Supervised Learning

Assignment 2

Name: Yomna Taher Abdallah **Id:** 20190624 **S:3**

Name: Adel Abdelmonem Arfa **Id:** 20190280 **S:3**

By using NN we classified the handwritten digits (MNIST)……

By using 60,000 samples from the dataset for training and 10,000 for testing and saved the values in a 1-dimensional array of dimension 784 after doing this we change the type to a 2-dimensional array with a dimensions 28*28 to be more easily in manipulated and visualized, so we can split the images into grids.

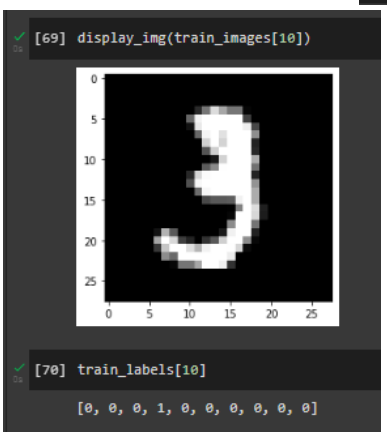
And by reshaping function, we used it to get the images as vector of 10 with zeros in all indexes and 1 in only the indexes of the number.

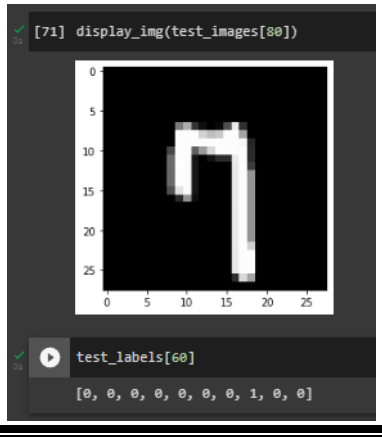
```
def label_reshape(arr_labele):
    label_vector=[]
    for i in range(len(arr_labele)):
        label_vector.append([0 for _ in range(10)])
        x = arr_labele[i]
        label_vector[i][x] = 1
    return label_vector
```

```
[67] train_labels = label_reshape(train_labels)
     test_labels = label_reshape(test_labels)
```

```
[68] def display_img(mnist_index):

        image = mnist_index
        image = np.array(image, dtype='float')
        pixels = image.reshape((28, 28))
        plt.imshow(pixels, cmap='gray')
        plt.show()
```





After we split the data to grids, we get the centroid for every grid by getting the centroid of the Xs and Ys.

```
[ ] def imaged_grid(img , row , col ):

        x , y = img.shape

        assert x % row == 0, "{} rows is not evenly divisble by {}".format(x, row)
        assert y % col == 0, "{} cols is not evenly divisble by {}".format(y, col)

        return (img.reshape ( x //row, row, -1, col)
                  .swapaxes(1,2)
                  .reshape(-1, row, col))
```

```
[ ] print(imaged_grid(test_images[20] , 7 , 14 ).shape)
    imaged_grid(test_images[20] , 7 , 14 )
```

```
          [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
             0,   0],
          [  0,  31, 140, 193,  44,   0,   0,   0,   0,   0,   0,   0,
             0,   0]],

         [[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
             0,  38],
          [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   2,
            43, 230],
          [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  21, 130,
           254, 254],
          [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 154, 254,
```

```
def get_centroid(img):

    feature_vector = []

    for grid in imaged_grid(img , 7 , 7 ) :

        Xc = 0
        Yc = 0
        sum = 0

        for index, x in np.ndenumerate(grid):
          sum+= x
          Xc += x * index[0]
          Yc += x * index[1]

        if sum != 0 :
            feature_vector.append( (Xc/ sum)/6 )
            feature_vector.append((Yc/ sum )/6)
        else :
             feature_vector.append(0)
             feature_vector.append(0)


    return np.array(feature_vector)
```

```
[ ] get_centroid(test_images[20])
```

```
array([0.        , 0.        , 0.        , 0.        , 1.        ,
       0.43545752, 0.        , 0.        , 0.        , 0.        ,
       0.69624439, 0.67855843, 0.48555192, 0.48490978, 0.26151013,
       0.00184162, 0.        , 0.        , 0.35787962, 0.69891631,
       0.41040735, 0.24892081, 0.        , 0.        , 0.        ,
       0.        , 0.33032507, 0.72876257, 0.10180995, 0.        ,
       0.        , 0.        ])
```

```
[ ] get_centroid(train_images[20])
```

```
array([0.9853211 , 0.65993884, 0.        , 0.        , 0.87639198,
       0.70019797, 0.        , 0.        , 0.40889044, 0.83136026,
       0.7631751 , 0.18229371, 0.58078624, 0.60054172, 0.        ,
```

Now let's talk about NN:

For first we build the layer class which will take the inputs of data and inside it will be the summation of the inputs' weights then get into the activation function which is the "sigmoid function" then get the output for the second layer, this way called "Forward propagation".

Then get the accuracy and see if there is a large error, we go throw the second way which is "Back propagation", in this way we compare the output with our label and get back with changing the weights of the output layer if the needed and then get back and check the input's weights.

After doing this we get back to the Forward propagation again.

The second class in NN is class Network:

In this class we take the data and train labels and the learning rate to use the ways of Forward propagation and Back propagation to get our output, so we can say that this class is the processor of the NN.

```
class Network:
    def __init__(self):
        self.layers = []

    def add(self, layer ):
        self.layers.append(layer )


    def predict(self, input_data):

        predicted = []
        for i in range( len(input_data) ):
            output = input_data[i]
            for layer in self.layers:
                output = layer.forward_propagation(output )
            predicted.append(output)

        return predicted


    def fit(self, train_data, train_labels, epochs, learning_rate):

        samples = len(train_data)

        # training loop
        for epoch in range(epochs):

            for sample in range(samples):

                cur_deltas = []

                output = train_data[sample]

                for layer in self.layers:
                    output = layer.forward_propagation(output )


                for layer in reversed(self.layers):
                    cur_deltas = layer.back_propagation(cur_deltas, learning_rate , train_labels[sample])
```

We use 2 layers in our NN:

First Layer has 32 input and get 16 outputs.

Second layer get the first 16 outputs as input and return 10 outputs.

The second layer return 10 output because our dataset is images of numbers from 0 to 9, so we need 10 outputs.

And we use 100 epoch and 0.05 learning rate.

```
[ ]
    NN = Network()
    NN.add(Layer(32, 16 , "sigmoid"))
    NN.add(Layer(16, 10 , "sigmoid"))
    NN.fit(train_features, train_labels, epochs=100, learning_rate= 0.05)
```

The last function is Accuracy function:

This function takes the predicted results of our NN and the main labels, then goes throw all of them and get the ratio between every result in the predicted values and the labels values.

```
[ ] def acc(predict,labels):
      X = []
      Y = []
      for i in predict:
        max_value = max(i)
        max_index = i.index(max_value)
        X.append(max_index)
      for j in labels:
        max_value = max(j)
        max_index = j.index(max_value)
        Y.append(max_index)
      rLabels = 0
      labels = len(X)
      for i in range(len(X)):
        if X[i]==Y[i]:
          rLabels=rLabels+1
      return(rLabels/labels)*100
```

```
print("Accuracy Score = {} %".format(acc(p,test_labels)))
```

```
Accuracy Score = 86.41 %
```