Project-2

CIS 666 Artificial Intelligence

Single Layer Perceptron Implementation in python

Fathima Syeda

Student ID: 2790024

Abstract

This project implements a Single Layer Perceptron in recognising Handwritten Digit Images into Digits from (0-9). The Handwritten digit images are taken from the MNIST dataset . Two sets of image datasets are taken, the smaller one is reserved for testing and is used only after the model is built using the other dataset. The single layer perceptron model is built by taking a subset of the large training dataset. Then the Error is calculated by comparing the output with the corresponding labels data. The models are built by not thresholding the input images once and by thresholding them the rest of the times. The model is built multiple times with 3 different values of learning rates on two sets of training data . The percentage error in recognising each digit is found for all three models constructed with three 3 different values of learning rates and it is plotted on bar charts. During the training process the Step function is used for activation and then later for one of the instances it also uses the Sigmoid function .

Contents

1. Introduction:

The Perceptron is one of the simplest Artificial Neural Network architectures and is based on a slightly different artificial neuron called a threshold logic unit (TLU), or sometimes a linear threshold unit (LTU). The inputs and output are numbers and each input connection is associated with a weight. A weighted sum of its inputs ($z = w1\ x1 + w2\ x2 + \cdots + wn\ xn = x^T w$),is computed then applies a step function to that sum and outputs the result: $hw(x) = step(z)$, where $z = x^T w$.

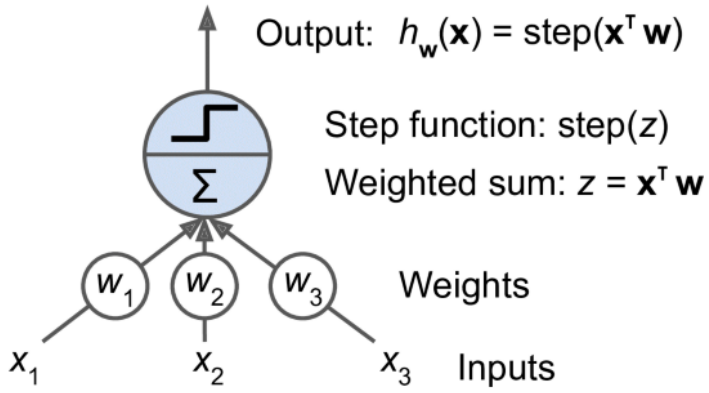


Figure 10-4. Threshold logic unit: an artificial neuron which computes a weighted sum of its inputs then applies a step function

The most common step function used in Perceptrons is the Heaviside step function. Sometimes the sign function is used instead.

Equation 10-1. Common step functions used in in Perceptrons (assuming threshold = 0)

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

It computes a linear combination of the inputs, and if the result exceeds a threshold, it outputs the positive class else it outputs the negative class. A Perceptron is simply composed of a single layer connected to all the inputs. When all the neurons in a layer are connected to every neuron in the previous layer (i.e., its input neurons), the layer is called a fully connected layer, or a dense layer. The inputs of the Perceptron are fed to special passthrough neurons called input neurons: they output whatever input they are fed. All the input neurons form the input layer. Apart from that an extra bias feature w0=1 is added . A Perceptron is a multioutput classifier as it can classify instances into many different binary classes.

Perceptron learning helps reduce the error by updating the weights. The Perceptron is fed one training instance at a time, and for each instance it makes its predictions. For every output neuron that produced a wrong prediction, it updates its corresponding weight. The following formula shows how to calculate the new weight :

Equation 10-3. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

- wi, j is the connection weight between the ith input neuron and the jth output neuron.
- xi is the ith input value of the current training instance.
- $y^n j$ is the output of the jth output neuron for the current training instance.
- yj is the target output of the jth output neuron for the current training instance.

Beta is the learning rate. The decision boundary of each output neuron is linear, so Perceptron are incapable of learning complex patterns However, if the training instances are linearly separable, this would converge to a solution and it is called the Perceptron convergence theorem.

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (p. 284- p. 288). O'Reilly Media. Kindle Edition

2. Steps involved in SLP implementation

2.1 Training Steps

The following steps are required in the implementation of a single layer perceptron .

1. Normalize the datasets. The input data is a 28x28 pixel image that is it contains 784 pixels and each point must be converted into either Binary data i.e. x_i =[0, 1] or Bipolar data i.e. x_i =[-1, 1].
2. Initialize the weights from a value between [0, 1]. The weight at first are initialized as any random values between 0,1 , They are represented as $\mathbf{W}_{ji}$, where j is the number of output nodes and i represents the input nodes.
3. Apply the input x^s from the training dataset. The input data set has 784 nodes and s patterns where s is the number of input images.
4. Compute the $V_j = \sum \mathbf{W}_{ji} * \mathbf{X}_i$. This is the summation of the product of the weights and the input nodes.
5. Find the system output Y_j using either step function or Sigmoid function.
   The Step function is given as $f(Vj) = yj = 1$ if $V_j >= 0$ or $yj = f(Vj) = 0$ if $V_j < 0$
   The sigmoid function is $Y_j = f(Vj) = \frac{1}{1+e^{-Vj}}$
6. Calculate the error $e_j = D_j - Y_j$, where D_j is the desired output or the classification label and Y_j, is the output from the training model.
7. If the error e_j is greater than Epsilon value which is almost equal to zero then modify the weights as , $w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$ where β is the learning curve value. The smaller the β value the slower the model learns and the higher the accuracy.
8. Calculate the Mean Square Error as $MSE = \frac{1}{2} \sum [D_j - Y_j]^2$
9. Repeat steps 4 to 8 until the model has error that is almost close to zero or less than the Epsilon value.

2.2 Testing Steps:

1. Get the testing Data X_test and its corresponding labels Test_Labels from the testing dataset.
2. Using the weights obtained at the end of training , Test the data by doing steps 4-6 from the training data set.
3. Calculate the error in classifying and plot the corresponding chart for it.

3 Program Outline

The project consists of three main parts, viz reading the input values and their labels along with modelling the Single layer perceptron, finding the mean Square error and plotting it as a curve and finally plotting a bar chart for different values of learning curve .

1. Design a fully connected network structure of 784 input nodes and 10 output nodes.
   - This is done by reading the training and testing files from the idx3-ubyte files and then calling *getData(Data, Labels, number*) to get 500 training and 100 testing data.
   - The labels values are replaced with an array[1x10] containing zeros and one 1 in the position of the value which is done by this method- *createLabelArray(Labels)*
   - The input X values must be thresholded and normalized for which this is called normalization*(XArray)*

2. Plot a learning curve that illustrates the mean square error versus iterations. (One iteration: apply all the training inputs once to the network and compute the mean square error).
   - The data read and prepared in the above section is then sent for training to build a model, by calling the *SLP_Training(X ,D, Beta=0.00025,sigmoid=False)* function. This returns the Weights W and an array of MSE values calculated at the end of each iteration which are then plotted .

3. Plot the percentage error in testing your handwritten digit recognition system as a bar chart. (Mean error occurred while testing each digit with the test data).

   To complete this section of the project, the testing(X, D, W, Beta) and the draw_barCharts(x_train, D, x_test, TestD, sig=False) functions are called extensively.

   - Task #1: Repeat this experiment for different learning rate parameters (at least 3 experiments. Start with a large value and gradually decrease to a small value).
     This task is done by giving the inputs read from section1 and the weights obtained from train the model in section 2 along with the testing data to the *draw_barCharts(x_train, D, x_test, TestD, sig=False)* function which then calls the training and testing functions 3 times for different values of Beta=0.5,0.05,0.005. It then calculates the percentage error in identifying each digit and plots this on bar charts.
   - Task #2: Repeat Task #1 with a large database. (5000 or more for training and 500 or more for testing).
   - Task #3: Repeat Task #2 with multilevel data (without thresholding the input data, normalize the input data, use sigmoid function for output thresholding)
     This is done by doing section 1 of the project but not thresholding the input data and using the sigmoid function to threshold the output data.

4 Program Implementation

The program consists of 6 functions to implement the single layer perceptron algorithm to correctly identify handwritten images into 10 digits from 0-9. The following are the functions:

- getData(Data, Labels, number)
- createLabelArray(Labels)
- normalization(XArray)
- SLP_Training(X ,D, Beta=0.00025,sigmoid=False)
- testing(X, D, W, Beta)
- draw_barCharts(x_train, D, x_test, TestD, sig=False)

4.1 getData(Data, Labels, number)

This function takes in the set of input values and their corresponding labels along with the number of elements to be read and returns a subset if the data and its corresponding labels according to the given number.

4.2 createLabelArray(Labels)

This function is called to convert the value sin the labels dataset from decimals to array of size 10 filled with zeros and 1 in the position of the value of the label. It takes in the set labels returns an array of zeros and 1 .

4.3 normalization(XArray)

This function is called to convert the values in the input dataset into 1s and 0s binary values

4.4 SLP_Training(X ,D, Beta=0.00025,sigmoid=False)

This function is called to train the single layer perceptron model. It takes in the set of training data X and the corresponding labels D along with optional arguments Beta(the learning rate) and sigmoid (represents which activation function to use.) At first the value of V is calculated as the matrix multiplication of $\mathbf{V_j = \sum W_{ji} * X_i}$. This is done using numpy library's **numpy.matmul()** function. Then the Step function is used for activation by default unless mentioned in the function's parameter to use the sigmoid function, **Y= np.where(V >=0 ,1 , 0)**. After which the Mean Square Error is calculated and all the steps ,mentioned in the training steps are done. It returns the weights and the MSE in each iteration which is then plotted against a curve using **matplotlib.pyplot.plot()**

4.5 testing(X, D, W, Beta)

This function takes the testing data and the corresponding labels, the weights from the trained model and following the steps in the testing steps mentioned earlier classifies the data. It also calculates the error and returns it .

4.6 draw_barCharts(x_train, D, x_test, TestD, sig=False)

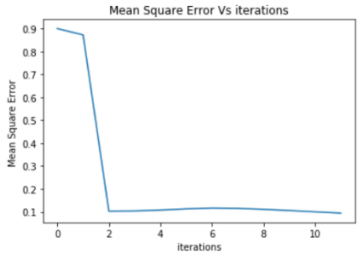
This function is given the training data and the corresponding labels, the testing data and the corresponding labels and a flag telling if you should use the sigmoid activation function. It takes the training data and trains three different models with 3 different learning curves **Beta=0.5, 0.05, 0.005** by calling **the SLP_Training(X ,D, Beta=0.00025,sigmoid=False)** function. Then it takes the weights obtained from those training models and tests them by calling the **testing(X, D, W, Beta**) function. Then it calculates the percentage error in

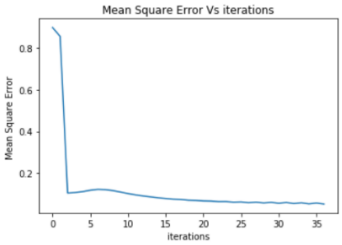
identifying each digit (0-9) and plots three bar charts for each learning curve using the **matplotlib.pyplot.bar()** function**.**

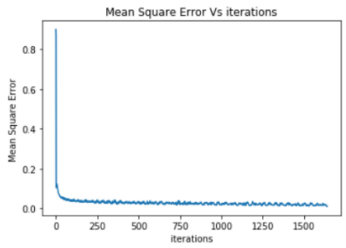
5 Testing

1. <u>Various values of epsilon</u>

   The algorithm was tested on the 5000-dataset and trained using three different values of Epsilon=0.01, 0.05, 0.1. Each time the epsilon value was increased and the model was trained and it was noticed that as the value of the Epsilon decreased the error decreased with increase in iterations and the curve got smoother



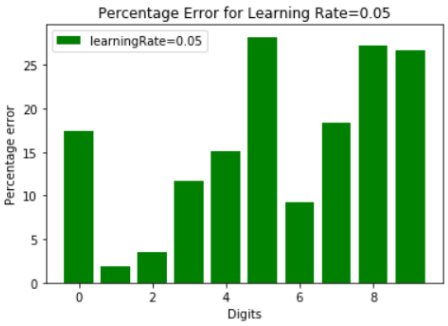




2. <u>Different datasets.</u>

The SLL algorithm was tested against two main datasets viz Images from 0-500 and 400-900.

The following is the output obtained when tested against the data from images 400-900 images.



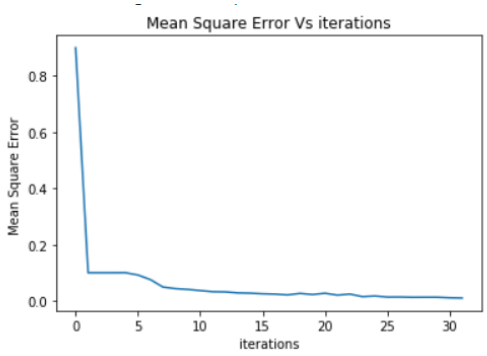
We notice that in this dataset we observe from the above charts that the digit with the maximum error in recognition is digit 8 with 40% error and the digits with minimum error are 0,4, with zero and 10% error in recognition.

We also notice that the error is high for Beta=0.5 with zero error for recognising digit 0 and less error for digit 4 and for Beta=0.05 it doesn't make much of a difference and then it

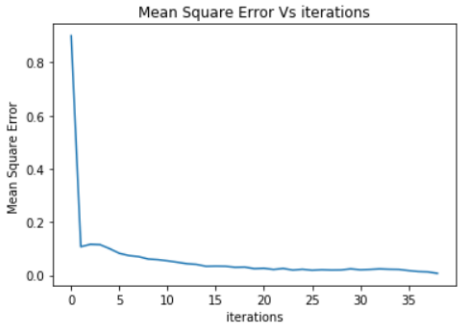
decreases for Beta=0.005 but we also notice that the error for zero increased to 12% and the error in recognising digit 4 became zero.

Max Error	Min Error	Beta value	Digit with Max	Digit with Min
30	3	0.5	8	1
29	3	0.05	5	1
35	2	0.005	8	1

3. MSE with different values of Beta=0.5, 0.05 was tested on the 5000-Training dataset.



Beta=0.5



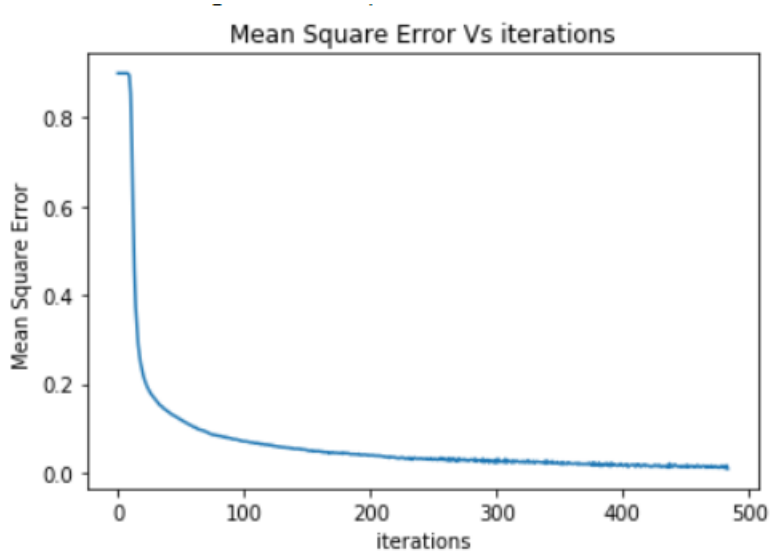
Beta=0.05

- It was observed that as the learning rate Beta decreased , the accuracy in recognising the digit increased i.e. the error decreased and the MSE curve got smoother.

6 Obtained Results

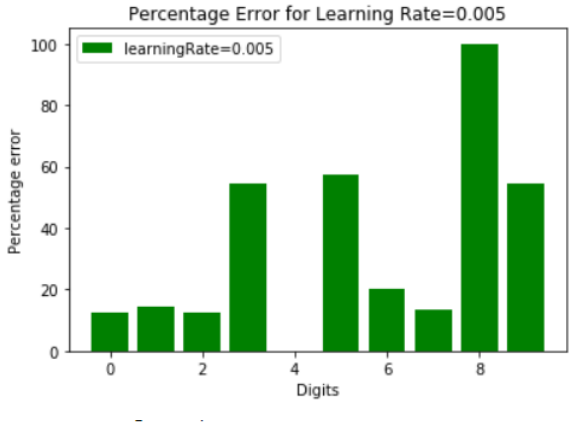
During the construction of the model by training it on a training set of 500 images we calculate the Mean Square Error(MSE) at the end of each iteration. These MSE computed are plotted against the number of iterations at the end of the training process. The below curve was obtained with an Epsilon value=0.001 and the learning curve Beta=0.00025

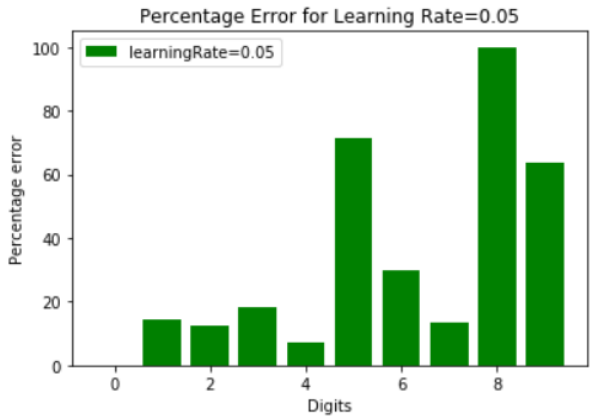
The curve was also plotted for various values of learning rate and it was observed that as the learning rate decreased the training process was slower and the result was more accurate. As the number of iteration increased the mean square error reduced along the curve and after 350 iterations the change became stagnant.

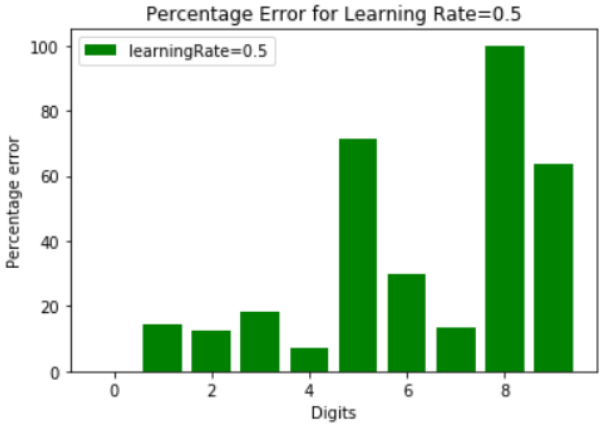


Task 1:

Below are three bar charts that were plotted for various values of the learning rate Beta=0.5,0.05,0.005. The training data used was of 5000 handwritten images and the testing data=1000 images. Epsilon=0.01, is used to check the error. Each bar chart below shows the Percentage Error in recognising each handwritten digit.



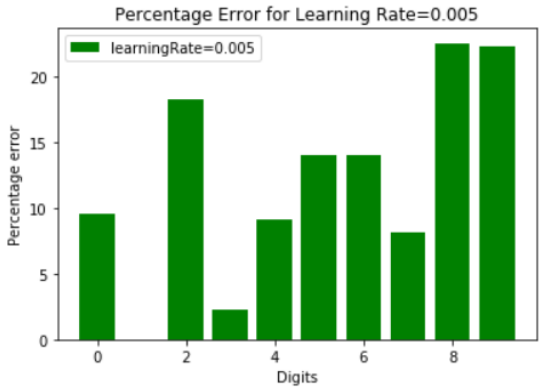


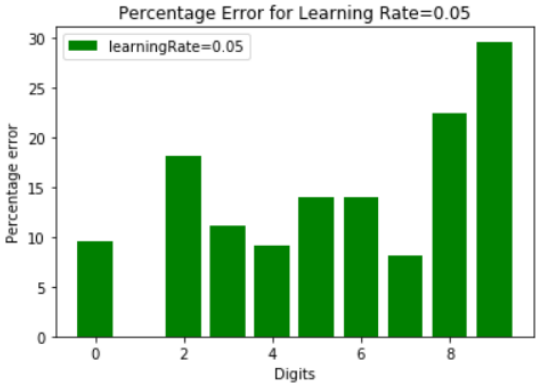


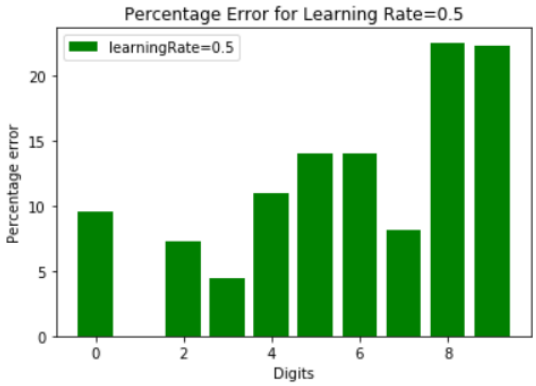
After observing the above charts we get that the digit with the maximum error in recognition is digit 8 with 100% error and the digits with minimum error are 0,4, with zero and 10% error in recognition.
We also notice that the error is high for Beta=0.5 with zero error for recognising digit 0 and less error for digit 4 and for Beta=0.05 it doesn't make much of a difference and then it decreases for Beta=0.005 but we also notice that the error for zero increased to 12% and the error in recognising digit 4 became zero.

Max Error	Min Error	Beta value	Digit with Max	Digit with Min
25	0	0.5	8	0
22	0	0.05	8	0
25	2	0.005	8	4

Task 2:
Training data set-5000, testing datset-500,Epsilon=0.001
The training data size=5000, testing data=1000 images . The output data was normalized using the Step function. Since the data set is large it took more time to display the results. Took 15 secs (4 secs each for the first 2 bar charts)





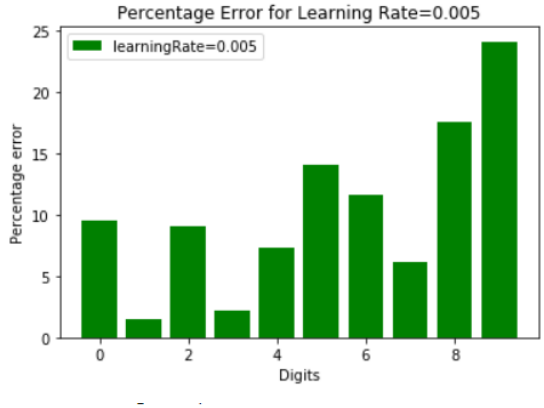


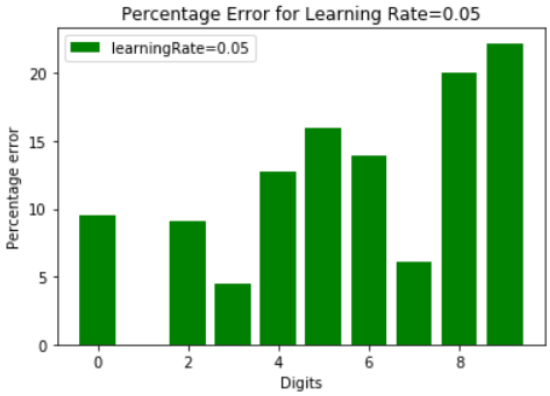
The digit with the maximum error in recognition is digit 8 followed by digit 9 with around 30% error the minimum error digits are 1, 3 with zero and 4% error in recognition.
We also notice that the error is high for Beta=0.5 then it increases for Beta=0.05and then decreases for Beta=0.005.

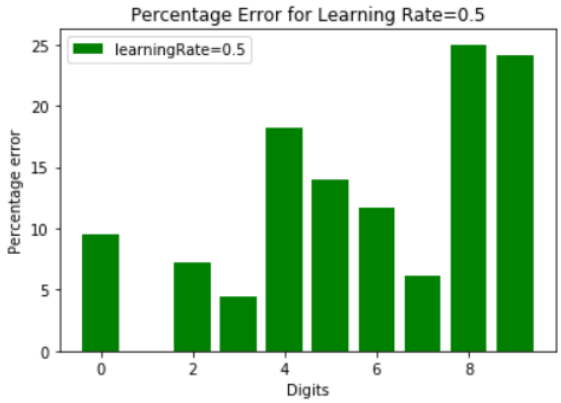
Max Error	Min Error	Beta value	Digit with Max	Digit with Min
22	0	0.5	8	1
30	0	0.05	9,8	1
22	0	0.005	8	1

Task 3:

The training data size=5000, testing data=1000 images . The input data wasn't thresholded before normalizing, and the output data was normalized using the Sigmoid function unlike in the above tasks. Since the data set is large it took more time to display the results. **Took 15 secs to load the data.** In order to reduce execution time, the Error comparison value was slightly increased to Epsilon=0.015.







The digit with the maximum error in recognition is digit 9 with 25 % error the minimum error digit is 1 with zero error in recognition.

It is noticed as in task-2 that the error is high for Beta=0.5 then it reduces for Beta=0.05and then slightly increases again for Beta=0.005.

Max Error	Min Error	Beta value
25	0	0.5
22	0	0.05
25	2	0.005

7 Conclusion

- The lower the learning rate Beta, the higher the accuracy in recognising the digit.
    - However, we noticed that in Task-3 the error increases again for Beta=0.005
    - In task-2 the error increases for Beta=0.05 and then decreases again for Beta=0.005
- The lower the value of epsilon the higher the accuracy rate, but the training model does a lot of iterations, up to 8000 for Epsilon=0.0001 for 5000 training dataset which can take a lot of time.
- The larger the training dataset the more accurate model is built. The model built in task-1 had error rates as high as 95 % but for the 5000-training dataset the max error was 30%
- The handwritten digit with the least amount error in recognition was Digit 1 followed by Digit 0 as they are very simple in shape.
- The handwritten digit with the greatest amount error in recognition was Digit 8 followed by Digit 9. This maybe due to the complicated shape of digits 8 and 9 with two circles or 1 and half circle.

8. References

- Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (p. 284- p. 288). O'Reilly Media. Kindle Edition
- Class Notes