n");
                if(i==10 && (L = br_intermdiate.readLine()) != null )
                {
                    i=0;
                    output.write(L+"\n");
                }
                i++;
                output.write(',');
            }
            output.close();
        }catch(Exception ex){
            ex.printStackTrace(); }
    }
}

```