My network consists of 784 input neurons, 10 hidden neurons and 10 output neurons.The code consists of several functions that works overall to build the multi-layer back propagation neural network. The functions are:

1. Initialize\_network

This function assigns the random initial weights to the hidden layer and output layer. In both cases, the weight is also assigned to the bias.

1. Forward\_propagate

This function calculates the output from the hidden layer and output layer. The matrix multiplication is used for fast computation. The output from each layer is sent to the sigmoid function to find the sigmoid approximation.

1. Sigmoid

Each output from hidden neurons and output neurons are sent to the function to calculate the sigmoid approximation.

1. Backward\_propagate\_error

This function calculates the error associated with the outputs from forward propagation. The errors are calculated for both output from output neurons and hidden neurons.

1. Deri\_output

This function calculates the difference in the expected output and predicated output in output unit. This function is called from backward\_propagate\_error function.

1. Update\_weights

This function updates the weights of network proportionately according to the calculated error.

We are using the Sequential method to train the network. Altogether 10 iterations are done, so there are 10 epochs. In each epoch 40000 rows of training data are scaled by 255 and passed to the network sequentially for training of the network. All the functions are called in correct order to obtain the final updated network weights.

1) Techniques to improve the efficiency of the code

- Vectorized multiplication - Cause nested looping for calculating various values doing operations is inefficient when we are dealing with a maasive set of data.

- Use of numpy nd array to store the data. Because matrix multiplication is very fast especially when we are dealing with ANN's hidden layers.

-Other approaches we have considered are scaling the input data, initializing weights with lower values.

- One other thing that we wanted to do was feature selection. The dataset had 784 features out of which some had no information in them. (White spaces). But leaving them as is didn't do much harm as the accuracy and training time taken to update the weights was considerable the same.

2) Our Multi-layered perceptron has three layers. The first one is the input layer which has 784 units, i.e., feature vectors as the input for the neural network initialization. the seond layer was hidden layer which had about 200 hidden neurons in it. This layer was a hit and trail kind of thing where we tried training the network with different neurons. And the third layer that is the output layer has about 10 units in it. Because the problem we are dealing with is a 10-class classification problem.

3) Our classifier after training on our training data gave --% accuracy and when tested on the testing data gave --% accuracy .

4) Depending te size of the training data that we considred and the number of iterations we made it run, our model takes anywhere in between 1 to 1.5 hours to learn the input data.

5) The methods we chose to train the data were batch and sequential. No particular reason why we chose these. We felt like these two techniques are the extremes in model training. One method considers the whole data as a batch in one iteration whereas the other takes one datapoint at a time. We also wanted to try mini-batch but couldn't because of the time constraint. So we decided mini-batch would be for futher investigation.

## Describe the code

The code consists of several functions that collectively helps in building the neural network. Some of the functions are

- scaling(dF) - takes in a dataframe and normalizes it.

- initialize\_weights(args) - Used to Initialize the initial weights that initialize the neural network.

- sigmoid(args) - A sigmoid function that takes in a ndarray and applies the sigmoid function and returns the sigmoid approximation of that ndarray.

- randomize(df) - This is used to shuffle the data after every iteration. It takes in a dataframe and returns the shuffed dataframe.

- forward\_prop(args) - This is where the forward propogation for the neural network happens. It takes in the input data and the weights as inputs and gives us the final Value of the neural network as the output along with other values that are obtained as a part of the forward propagation.

- error\_function(args) - Calculates the errors associated with the outputs that were obtained by the forward propagation function. The outputs obtained are total error of the neural network, the error of the output unit of the neural network, and the error of the hidden\_layer of the neural network.

- backward\_func(args) - This function is used to update the weights . It updates the input\_weights, output\_weights and as well as those associated with the bias units.

- After training is done, it is finally time to test the model on the test data and predict the output labels. And then we calculate the accuracy associated with the model.

6) Analysis and discussion of the results obtained fromt the MLP.

Ans) Some of the interesting things that we have observed while doing the project were

- sequential technique took more time to train than the batch process.

- The weights had to be sufficiently low for the batch technique, else, it threw an overflow warning and all the output values were approximated to one.

- Also, scaling proved useful in case of both sequential and the batch techniques. We tried to train the model without scaling the data and encountered an overflow warning in case of the sequential tehinique. This was resolved after scaling the training data.

-