Note

Before starting this assignment it is mandatory to complete the first 2 assignments of neural network.

Note

Before starting this assignment please make sure that you know all the fundamentals of Object Oriented Programming(oops) / classes and object. This assignment needed oops for implementation.

If you are not familiar with Object Oriented Programming please visit below mentioned link and watch the video.

https://www.youtube.com/watch?v=JeznW_7DlB0

Note

Please make sure that you know/ understand how forward propagtaion and backward propagation works. If not please visit below mentioned links and watch the videos.

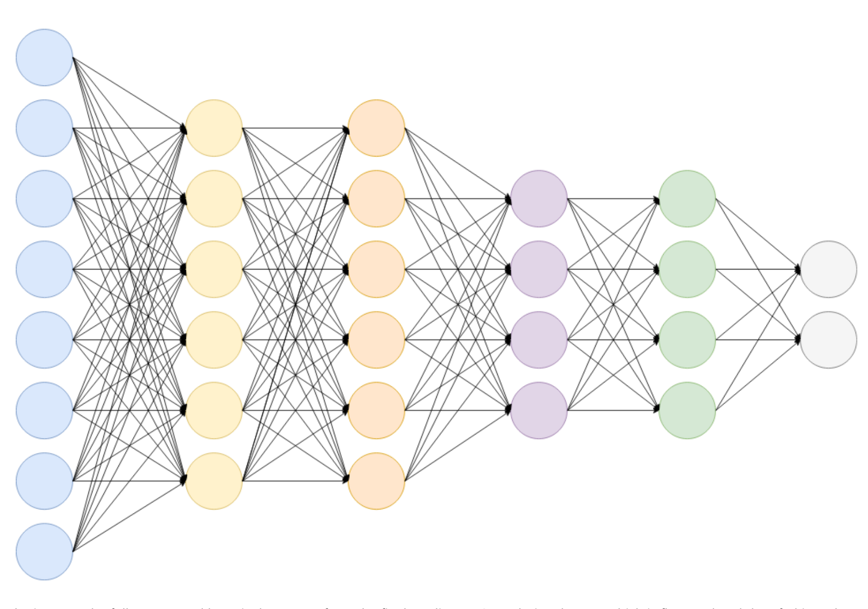
https://www.youtube.com/watch?v=qzPQ8cEsVK8

backward propagation:

https://www.youtube.com/watch?v=q555kfIFUCM

```
# import numpy library
```

Let's Build A FC (Fully Connected) Layer



The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

Each layer includes following values terms:

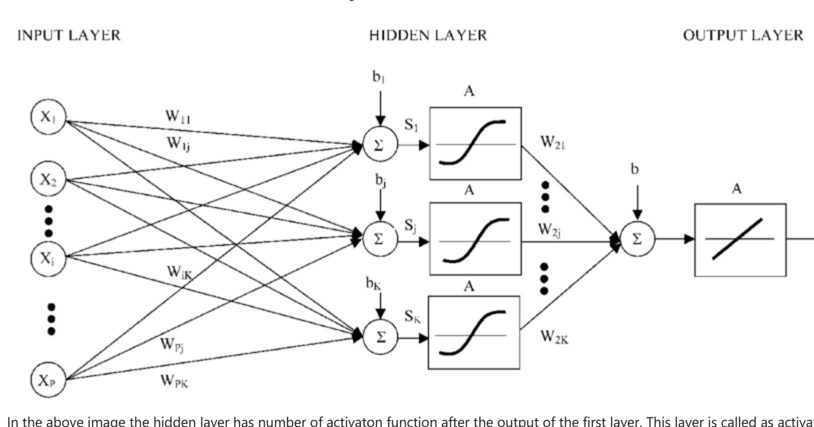
- input size
- output size
- weights
- bias

For this layer we will be having two functionality:

- Forward Propagation
- Bakward Propagation

```
# define a class named FCLayer

  '''
  define a _init_ function for ininitializing some values such as:
    input_size of the data
    outputsize of the data
    and
    generate some weights based on input and output size
    also generate Bias with range of 1 to output size
    '''
  # define a _init_ function with parameters: input_size and output_size

    # set the objects input size same as the input_size passed to the function

    # set the objects output size same as the output_size passed to the function

    # generate the objects weights list with numpy random values ranging from input_size to output_size

    # divide the weights with square root of sum of input size and output size

    # generate the objects bias list with range between 1 to output_size

    # divide the bias  with square root of sum of input size and output size

  # define a function named forward with 1 parameter named input

    # set the input of the current object to input received as parameter

    # return dot product of input and weights and add bias to the dot product

  # define a function named backward with 2 parameter named: output_error, learning_rate

    # create a variable named input_error and store dot product of outout_error and transpose of weights

    # create a variable named weights_error and store dot product of transpose of output_error

    # Set weights of the current object to difference of weights and product of learning_rate with weights_erro

    # Set bias of the current object to difference of bias and product of learning_rate with output_error. i.e

    # return input_error
```

lets build the ActivationLayer



In the above image the hidden layer has number of activaton function after the output of the first layer. This layer is called as activation layer.

This layer takes input data from previous layer + type of activation function

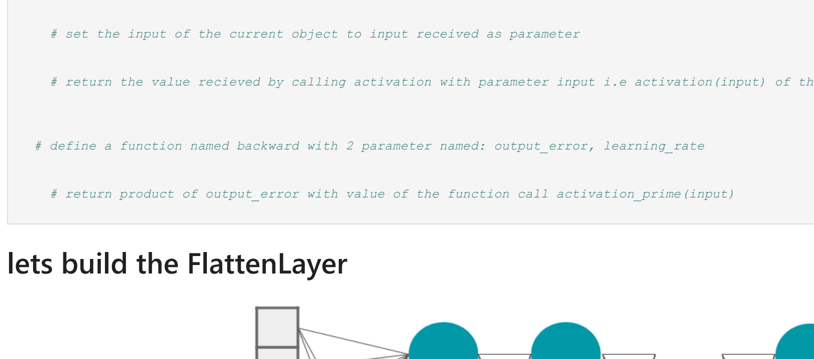
- input data
- activation function

This layer have two functionality:

- Forward Propagation
- Backward Propagation

```
# define a class named ActivationLayer

  # define a function name _init_ with 2 parameter named: activation, activation_prime

    # set the current objects activation to activation recevied as parameter

    # set the current objects activation_prime to activation_prime recevied as parameter

  # define a function named forward with 1 parameter named input

    # set the input of the current object to input received as parameter

    # return the value recieved by calling activation with parameter input i.e activation(input) of the current

  # define a function named backward with 2 parameter named: output_error, learning_rate

    # return product of output_error with value of the function call activation_prime(input)
```

lets build the FlattenLayer



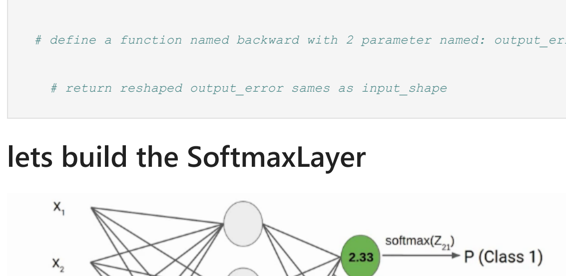
The data needs to be in the form of a 1-dimensional linear vector. Rectangular or cubic shapes can't be direct inputs. And this is why we need flattening.

Flattern layer has two functionality

- Forward Propagation(Where images are converted into 1 dimensional array)
- Backward propagation(Where flattend images are converetd in to original shaped images which were during input)

```
# define a class named FlattenLayer

  # define a function named _init_ with parameter named input_shape

    # set the objects input shape same as the input_shape passed to the function

  # define a function named forward with 1 parameter named input

    # return rehaped array with 1 dimension i.e single vector

  # define a function named backward with 2 parameter named: output_error, learning_rate

    # return reshaped output_error sames as input_shape
```

lets build the SoftmaxLayer



The last layer of the neural net called output layer includes functions named softmax for each output value

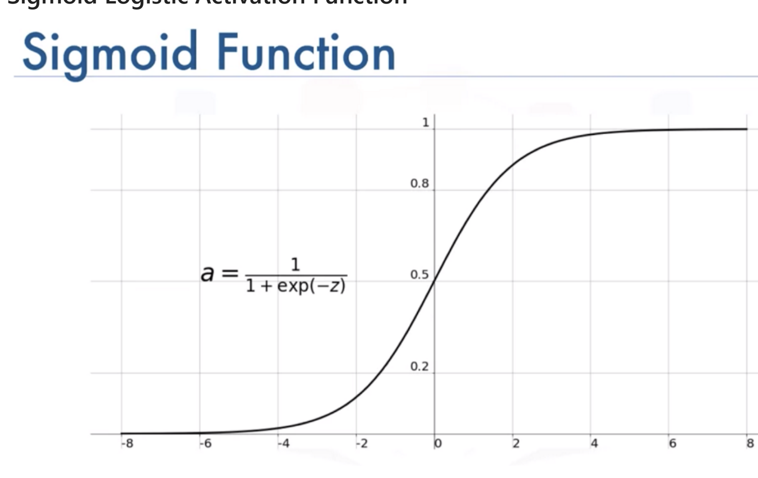
Formula for softmax function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

```
# define a class named SoftmaxLayer

  # define a _init_ function with parameters: input_size and output_size

    # set the objects input size same as the input_size passed to the function

  # define a function named forward with 1 parameter named input

    # set the input of the current object to input received as parameter

    # create a temporary varibale to store the value o fthe formula e^input

    # set the current objects output to the value obtaine by temporary/(sum of temp)

    # return current objects ouput value

  # define a function named backward with 2 parameter named: output_error, learning_rate

    # create a variabe to store generated zeros array of shape smae as output_error shape

    # create a varibale to store array of repeated values of transpose of output and shape same as input_size

    # create a vaibale to store generated Identity array of input_size

    # create a varibale to store dot product of output_error with (identity array - output genrated using np.t

    # return product of output with dot product we got in above line
```

lets define all of the activation function needed

Sigmoid Logistic Activation Function



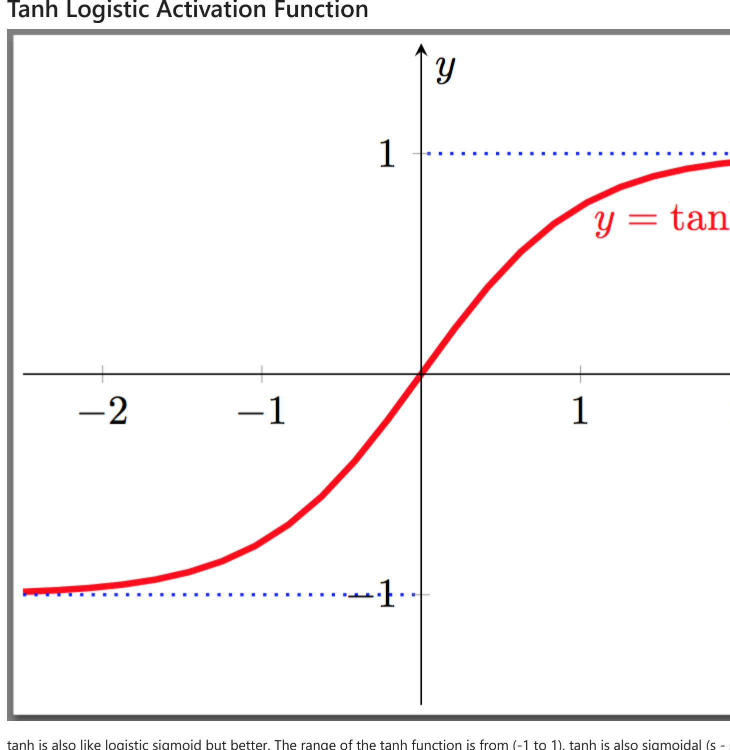
The Sigmoid Function curve looks like a S-shape.

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

```
# define a function named sigmoid with parameter x

  # return the calculated value by the formula 1/(1+e^(-x))
```

```
# define a function named sigmoid_prime with parameter x

  # return the calculated value by the formula e^(-x)/(1+e^(-x))^2
```

Tanh Logistic Activation Function



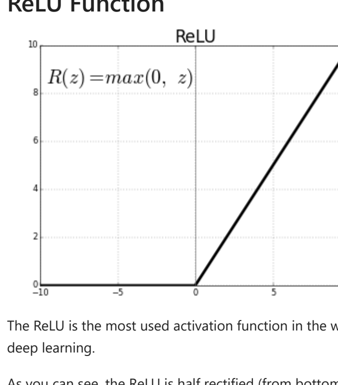
tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

```
# define a function named tanh with parameter x

  # return the calculated value by the formula tanh(x)
```

```
# define a function named tanh_prime with parameter x

  # return the calculated value by the formula 1-tanh(x)^2
```

ReLU Function



The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

Range: [0 to infinity]

```
# define a function named relu with parameter x

  # return teh maximum values in x grater than zero i.e All positive numbers
```

```
# define a function named relu_prime with parameter x

  # return the array with value 1 for elemts which are greater than of equal to 0 with astype = 'int'
```

Mean Squared Error Function

The Mean Squared Error (MSE) is perhaps the simplest and most common loss function, often taught in introductory Machine Learning courses. To calculate the MSE, you take the difference between your model's predictions and the ground truth, square it, and average it out across the whole dataset.

The MSE will never be negative, since we are always squaring the errors. The MSE is formally defined by the following equation:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

```
# define a function named mse with parameter y_true, y_pred

  # return mean of (y_true - y_pred)^2
```

```
# define a function named mse_prime with parameter y_true, y_pred

  # return the calculated value as per the formula: 2 x (y_pred - y_true)^2/y_pred.size
```

Lets Import MNIST Dataset

We will be importing and loading MNIST handwritten digit from tensorflow database.

```
# Import MNIST Dataset from keras

# import utils from keras

# Load MNIST Dataset into train and test set
```

```
# check the number of classes and names of classes
```

```
Number of classes/labels:  10
Names of classes/labels:  [0 1 2 3 4 5 6 7 8 9]
```

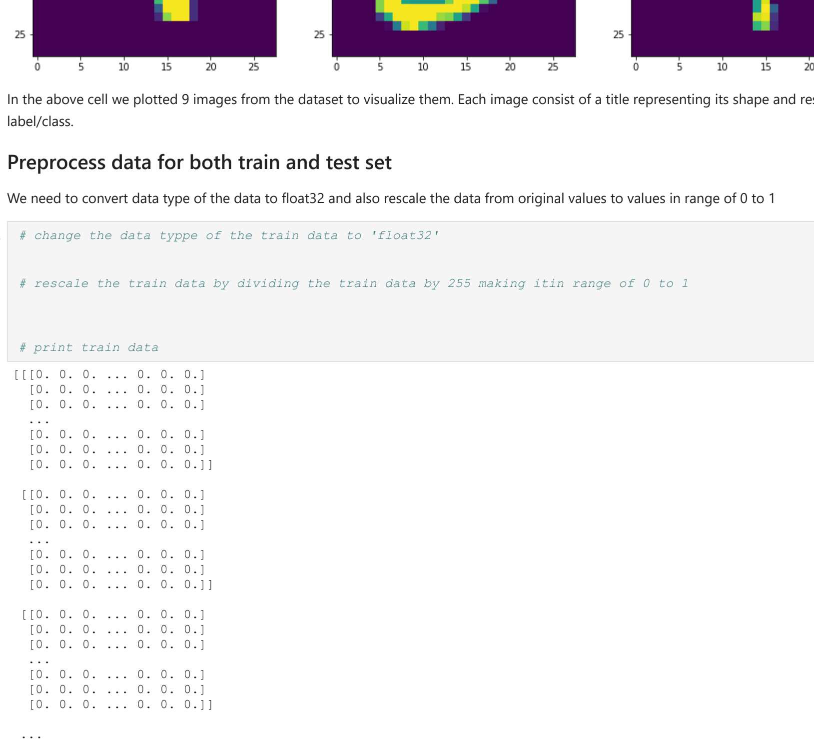
Ploting sample images

Ploting image data with their shape and true label as plot title

```
# define a function named show_samples for ploting the sample images (parameters: dataset, label)

  # create a variable for columns and rows number I.e 3 rows  x 3 columns for 9 plots

  # create a figure variable

  # print number of samples to to be plot

  # initalize a variabel i to value 1

  # simultaneously loop through dataset and label till 9 values using zip function

    # create a subplot with (3,3, position of the plot)

    #  plot the images using imshow and np.squeeze to sqeeze the array of image

    # Add title with values as image shape and label

    # increment the value of variable i by 1

  # display the plot using plt.show()
```

```
# pass train data to function we create to plot sample images
```

```
9  samples from the dataset
```



In the above cell we plotted 9 images from the dataset to visualize them. Each image consist of a title representing its shape and respective label/class.

Preprocess data for both train and test set

We need to convert data type of the data to float32 and also rescale the data from original values to values in range of 0 to 1

```
# change the data typpe of the train data to 'float32'

# rescale the train data by dividing the train data by 255 making itin range of 0 to 1

# print train data
```

```
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 ...

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]
```

One Hot Encode The Labels

The labels we have are in the form of integer values ranging between 0 to 9. we need to convert this values into binary values.

```
# perform one_hot_encoding on the trin labels

# print train labels
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]]
```

```
# selecting first 1000 sample from train data

# selecting first 1000 sample from train data

# print sizes of both train data and train labels
```

```
Size of the train data:  1000
Size of the train label:  1000
```

```
# change the data typpe of the test data to 'float32'

# rescale the train data by dividing the train data by 255 making itin range of 0 to 1

# perform one_hot_encoding on the trin labels
```

Creating a network of all layers created in above cells

Finally connecting all the layers we created above together and forming a neural network. With a specific epoch value and suitable learning rate.

The input size of teh network will be 28x28 because the image data we will be importing consist of images with size 28x28 pixels

The ouput layer consist of 10 classes

```
'''create a list named network
    with elements:
      FlattenLayer(input_shape),
      FCLayer(input_size, output_size),
      ActivationLayer(relu, relu_prime),
      FCLayer(input_size, output_size),
      SoftmaxLayer(input_size)  --> this is the outputlayer with 10 classes
'''

# add layers in the below network list as mentioned in above comment
network = [

]

# mention number of epochs to train the model

# mention learning rate to train the model
```

Training the model

Lets try training the model on the dataset and visualizing all the errors

```python
# loop through range of epochs mentioned in above cell

  # initialize error variable to 0

  # loop simulatneously through train data and train labels using zip function

      # forward propagation

      # create a variable output and store current interated train data sample

      # loop through network list create in above cell

        # set the output variable value to returned value of each network layers forward propagation value I.e


      # error (display purpose only)
      # increamet the error variable created above by value returned by mean square error function created in a



      # backward propagation
      # set the output_error variable created above by value returned by mean square error prime function creat


      # loop through reversed network list created in above cell

        # set the output_error variable value to returned value of each network layers backward propagation va


  # set the value of error variable to division: error / length of the tain data

  # print the current epoch, total epoch and error
```

```
1/40, error=0.071879
2/40, error=0.034677
3/40, error=0.022326
4/40, error=0.017151
5/40, error=0.013985
6/40, error=0.011749
7/40, error=0.009981
8/40, error=0.008539
9/40, error=0.007399
10/40, error=0.006484
11/40, error=0.005698
12/40, error=0.004995
13/40, error=0.004510
14/40, error=0.004101
15/40, error=0.003741
16/40, error=0.003431
17/40, error=0.003180
18/40, error=0.002973
19/40, error=0.002803
20/40, error=0.002654
21/40, error=0.002524
22/40, error=0.002413
23/40, error=0.002326
24/40, error=0.002251
25/40, error=0.002184
26/40, error=0.002123
27/40, error=0.002063
28/40, error=0.002008
29/40, error=0.001961
30/40, error=0.001911
31/40, error=0.001835
32/40, error=0.001779
33/40, error=0.001737
34/40, error=0.001659
35/40, error=0.001604
36/40, error=0.001582
37/40, error=0.001565
38/40, error=0.001549
39/40, error=0.001536
40/40, error=0.001523
```

The model error has been reduced from 0.07 to 0.001 approx

Let's predict the test data using the trained model

Alright we already trained the model on datastet, now we need to make prediction on test data

```python
# define a function named predict with parameter: network, input

  # create a variable named output and set its value as input recieved in parameter

  # loop through network list created in above cell

    # set the output variable value to returned value of each network layers forward propagation value i.e laye

  # return output value



# create an empty list named ratio

# create an empty list named error


# loop through test data and test label simulataneously using zip function


  # for each max value in true label and max value of predicted label if both are equal append true to ratio li

  # append mean squared value by passing true labels and predicted labels to the function mse create in above c



# the sum of ratio i.e sum of true elements. divided by length of test data

# Divide the sum of error by length of test data



# print the ratio

# print the error
```

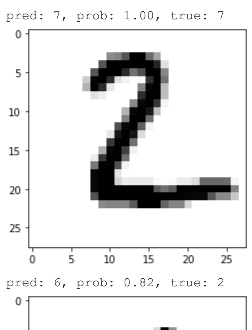
```
ratio: 0.87
mse: 0.0190
```

from the above otput we can understand that approx 87% data has been predicted correctly and the mean squared error obtained is approx 0.019.

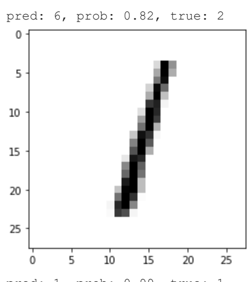
Let's visualize the prediction

We wil be plotting first 10 samples from the test data including the predicted, prediction probabiltiy and true label of each image.

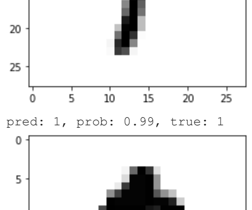
```python
# import the matplotlib.pyplot library


# create a variable named samples with initial value equal to 10


# loop through first 10 test data and test labels simulataneously using zip function

  # create a variable named image store reshaped current iterated test data sample to 28x28 pixel

  # plot data usig imshow with cmap = 'binary

  # show images using plt.show()

  # create a varibale pnamed pred to store predicted labels for current image

  # Get index of the maximum value in predicted label array

  # Get index of the maximum value in true label array


  # print index of predicted labels, index value of predicted label (i.e: probaility) and true labels
```



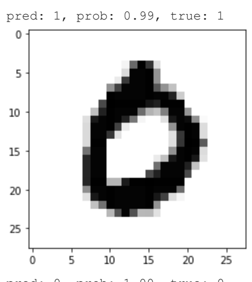
```
pred: 7, prob: 1.00, true: 7
```



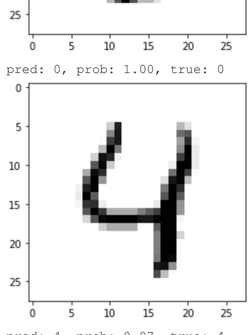
```
pred: 6, prob: 0.82, true: 2
```



```
pred: 1, prob: 0.99, true: 1
```



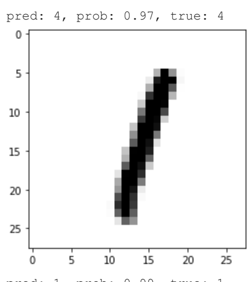
```
pred: 0, prob: 1.00, true: 0
```



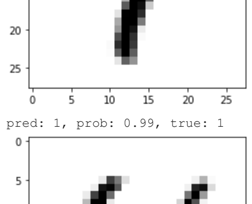
```
pred: 4, prob: 0.97, true: 4
```



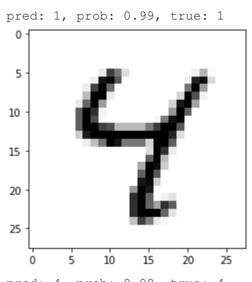
```
pred: 1, prob: 0.99, true: 1
```



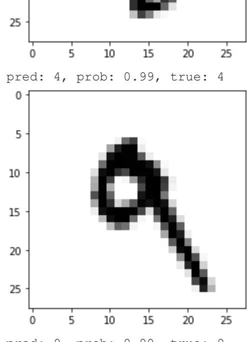
```
pred: 4, prob: 0.99, true: 4
```



```
pred: 9, prob: 0.99, true: 9
```



```
pred: 2, prob: 0.66, true: 5
```



```
pred: 9, prob: 0.87, true: 9
```

From the above sample images we can understand that the 80% of teh images have been predicted correctly. some have the probability of 100% and some have probaility more than 80%.

Assignment Summary

We build neural network from scratch.

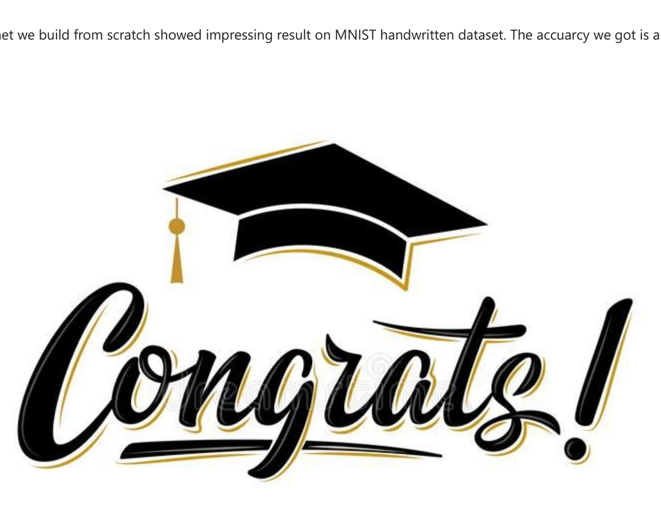
We learned following:

1. How to build each and every layer in neural net. such as flatten layer, fully connected layer, activation layer, output layer.
2. we learned to build and use various activation functions from scratch.
3. We learned to implement forward and backward propagation.

Inshort we learned all the mathematical terms involved in building neural net.

Conclusion

The neural net we build from scratch showed impressing result on MNIST handwritten dataset. The accuarcy we got is around 87%.



Congratlations!

You have completed this assignment and you are now to work on the next assignmnet