**Report for Assignment 3:**

**-**First, I import the data as ‘MNIST’:

-For the input values (X), I drop the label column from MNIST, I convert it to a NumPy array and then I normalize by dividing X by 255.

-For the output values (label column), I select the first column of MNIST data (which is the label column) and then I convert it to NumPy Array. After this I reshape the array.

-I repeat the above steps for the test data also (X\_test, Y\_test).

-The **sigmoid** function is our activation function.

-The **sigmoid\_der** function is the derivative of the sigmoid function.

-The **predict** function will be used later to take in X\_test values and return predicted Y values. (It will use our updated weights and biases after backpropagation is done). It is basically a forward pass.

-The **accuracy** function is used to show the accuracy between two arrays of the same dimensions. We start with k = 0. If ‘i’ from the first array is equal to ‘j’ from the second array, we increment by one. At the end we return k divided by the length of the array times 100.

-**Epoch: 1000, Alpha(learning rate) = 0.001**

**-Input Layer Neurons = 784 (number of columns of X), Output Neurons** = 1

-**Hidden Layers**: 1

-**Overall Accuracy** is 99.85% (for best try).

**Main Algorithm:**

-I start by initializing the weights randomly between -1 and 1. Weight1 takes dimension (784,1), weight2 takes dimension (1,1).

-I initialize random bias values of dimension (1,1) each.

-Training the model (for one epoch):

-I perform the operations from class, beginning with the forward pass.

-I perform the back propagation, first finding the error between our expected Y values and the output from the forward pass- and then propagating the error back into the network.

-I then update the weights and biases accordingly.

-I repeat the training 1000 times.

-Now we use the updated weights and biases in the predict function on X\_test (which is from the test data) to try to predict Y\_test values.

-When we get these values, we round either up or down as needed so that we can calculate accuracy. So, for example, 0.994 would now be a ‘1’ prediction and 0.033 would now be a ‘0’ prediction.

-**Accuracy** is about 99.85%, as stated above.

**BONUS-----------------------------------------------------------------------------------------------:**

Mostly, the architecture remained the same as the original. A few changes made from the original code/algorithm were:

-For my training, I first normalized my Y output values, because I am still using the sigmoid activation function and I wanted them to be between 0 and 1. So I divided Y by 4.

-Then I run everything for 1000 epochs with 0.001 learning rate, just like the main part.

-However, for this part I use 4 hidden layers (along with 784 input neurons and 1 output neuron like the original).

-Once this is done and I have my weights and biases to use, I use the **predict** function on X\_test to get predicted values for the test data.

-Because I originally divided the Y training output data by four, I then multiply the outputs from the **prediction** by 4.

-These are now my predicted unrounded Outputs for my test data.

-To better measure I accuracy, I then round them (up or down as needed).

- I got about 95% **accuracy** with this method on my best try, which is substantially better than a random guess of about 20%.