Classifying MNIST Digits

Raju Singh
001286876

Vikram Ramesh
001856230

Abstract:

In this project we present an Artificial Neural Network(ANN) to recognize human handwritten digits. For the computer an image is just a collection of pixels with different colours. Whatever is actually in the picture is very hard for a computer to identify. Yet, state-of-the-art neural networks are already been able to automatically identify faces or describe the actual content of a photo. The ANN proposed here is experimented on the well-known MNIST data set to identify hand-written digits.

Introduction:

In this project we mainly focus on identifying digits 0-9 from segmented pictures of handwritten digits. The input of the program is a grey level image, the intensity level of which varies from 0 to 255 (rgb values). For simplicity, input images are pre-treated to be of certain fixed size, and each input image should contain only one unknown digit in the middle. The MNIST data set is the most popular dataset and is a standard data set for testing various algorithms. The output of the program will be the corresponding 0-9 digit contained in input image.

Dataset:

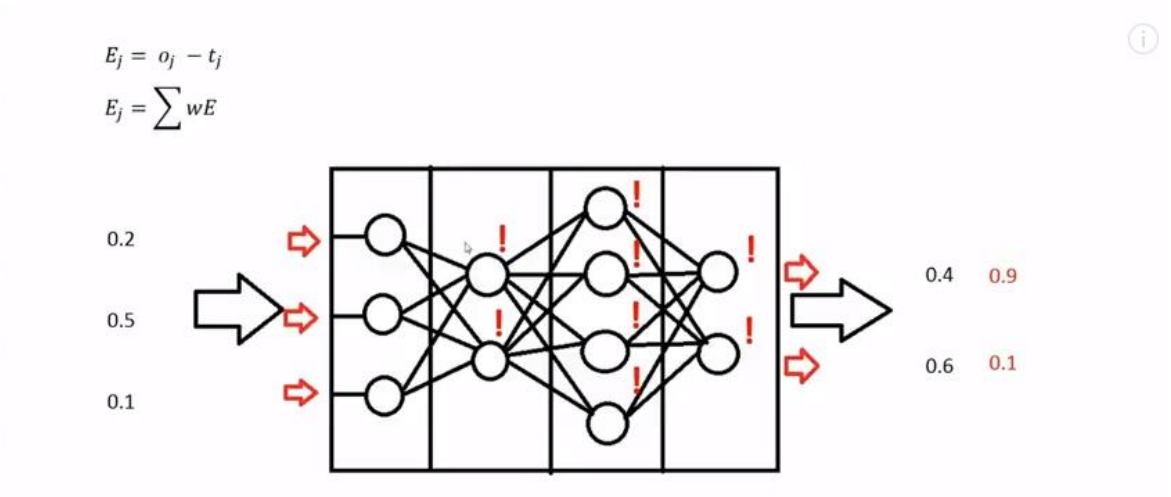
The MNIST problem is a dataset developed for evaluating machine learning models on the handwritten digit classification problem. The dataset was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST dataset.

Images of digits were taken from a variety of scanned documents, normalized in size and cantered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on the machine learning with very little data cleaning or preparation required.

Each image is a 28 x 28 pixels square (784 pixels total). A standard split of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it. It is a digit recognition task. As such there are 10 digits (0 to 9) or 10 classes to predict. Results are reported using prediction error, which is nothing more than the inverted classification accuracy.

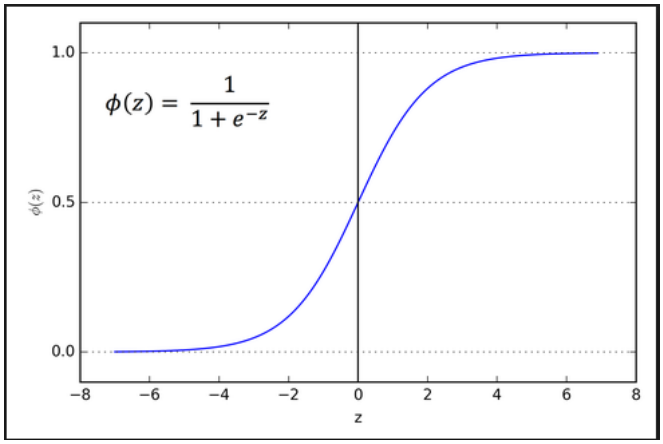
Methods:

- **Backpropagation**



This class implements a backpropagation training algorithm for feed forward neural networks. It works by analysing the error of the output of the neural network. In our program, we calculate the error in the output nodes by calculating the difference between the expected value and the actual value during the model training. For layers other than the input layers, the weight is adjusted by calculating the sum of the product of error and the weight of the node. This process continues working its way backwards through the layers of the neural network. The learning rate specifies the degree to which the weight matrixes will be modified through each iteration.

- **Sigmoid Function**



We used Sigmoid function as an activation function where the "z" value is a sum of the weights and the bias of each node. Its output is always bound between 0 and 1.

- **Neural Structure:**

A multilayer perceptron algorithm consists of several layers of nodes, interconnected through weighted acyclic arcs from each preceding layer to the following, without lateral or feedback connections. Each node calculates a transformed weighted linear combination of its inputs (output activations from the preceding layer), with one of the weights acting as a trainable bias connected to a constant input.

In our program, we have a total of 4 layers:

- Input Layer: The input layer is a 2-D Array and it consists of 28 x 28 nodes. Each node has a set of input weights, which is obtained from the pixel values through the csv file. The error values are not calculated for the input layer since all the rows are linked to a label in the label file.
- Two Hidden Layers: We have 2 hidden layers in between the input and output layers. First layer has 70 nodes and the second has 35 nodes.
- Output Layer: The output layer consists of 10 nodes, one each for the digits from 0-9. Based on the weights from previous nodes, weight of each node is calculated. The index value of the node, which has the highest weight, is the output of the image.

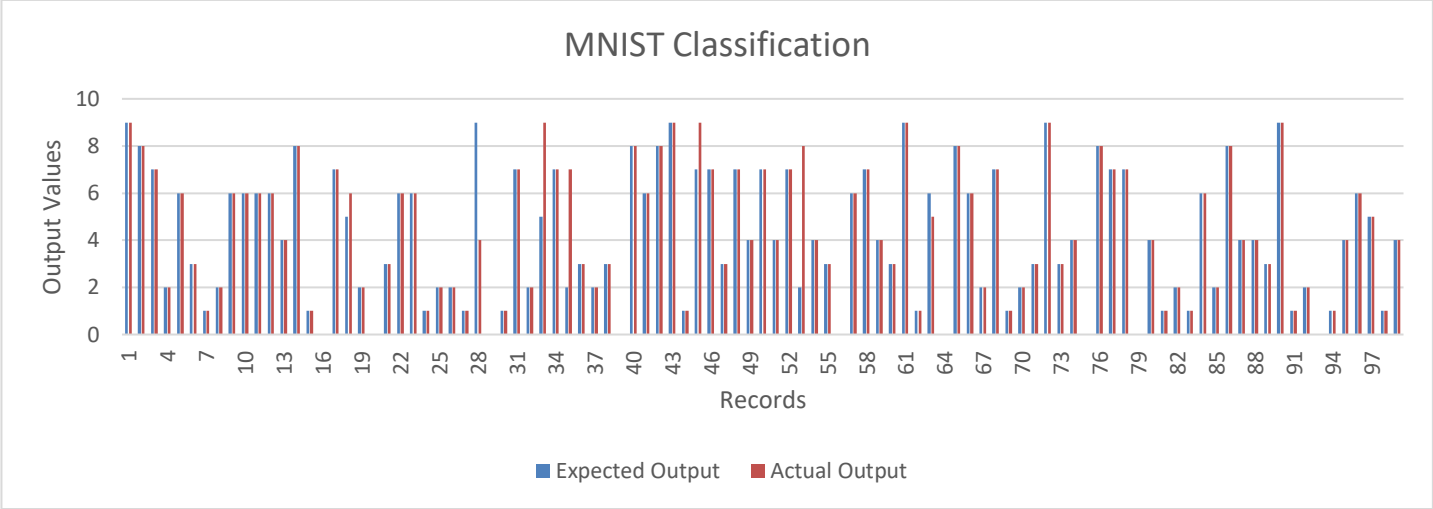
- **Neurons**:
  This class consists of the neurons or nodes in each layer. Initially each neuron consists of a random weight and a random bias, between -0.5 and 0.7. This class implemented the backpropagation method, MeanSquareError method and method to calculate the index of out the output of the highest value.

- **Training and test Methods**
  For training the model we read the data from the CSV files, containing 5000 records using BufferReader to read each record. The first column is parse as label and the rest of the columns is the pixel intensity. We feed a part of the training dataset to train our model. The training occurs using the backpropagation and activation functions.

  To test the trained model, we utilize the unused 5000 records from the dataset and test the accuracy of the model. The accuracy is calculated by checking the ratio of the actual output by expected output.

Observation:

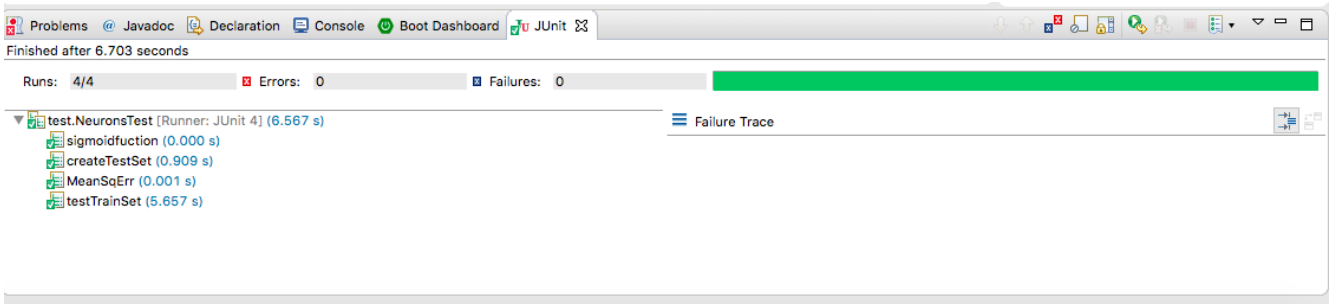
Taken subset of result for plotting purpose:



In the above graph we have the digit values between 0-9 on the Y-axis and the image record on the X-axis. If both the bars are of same height, then output number matches the image label, in case they are of different heights, then we have an expected output- an error. As seen from the graph, most of the bars are of the same height and this shows that, the accuracy of the model is very good.

Unit Test Cases:

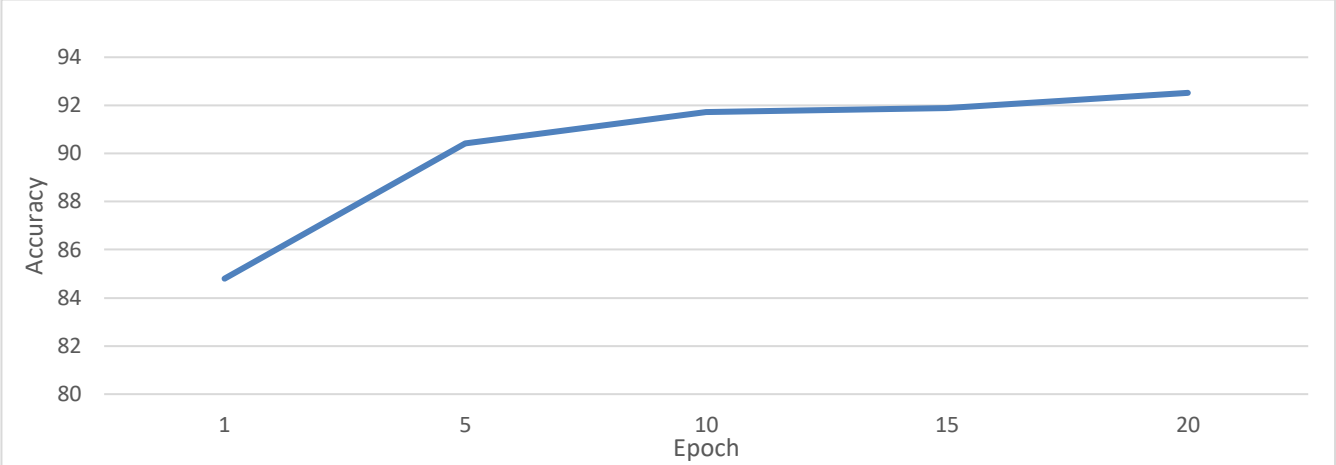
1. SigmoidFunction: Function to test the activation function.
2. CreateTestSet: Checks if the input value is transformed into double value for further calculation
3. MeanSqrError: Calculates the error during back propagation
4. testTrainSet: Checks the expected value against the actual value



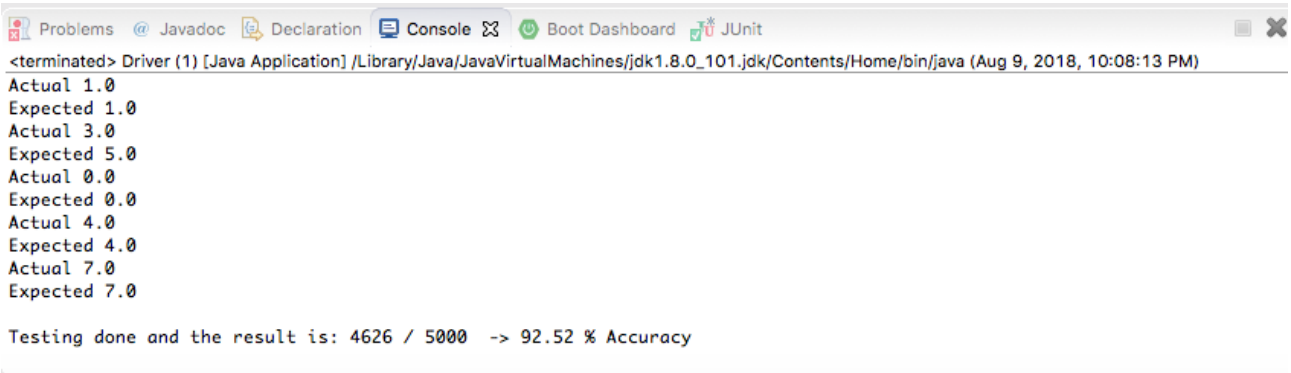
Working:

1. Initially we take in 5000 records from the labelled dataset and we train the model. The neurons contain random weights and bias values and the input layer takes the data from the dataset.
2. The hidden layers are then trained through a series of functions that include backpropagation and activation functions. These functions help in training the model to resemble the labelled values as much as possible.
3. This backpropagation keeps on going till the model is close to the labelled value, at each node there is activation function which output only if the weight and bias sum increases the threshold value. Finally, the output layer gives the highest weighted index of the array[0-9].
4. The training of the model depends on the epoch and learning rate. These play a major role in determining the accuracy of the model.
5. After training our model with 5000 records, and testing them against 5000 records, we get the 4595 as the correct output which gives our model an accuracy of 92%.
6. As we increase the epochs and get out model more trained we get more and more accuracy which goes till 97%.

Epoch	Accuracy
1	84.8
5	90.42
10	91.72
15	91.89
20	92.52



From the above graphs, we can determine that, as the number of epochs increases, the accuracy of the model increases. Below is a screenshot of the accuracy obtained for 20 epochs.



The output file from the program gives the ExpectedVsActual.csv file which has two columns- one for the Expected Output and another for the Actual calculated output. These two columns can be compared to check the accuracy of whole data set i.e. which one is a correct and which is an incorrect label.

Conclusion:

After training our model with 5000 records, and testing them against another 5000 records, we can successfully conclude that our model has given an accuracy of 92.52% for 20 epochs.

References:

- Neural Network Tutorial – By Finn Eggers
  https://www.youtube.com/channel/UCaKAU8vQzS-_e5xt7NSK3Xw
- Classifying MNIST Digits
  http://theanets.readthedocs.io/en/stable/examples/mnist-classifier.html