Artificial Neural Network

This project is built on an artificial neural network. A simple perceptron has been created which can hold unlimited number of input, hidden and output neurons.

Single layer perceptron can only solve linearly separable problems and thus it cannot solve complex problems like digit recognitions and others.

Thus, we have implemented hidden layer as well.

We have trained and tested datasets of "MNIST dataset" & "Arabic Handwritten Digits" using our artificial neural network.

Our ANN are made up of below configuration for each dataset.

- MNIST dataset – Digit recognition from 0 to 9

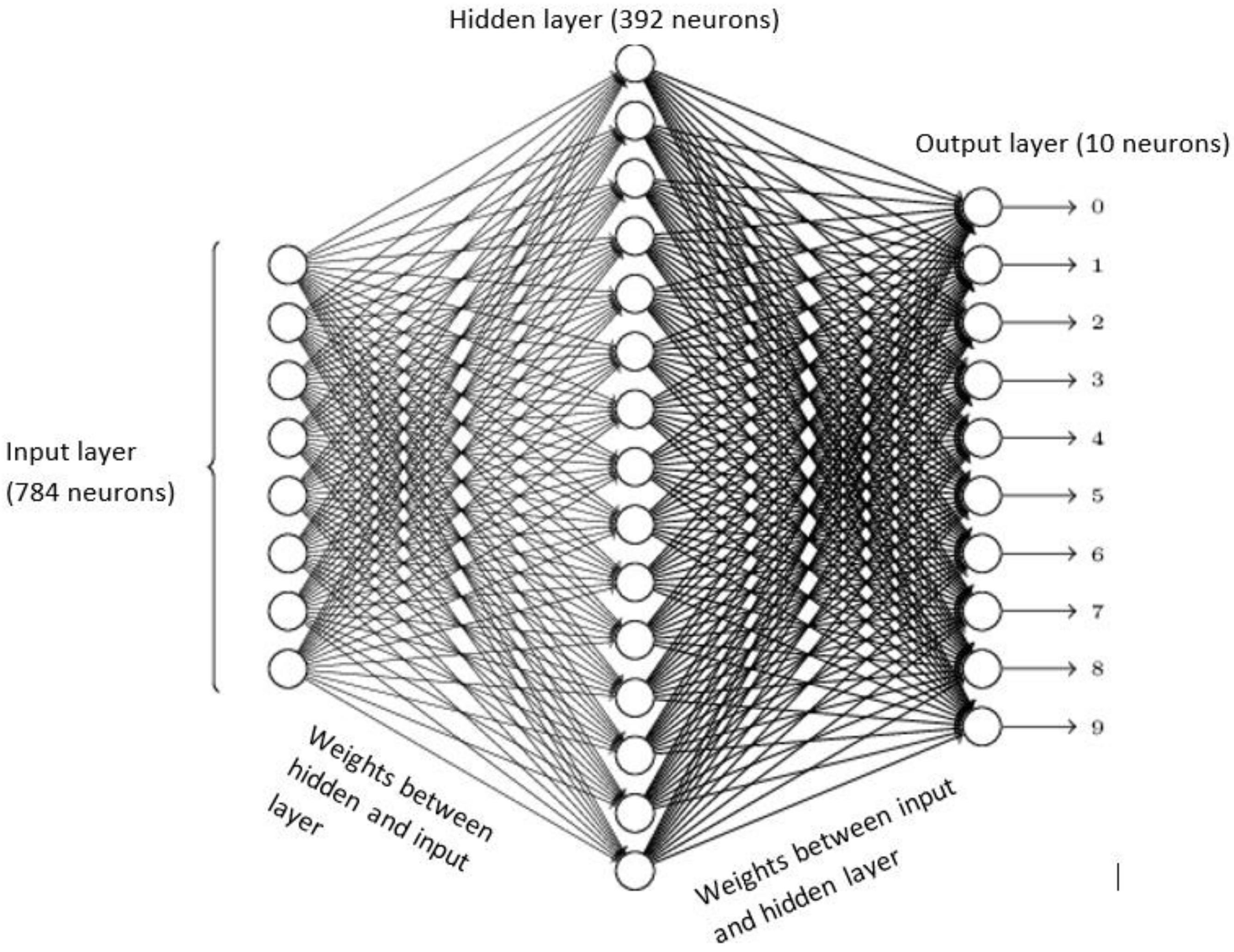
✓ 1st layer is the input layer which is composed of 784 neurons,
✓ 2nd layer is the hidden layer which has 392 neurons,
✓ 3rd layer, namely the output layer has 10 neurons.

- Arabic Handwritten digits dataset – digit recognition from 0 to 9

✓ 1st layer is the input layer which is composed of 784 neurons,
✓ 2nd layer is the hidden layer which has 392 neurons,
✓ 3rd layer, namely the output layer has 10 neurons.

Three classes have been developed for this project
✓ Driver – the main class (used to read the input dataset in CSV format & graph creation
✓ Matrix – used for matrix calculation (addition, multiplication, subtraction, transpose)
✓ Neural Network – creates input, hidden and output neurons; adjusts the weights between input/hidden layer and weights between hidden and output layer, adjusting the bias and setting up the learning rate.

Below is the structural representation of our ANN


Hidden layer (392 neurons)
Output layer (10 neurons)
Input layer
(784 neurons)
0
1
2
3
4
5
6
7
8
9
Weights between hidden and input layer
Weights between input and hidden layer

Input layer: Input layer consist of the total number of pixels of the digits which are 784(28*28)

Hidden layer: half of the input neurons

Output layer: it is list of possible outcomes. In our case it is 10

Matrix class: This class is used as a library in our project. Matrix object is 2D array with rows and columns. Below is the list of functions created inside Matrix class.

- ✓ randomizeMatrix() – used to randomize all the elements of 2D array between -1 & 1
- ✓ multiplyMatrixByScalar() – used to multiply 2 matrices element by element
- ✓ add2Matrix() – adds 2 matrices
- ✓ matrixProductByObjects() – used to dot product between 2 matrices
- ✓ transposeMatrix() – transposes matrix
- ✓ fromArrayToMatrix() – converts array into matrix format
- ✓ convertMatrixToArray() – converts matrix into an array
- ✓ activationFunction() – sigmoid function is applied to each element of matrix
- ✓ sigmoid(int x) – returns 1 / (1 + Math.exp(-x))
- ✓ derivativeOfSigmoid() – each element of matrix is changed to y * (1 – y) format

Neural Network class: This class is responsible for setting below parameters

- ✓ Input neurons
- ✓ Hidden neurons
- ✓ Output neurons
- ✓ Weights between input and hidden neurons (random weights)
- ✓ Weights between hidden and output neurons (random weights)
- ✓ Hidden bias – bias for hidden neurons
- ✓ Output bias – bias for output neurons
- ✓ Learning rate – 0.005

- ✓ train() – This function is used to train the neural network by first trying to predict the output of given inputs and target; then calculating the errors. The gradient will be calculated from error which is the derivative of sigmoid function of errors. And then the gradient is multiplied with learning rate. The delta weights are calculated and added with original weights.
- ✓ Feedforward function - This function is used to predict the output of the inputs that we have trained.

Below are the steps we used in training a neuron

1) Provide the perceptron with inputs along with known answer

2) Ask the perceptron to predict an answer

3) Compute the error (difference of actual outcome and predicted outcome)

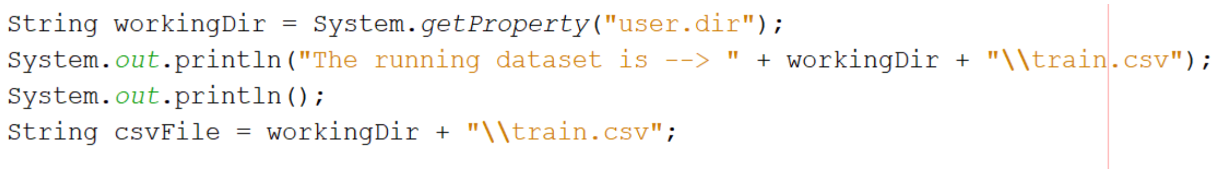
4) Calculate the output and hidden gradient

5) Adjust all the weights according to the error and add biases

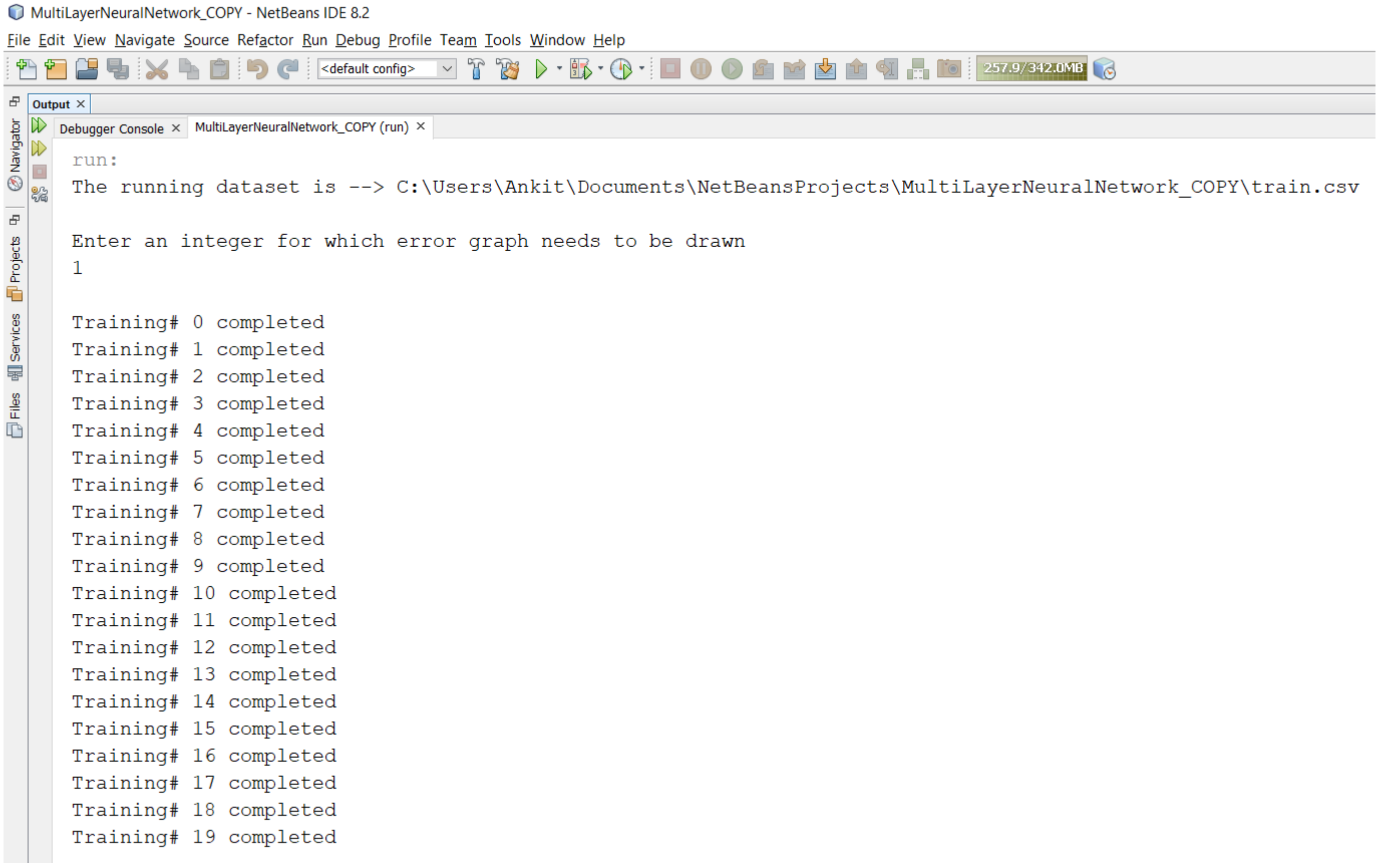
6) Return to step 1 and repeat

1) Results for MNIST dataset:

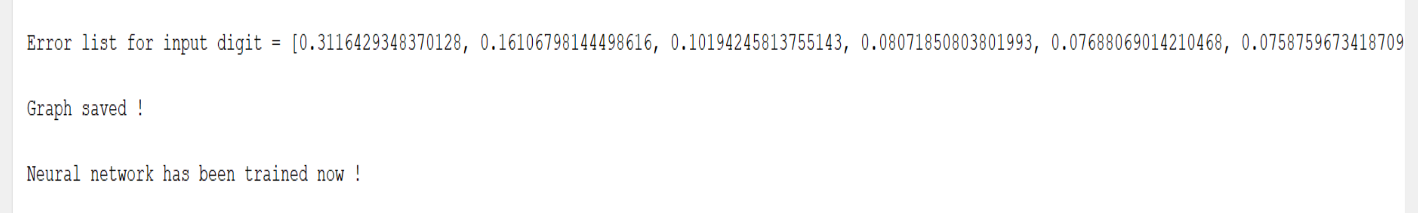
We ran the "train" file for MNIST data for 20 times and predicted the output from 1002 to 1021 row.



```
String workingDir = System.getProperty("user.dir");
System.out.println("The running dataset is --> " + workingDir + "\\train.csv");
System.out.println();
String csvFile = workingDir + "\\train.csv";
```

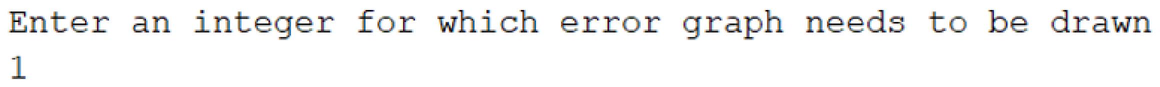

MultiLayerNeuralNetwork_COPY - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
257.9/342.0MB
Output
Debugger Console
MultiLayerNeuralNetwork_COPY (run)
Navigator
Projects
Services
Files

```
run:
The running dataset is --> C:\Users\Ankit\Documents\NetBeansProjects\MultiLayerNeuralNetwork_COPY\train.csv

Enter an integer for which error graph needs to be drawn
1

Training# 0 completed
Training# 1 completed
Training# 2 completed
Training# 3 completed
Training# 4 completed
Training# 5 completed
Training# 6 completed
Training# 7 completed
Training# 8 completed
Training# 9 completed
Training# 10 completed
Training# 11 completed
Training# 12 completed
Training# 13 completed
Training# 14 completed
Training# 15 completed
Training# 16 completed
Training# 17 completed
Training# 18 completed
Training# 19 completed
```



```
Error list for input digit = [0.3116429348370128, 0.16106798144498616, 0.10194245813755143, 0.08071850803801993, 0.07688069014210468, 0.0758759673418709

Graph saved !

Neural network has been trained now !
```

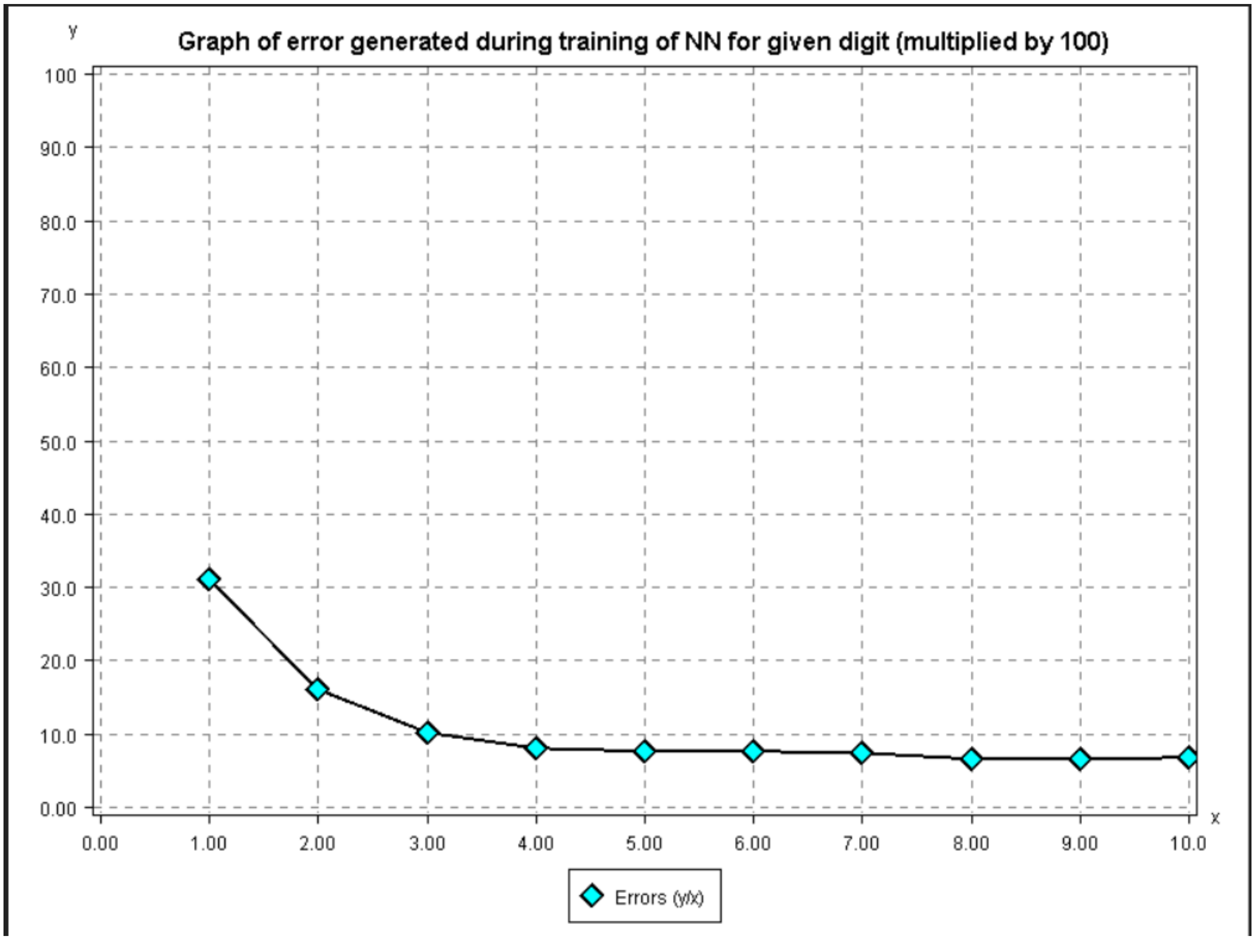
Also, we ask the user for an input for which system will create error graph.



```
Enter an integer for which error graph needs to be drawn
1
```

System will plot error graph of 10 iterations/error generated during prediction of each input. i.e. during each iteration, the difference between actual outcome and predicted outcome is plotted. The graph is saved in the directory of project.

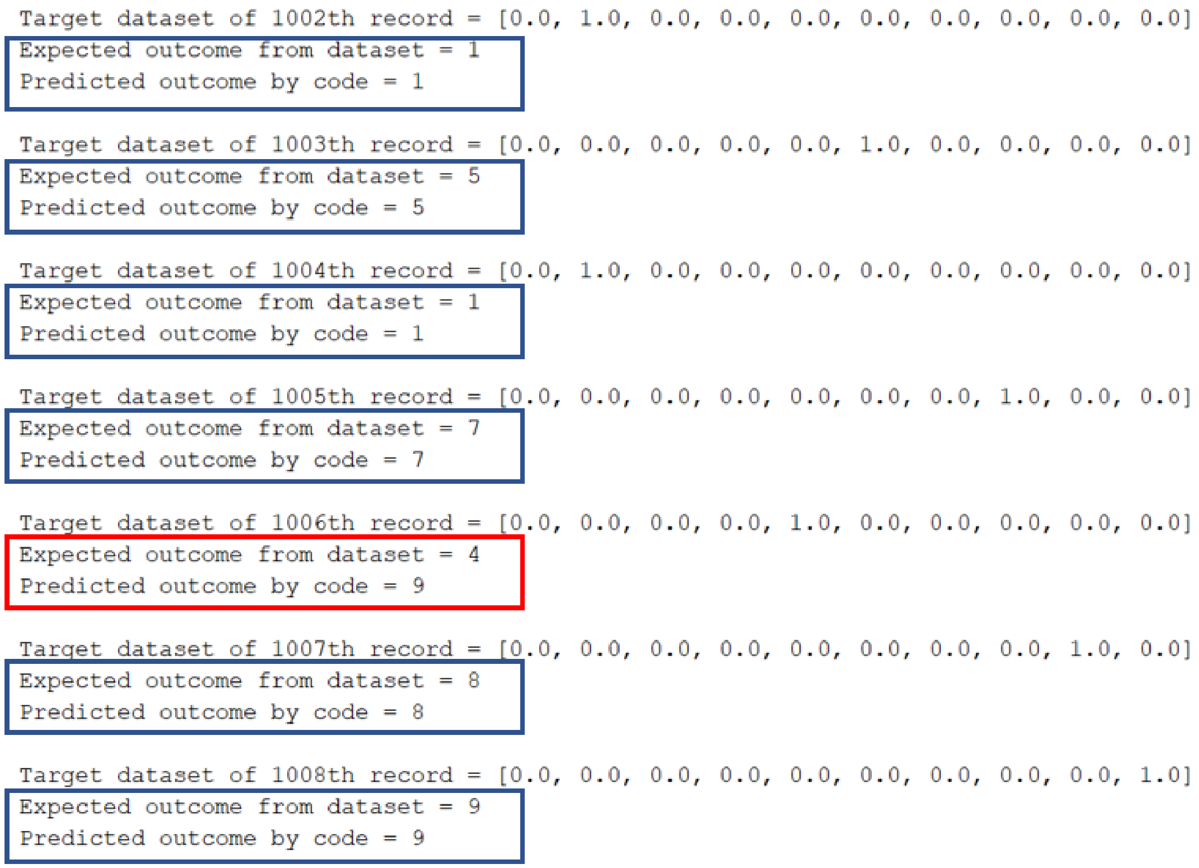
For display purpose, we multiplied each error with 100 so that points can be distinguishable in the graph.



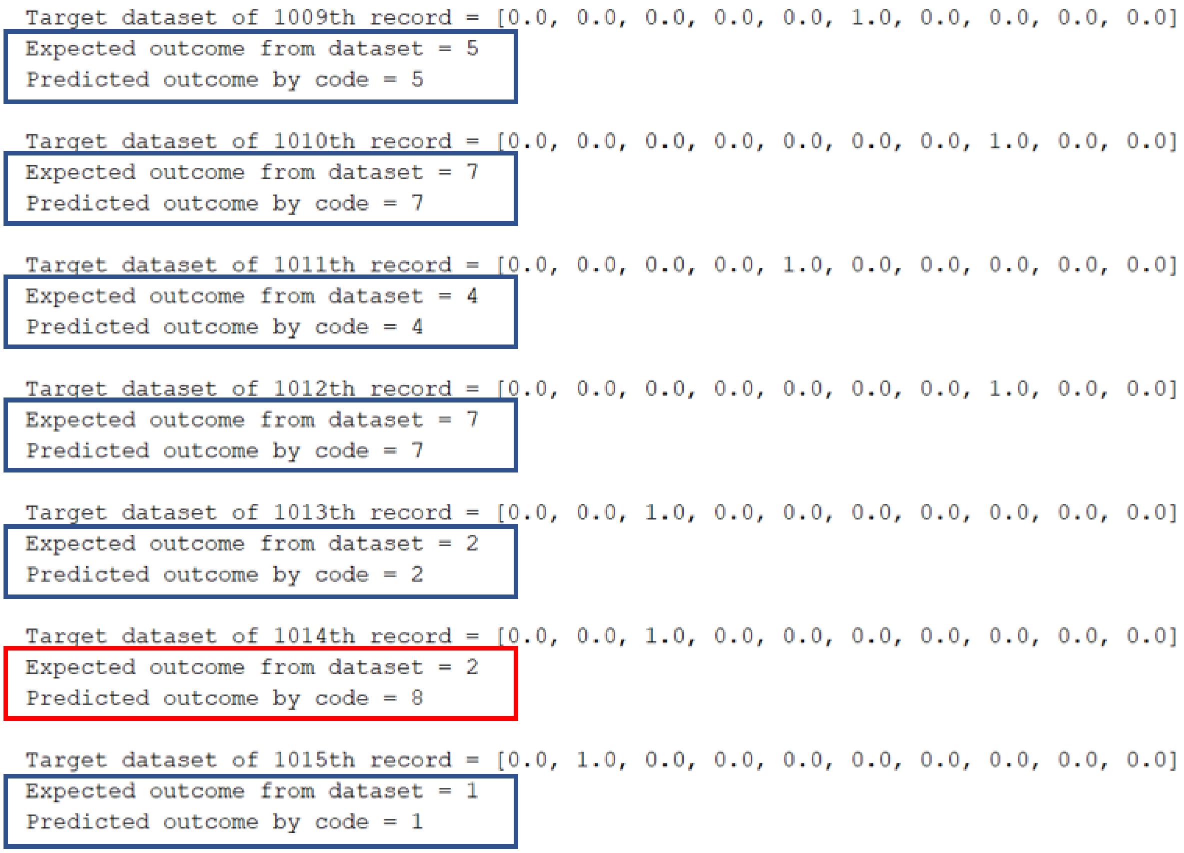
Below is the snapshot of output. The 1st line corresponds to array output where the index of array represents actual digit.

The steps marked in blue represents 100% accuracy where expected and actual outcome are same.

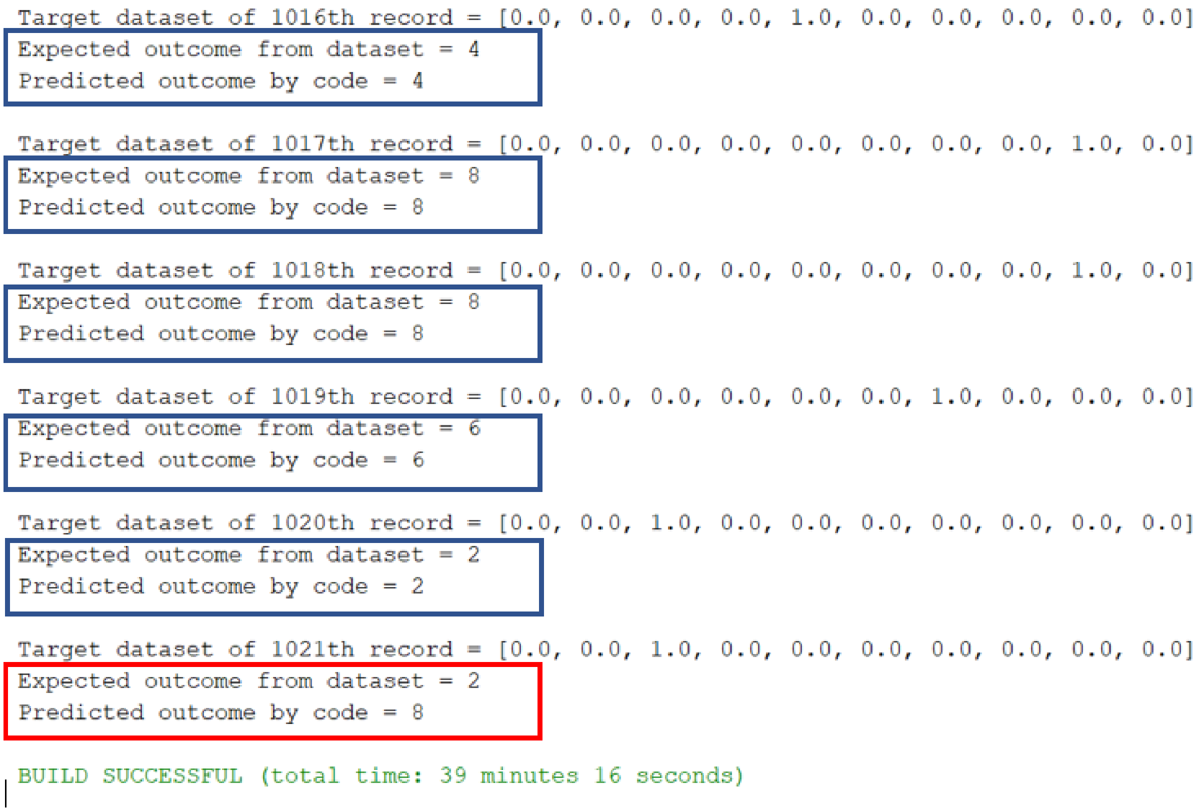
And the steps marked in red represents 0% accuracy where expected and actual outcome differ.



```
Target dataset of 1002th record = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 1
Predicted outcome by code = 1

Target dataset of 1003th record = [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 5
Predicted outcome by code = 5

Target dataset of 1004th record = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 1
Predicted outcome by code = 1

Target dataset of 1005th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
Expected outcome from dataset = 7
Predicted outcome by code = 7

Target dataset of 1006th record = [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 4
Predicted outcome by code = 9

Target dataset of 1007th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
Expected outcome from dataset = 8
Predicted outcome by code = 8

Target dataset of 1008th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
Expected outcome from dataset = 9
Predicted outcome by code = 9
```



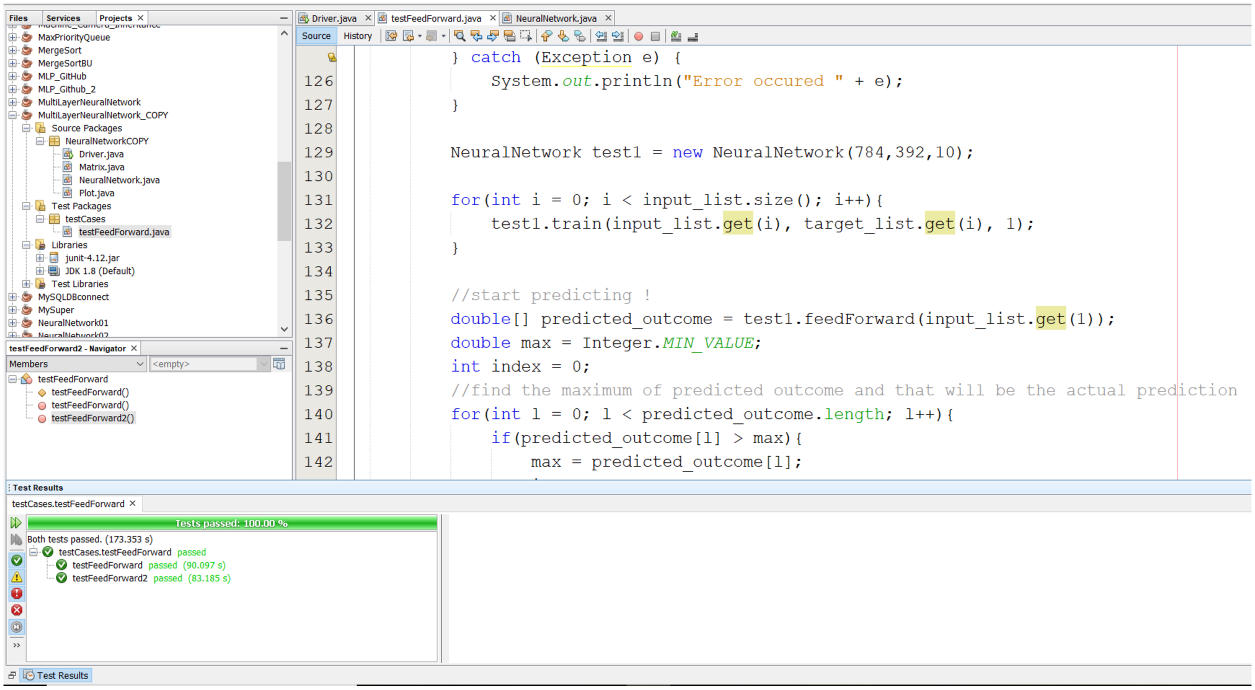
```
 Target dataset of 1009th record = [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
 Expected outcome from dataset = 5
 Predicted outcome by code = 5

 Target dataset of 1010th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
 Expected outcome from dataset = 7
 Predicted outcome by code = 7

 Target dataset of 1011th record = [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
 Expected outcome from dataset = 4
 Predicted outcome by code = 4

 Target dataset of 1012th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
 Expected outcome from dataset = 7
 Predicted outcome by code = 7

 Target dataset of 1013th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
 Expected outcome from dataset = 2
 Predicted outcome by code = 2

 Target dataset of 1014th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
 Expected outcome from dataset = 2
 Predicted outcome by code = 8

 Target dataset of 1015th record = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
 Expected outcome from dataset = 1
 Predicted outcome by code = 1
```



```
Target dataset of 1016th record = [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 4
Predicted outcome by code = 4

Target dataset of 1017th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
Expected outcome from dataset = 8
Predicted outcome by code = 8

Target dataset of 1018th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
Expected outcome from dataset = 8
Predicted outcome by code = 8

Target dataset of 1019th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 6
Predicted outcome by code = 6

Target dataset of 1020th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 2
Predicted outcome by code = 2

Target dataset of 1021th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 2
Predicted outcome by code = 8

BUILD SUCCESSFUL (total time: 39 minutes 16 seconds)
```

Accuracy for the above dataset - 85.71%

Below is the snapshot of testcases created. We predict a single digit and the result is shown by the testcase.

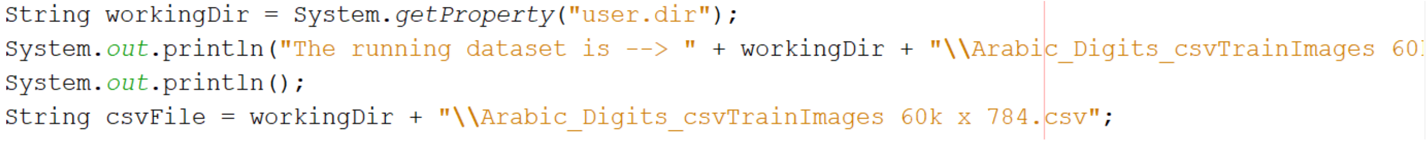


```
} catch (Exception e) {
    System.out.println("Error occured " + e);
}

NeuralNetwork test1 = new NeuralNetwork(784,392,10);

for(int i = 0; i < input_list.size(); i++){
    test1.train(input_list.get(i), target_list.get(i), 1);
}

//start predicting !
double[] predicted_outcome = test1.feedForward(input_list.get(1));
double max = Integer.MIN_VALUE;
int index = 0;
//find the maximum of predicted outcome and that will be the actual prediction
for(int l = 0; l < predicted_outcome.length; l++){
    if(predicted_outcome[l] > max){
        max = predicted_outcome[l];
```

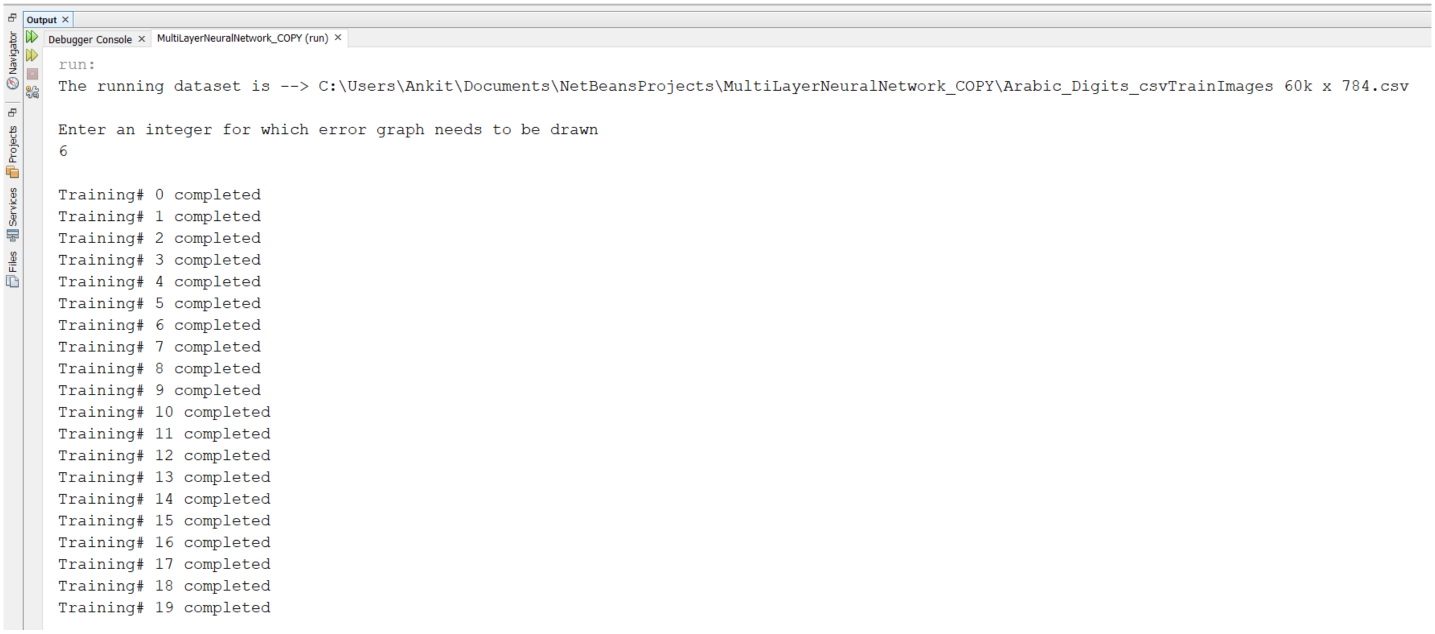
```
Tests passed: 100.00 %
Both tests passed. (173.353 s)
  testCases.testFeedForward passed
    testFeedForward passed (90.097 s)
    testFeedForward2 passed (83.185 s)
```

2) Result of Arabic Handwritten digits:

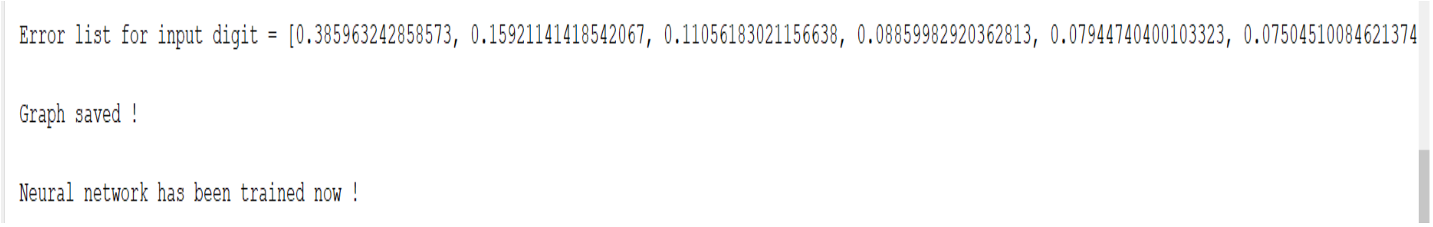
We ran the “train” file for Handwritten data for 20 times and predicted the output from 1002 to 1021 row.



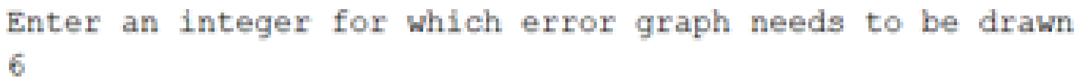
```
String workingDir = System.getProperty("user.dir");
System.out.println("The running dataset is --> " + workingDir + "\\Arabic_Digits_csvTrainImages 60
System.out.println();
String csvFile = workingDir + "\\Arabic_Digits_csvTrainImages 60k x 784.csv";
```



```
run:
The running dataset is --> C:\Users\Ankit\Documents\NetBeansProjects\MultiLayerNeuralNetwork_COPY\Arabic_Digits_csvTrainImages 60k x 784.csv

Enter an integer for which error graph needs to be drawn
6

Training# 0 completed
Training# 1 completed
Training# 2 completed
Training# 3 completed
Training# 4 completed
Training# 5 completed
Training# 6 completed
Training# 7 completed
Training# 8 completed
Training# 9 completed
Training# 10 completed
Training# 11 completed
Training# 12 completed
Training# 13 completed
Training# 14 completed
Training# 15 completed
Training# 16 completed
Training# 17 completed
Training# 18 completed
Training# 19 completed
```



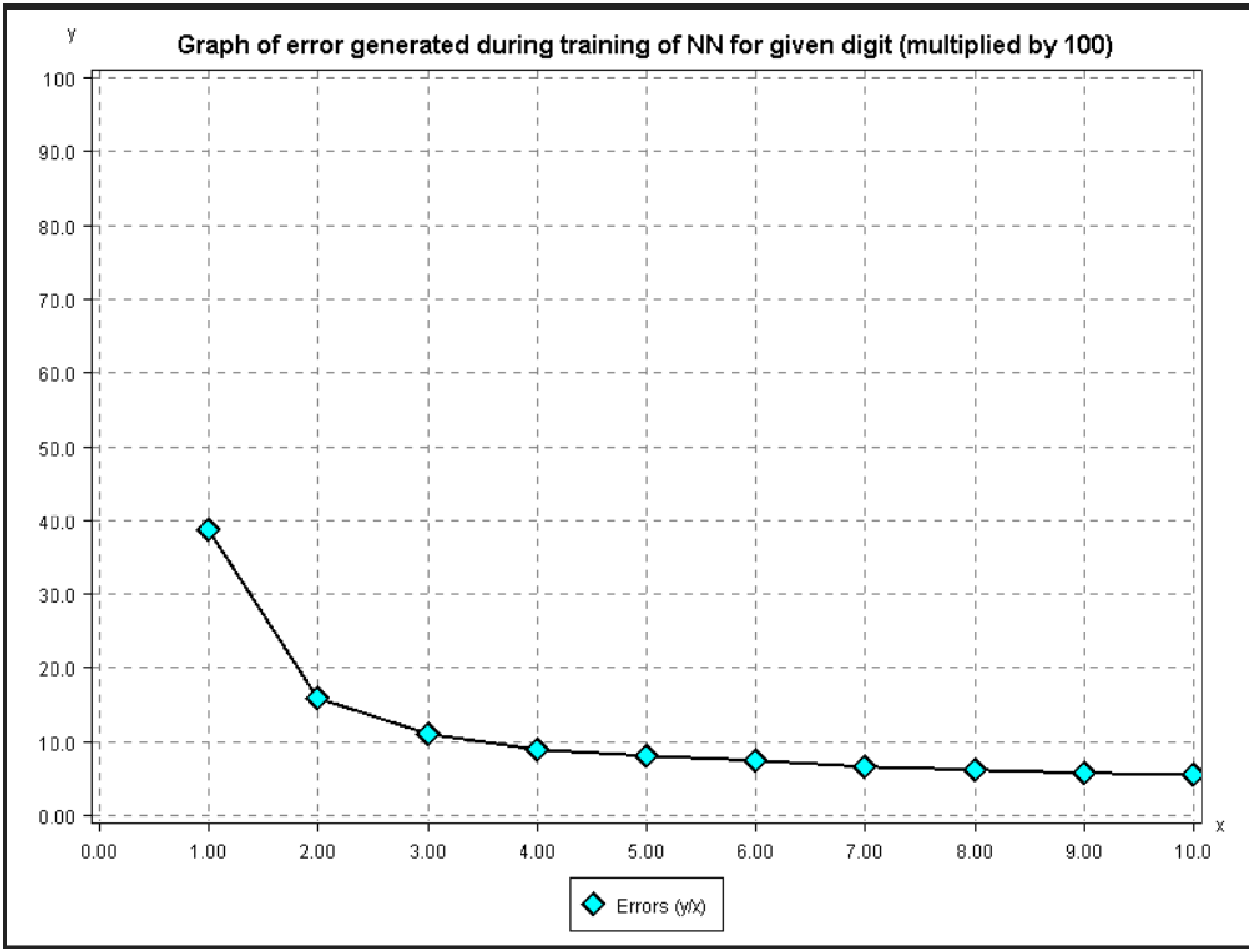
```
Error list for input digit = [0.385963242858573, 0.15921141418542067, 0.11056183021156638, 0.08859982920362813, 0.0794474040010323, 0.07504510084621374

Graph saved !

Neural network has been trained now !
```

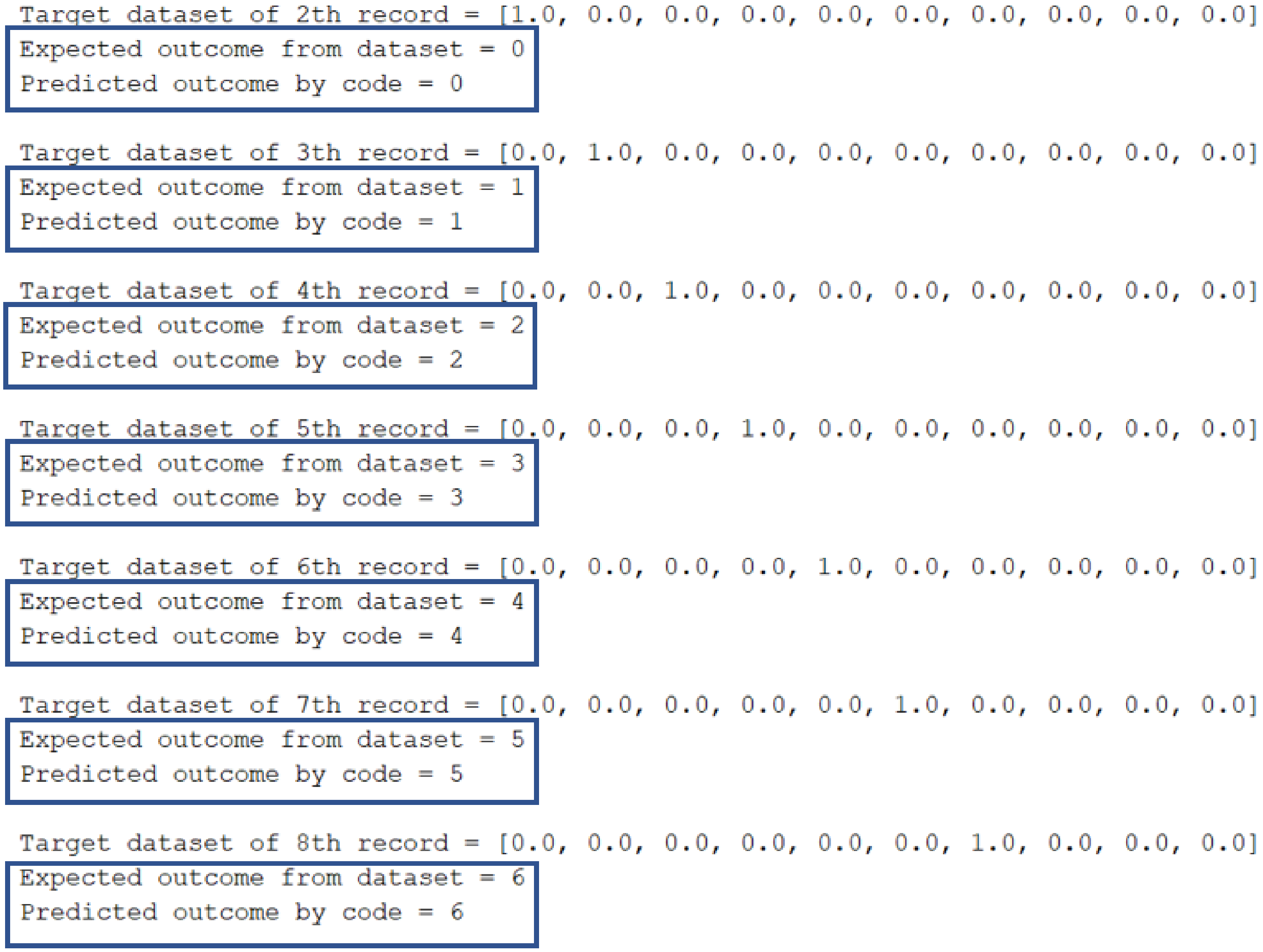


```
Enter an integer for which error graph needs to be drawn
6
```

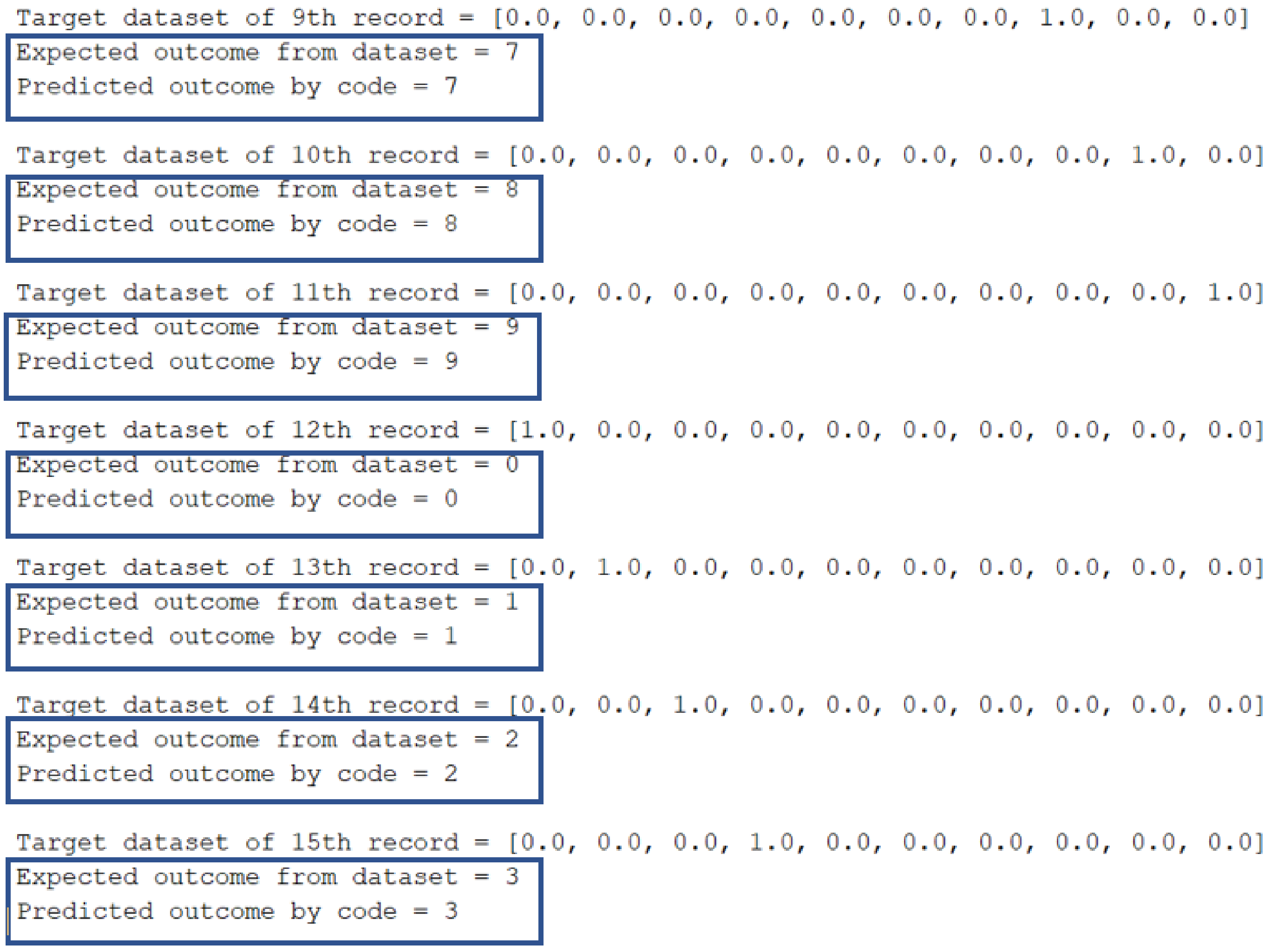
System will plot error graph of 10 iterations/error generated during prediction of each input. i.e. during each iteration, the difference between actual outcome and predicted outcome is plotted. The graph is saved in the directory of project.

For display purpose, we multiplied each error with 100 so that points can be distinguishable in the graph.

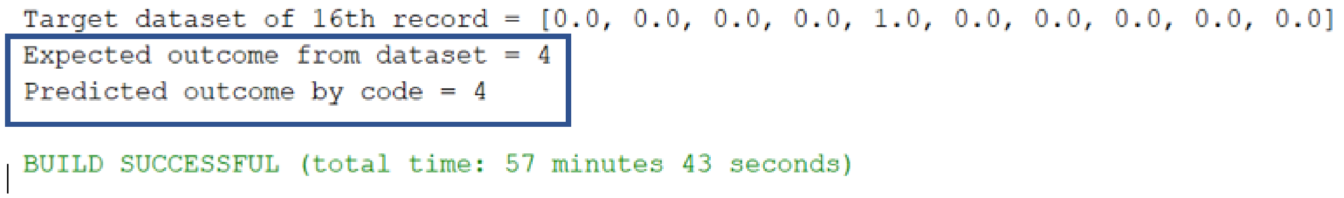




```
Target dataset of 2th record = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 0
Predicted outcome by code = 0

Target dataset of 3th record = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 1
Predicted outcome by code = 1

Target dataset of 4th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 2
Predicted outcome by code = 2

Target dataset of 5th record = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 3
Predicted outcome by code = 3

Target dataset of 6th record = [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 4
Predicted outcome by code = 4

Target dataset of 7th record = [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 5
Predicted outcome by code = 5

Target dataset of 8th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 6
Predicted outcome by code = 6
```



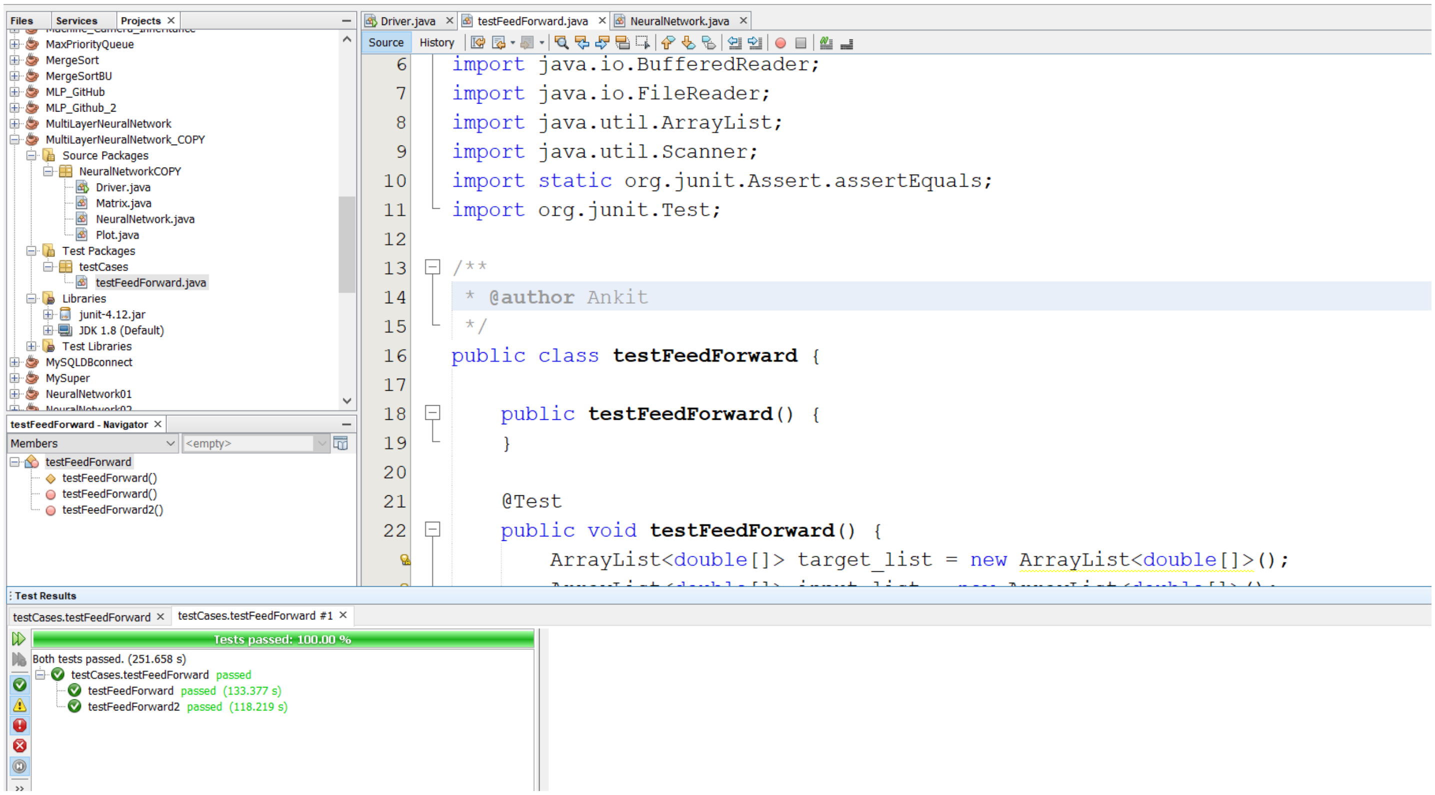
```
Target dataset of 9th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
Expected outcome from dataset = 7
Predicted outcome by code = 7

Target dataset of 10th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
Expected outcome from dataset = 8
Predicted outcome by code = 8

Target dataset of 11th record = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
Expected outcome from dataset = 9
Predicted outcome by code = 9

Target dataset of 12th record = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 0
Predicted outcome by code = 0

Target dataset of 13th record = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 1
Predicted outcome by code = 1

Target dataset of 14th record = [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 2
Predicted outcome by code = 2

Target dataset of 15th record = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 3
Predicted outcome by code = 3
```



```
Target dataset of 16th record = [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Expected outcome from dataset = 4
Predicted outcome by code = 4

BUILD SUCCESSFUL (total time: 57 minutes 43 seconds)
```

Accuracy for the above dataset - 100%

Below is the snapsot of testcase result.



```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.Scanner;
import static org.junit.Assert.assertEquals;
import org.junit.Test;

/**
 * @author Ankit
 */
public class testFeedForward {

    public testFeedForward() {
    }

    @Test
    public void testFeedForward() {
        ArrayList<double[]> target_list = new ArrayList<double[]>();
```

```
Tests passed: 100.00 %
Both tests passed. (251.658 s)
  testCases.testFeedForward passed
    testFeedForward passed (133.377 s)
    testFeedForward2 passed (118.219 s)
```