This notebook contains a class that uses numpy to build a simplest of NN and we the networks performance on MNIST dataset.

In [131]:

```
#Importing requires libraries
import numpy as np
import pandas as pd
import scipy.special
import matplotlib.pyplot as plt
%matplotlib inline
from tqdm import tqdm
```

#Neural Network Class

In [102]:

```
class neuralNetwork():

  def __init__(self, num_input_node, num_hidden_node, num_output_node, learning_rate):
    self.input_nodes = num_input_node
    self.hidden_nodes = num_hidden_node
    self.output_nodes = num_output_node
    self.learning_rate = learning_rate

    #Initializing weights
    self.wgt_ih = np.random.normal(0.0, pow(self.hidden_nodes, -0.5), (self.hidden_nod
    self.wgt_ho = np.random.normal(0.0, pow(self.output_nodes, -0.5),(self.output_node

    #Activation function definition
    self.activation_function = lambda x: scipy.special.expit(x)

  #Method used for training the model
  def train(self, input_list, target_list):

    input = np.array(input_list, ndmin = 2).T
    target = np.array(target_list, ndmin = 2).T

    input_hidden_layer = np.dot(self.wgt_ih, input)
    output_hidden_layer = self.activation_function(input_hidden_layer)

    input_final_layer = np.dot(self.wgt_ho, output_hidden_layer)
    output_final_layer = self.activation_function(input_final_layer)

    #Calculating error
    output_error = target - output_final_layer
    hidden_layer_error = np.dot(self.wgt_ho.T, output_error)

    #Updating/Learning weights from the error
    self.wgt_ho += self.learning_rate * np.dot((output_error * output_final_layer * (1
    self.wgt_ih += self.learning_rate * np.dot((hidden_layer_error * output_hidden_lay

  #Query method used for prediction using test data
  def query(self, input_list):

    input = np.array(input_list, ndmin = 2).T

    input_hidden_layer = np.dot(self.wgt_ih, input)
    output_hidden_layer = self.activation_function(input_hidden_layer)

    input_final_layer = np.dot(self.wgt_ho, output_hidden_layer)
    output_final_layer = self.activation_function(input_final_layer)

    return output_final_layer
```

#Loading Data

data source: http://pjreddie.com/projects/mnist-in-csv/ (http://pjreddie.com/projects/mnist-in-csv/) Here we can find the MNIST data set in csv format, we can simply download the files, in this each row contains information of an image, with first column being class label and rest of the columns as image pixel.

In [6]:

```
train_data = pd.read_csv('/content/drive/My Drive/mnist_train.csv')
test_data = pd.read_csv('/content/drive/My Drive/mnist_test.csv')
```

In [7]:

```
train_data.head()
```

Out[7]:

	5	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [8]:

```
test_data.head()
```

Out[8]:

	7	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Let us plot the image for 2 data points

In [17]:

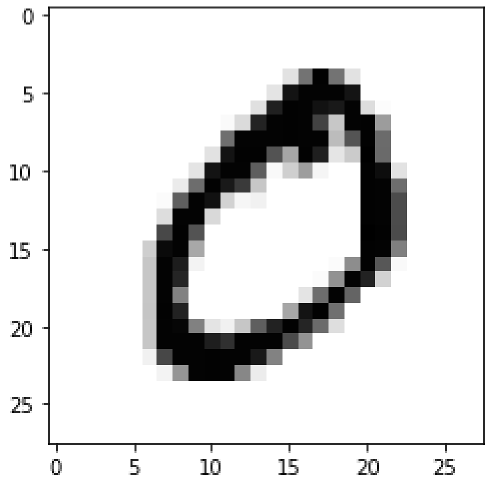
```
img_array = train_data.iloc[0][1:].values.reshape(28, 28)
```

In [21]:

```
1 plt.imshow(img_array, cmap = 'Greys')
```

Out[21]:

<matplotlib.image.AxesImage at 0x7fa9e89da630>



In [22]:

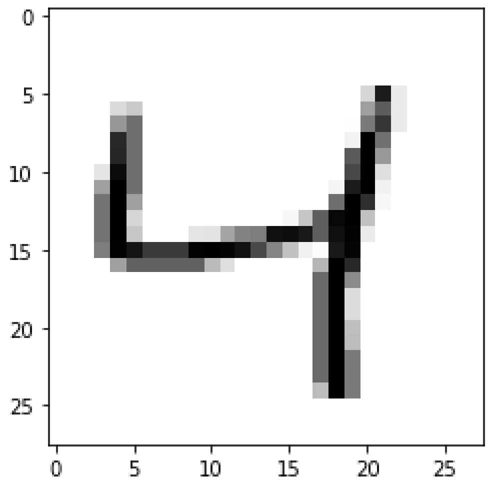
```
1 img_array = train_data.iloc[1][1:].values.reshape(28, 28)
```

In [23]:

```
1 plt.imshow(img_array, cmap = 'Greys')
```

Out[23]:

<matplotlib.image.AxesImage at 0x7fa9e84643c8>



#Data Preparation

In [60]:

```
train_x = train_data[train_data.columns[1:]].values
```

In [37]:

```
#train_y = train_data['5'].values
```

In [61]:

```
test_x = test_data[test_data.columns[1:]].values
#test_y = test_data['7'].values
```

In [62]:

```
print('shape of train is: {}'.format(train_x.shape))
print('shape of test is: {}'.format(test_x.shape))
```

```
shape of train is: (59999, 784)
shape of test is: (9999, 784)
```

Scaling data

We need to rescale the input from the larger rang of 0 to 255 toa much smaller range of 0.01 to 1, rescaling to 0.01 instead of zero is chosen to make sure none of our inputs are 0, as they can artificially kill weight updates.

input to the model

In [63]:

```
train_x = train_x /255.0 * .99 + 0.01
```

In [64]:

```
test_x = test_x / 255.0 * .99 + 0.01
```

output labels

Here we are trying to perform a classification task, with our labels being numbers between 0 and 9. So, output layer should have 10 nodes, one for each possible labels.

Now, our activation function is logistic which have values ranging from 0.0 and 1.0, also, we can't reach 0.0 and 1.0 as the logistic function only approaches these extremes without actually getting there. So, we may need to scale our target values during training as well.So, our target array would contain 10 elements. So, if we have a target of number 5, it should be [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]. Also, trying to get our NN producing outputs 0, 1 are impossible for activation function, so, we will scale it such that instead of 0 we use 0.1 and insted of 1 we use 0.99. So, for 5 this will look like: [0.1, 0.1, 0.1, 0.1, 0.1, 0.99, 0.1, 0.1, 0.1, 0.1].

In [80]:

```
def get_vector(value):
  vec = np.zeros(10) + 0.01
  vec[value] = 0.99
  return list(vec)
```

In [83]:

```
train_y = train_data[train_data.columns[0]].values
```

In [84]:

```
train_y = [get_vector(target) for target in train_y]
```

In [91]:

```
train_y = np.array(train_y)
```

In [92]:

```
train_y.shape
```

Out[92]:

```
(59999, 10)
```

In [93]:

```
test_y = test_data[test_data.columns[0]].values
test_y = [get_vector(target) for target in test_y]
test_y = np.array(test_y)
test_y.shape
```

Out[93]:

```
(9999, 10)
```

#Training the network

In [126]:

```
#Calling our neural netowrk class
input_nodes = 784
hidden_nodes = 500
output_nodes = 10
learning_rate = 0.1

nn = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)
```

Training the network

Doing it for 5 epochs, their are multiple hyperparameter to consider example the learning rate, number of hidden layers, epochs, with 5 epoch, 500 hidden layers and 0.1 learning rate, got best result...

In [128]:

```python
for epochs in range(5):
  for i in tqdm(range(train_x.shape[0])):
    nn.train(train_x[i], train_y[i])
```

```
100%|██████████| 59999/59999 [02:00<00:00, 497.12it/s]
100%|██████████| 59999/59999 [01:57<00:00, 511.80it/s]
100%|██████████| 59999/59999 [01:57<00:00, 511.54it/s]
100%|██████████| 59999/59999 [01:57<00:00, 510.28it/s]
100%|██████████| 59999/59999 [02:00<00:00, 496.81it/s]
```

In [129]:

```python
correct_prediction = 0
for i in tqdm(range(test_x.shape[0])):
  prediction = np.argmax(nn.query(test_x[i]))
  if prediction == test_data[test_data.columns[0]].values[i]:
    correct_prediction += 1
```

```
100%|██████████| 9999/9999 [00:02<00:00, 3642.25it/s]
```

In [130]:

```python
print('model performace is: {}'.format(correct_prediction/test_data.shape[0]))
```

```
model performace is: 0.9730973097309731
```

Wow the simple network did a great job if we see, model performance is pretty high, which is good. This notebook is not about creating the complex of the models, but about implementing the simplest of NN using python which indeed helped in understanding how a model actually works....