Multilayerperceptron

February 21, 2024

1 Training an MLP from Scratch

2 1- init__params(nx, nh, ny)

[40]:
```
import numpy as np
```

[41]:
```
def init_params(nx,nh,ny):
    #nx is the nbr of neurons in the input layer
    #nh is the nbr of neurons in the hidden layer
    #ny is the nbr of neurons in the output layer
    # shape te3 w hwa (nx+1,nh)
    w1=np.random.normal(loc=0.0, scale=0.3, size=(nh,nx+1))
    w2=np.random.normal(loc=0.0, scale=0.3, size=(ny,nh+1))
    params={"w1":w1,
            "w2":w2}
    return params
```

[42]:
```
params=init_params(2,2,4)
```

[43]:
```
params
```

[43]:
```
{'w1': array([[-0.03617072,  0.11845869, -0.55649705],
        [-0.15783306, -0.19837909, -0.36638267]]),
 'w2': array([[ 0.05017689, -0.40892273, -0.20164522],
        [ 0.7846449 ,  0.15423692, -0.14586556],
        [-0.4479588 , -0.11213297, -0.48424308],
        [-0.34472698, -0.15315491,  0.13525811]])}
```

[44]:
```
params['w1']
```

[44]:
```
array([[-0.03617072,  0.11845869, -0.55649705],
       [-0.15783306, -0.19837909, -0.36638267]])
```

[45]:
```
params['w2']
```

[45]:
```
array([[ 0.05017689, -0.40892273, -0.20164522],
       [ 0.7846449 ,  0.15423692, -0.14586556],
```

```
       [-0.4479588 , -0.11213297, -0.48424308],
       [-0.34472698, -0.15315491,  0.13525811]])
```

[46]:
```
X=np.array([[0.5403, -0.4161],[-0.9900,-0.6536],[0.2837,0.9602]])
X.shape
```

[46]:
```
(3, 2)
```

[47]:
```
m = X.shape[0]
X=np.c_[np.ones((m,1)),X]
X
```

[47]:
```
array([[ 1.    ,  0.5403, -0.4161],
       [ 1.    , -0.99  , -0.6536],
       [ 1.    ,  0.2837,  0.9602]])
```

3 2- Forward prop

[48]:
```
#from scipy.special import softmax
def tanh(x):
    return np.tanh(x)
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)
def forward(params,X):
    #b = X.shape[0]
    #X=np.c_[np.ones((b,1)),X]
    z1=np.matmul(params['w1'],X.transpose())
    a1=np.tanh(z1)
    m = a1.shape[1]
    a1=np.r_[np.ones((1,m)),a1]
    z2=np.matmul(params['w2'],a1)
    y_hat=softmax(z2)
    outputs={"z1":z1,
        "a1":a1,
        'z2':z2}
    return outputs,y_hat
```

[49]:
```
outputs,y_hat = forward(params,X)
outputs['a1']
```

[49]:
```
array([[ 1.        ,  1.        ,  1.        ],
       [ 0.25372577,  0.20723607, -0.49064726],
       [-0.11209241,  0.27108077, -0.51235207]])
```

[50]:
```
y_hat
```

[50]: array([[0.21016811, 0.2103051 , 0.27443618],
       [0.5022065 , 0.50012354, 0.42170043],
       [0.14213145, 0.12587916, 0.1666263],
       [0.14549394, 0.1636922 , 0.13723709]])

[51]:
```
sum_of_softmax = np.sum(y_hat)
print(f"Sum of Softmax values: {sum_of_softmax}")
```

Sum of Softmax values: 3.0

4 3- Loss_Accuracy

[52]:
```
y=np.array([[0,0,0,1],[0,1,0,0],[0,0,1,0]])
y.transpose().shape
```

[52]: (4, 3)

[53]:
```
def loss_accuracy(y, y_hat):
   y=y.transpose()
   epsilon = 1e-10  # Small constant to avoid log(0)
   loss = -np.sum(y * np.log(y_hat)) / y.shape[1]
   accuracy = np.mean(np.argmax(y_hat, axis=0) == np.argmax(y, axis=0))

   return loss, accuracy
```

[54]:
```
loss,accuracy=loss_accuracy(y,y_hat)
loss
```

[54]: 1.4708408874785344

[55]:
```
accuracy
```

[55]: 0.3333333333333333

5 4- Backward prop

[56]:
```
def backward(x,params,outputs,y_hat, y):
    n=y.shape[0]
    y=y.transpose()
    w1=params['w1']
    w2=params['w2']
    z1=outputs['z1']
    z2=outputs['z2']
    a1=outputs['a1']
    dj_dyhat=y_hat-y #derivative of the cross entropy loss
```

```
    dyhat_dz2=softmax(z2)*(1-softmax(z2))#1# derivative of the softmax␣
↪activation fct
    dz2_dw2=a1
    dj_dw2=np.dot(dj_dyhat*dyhat_dz2,dz2_dw2.transpose())
    # dj_dw1

    dz2_da1=w2[:,1:]
    da1_dz1=1 - np.tanh(z1)**2
    dz1_dw1=x
    dj_dz2=dj_dyhat*dyhat_dz2

    dj_da1=np.dot(dj_dz2.transpose(),dz2_da1)
    dj_dz1=dj_da1.transpose()*da1_dz1
    dj_dw1=np.dot(dj_dz1,dz1_dw1)
    # weights update
    grads={"dj_dw1":dj_dw1,
           "dj_dw2":dj_dw2}

    return grads
```

[57]:
```
grads=backward(X,params,outputs,y_hat, y)
grads['dj_dw1'].shape
```

[57]: (2, 3)

[58]:
```
grads['dj_dw1']
```

[58]:
```
array([[-0.01567247,  0.04720056,  0.01946558],
       [-0.01678777, -0.02581464,  0.03803312]])
```

[59]:
```
grads['dj_dw2']
```

[59]:
```
array([[ 0.12446019, -0.01072199, -0.02244061],
       [ 0.10341981, -0.04450108, -0.10063999],
       [-0.08454288,  0.06404713,  0.06110354],
       [-0.06757858, -0.03028375,  0.00965763]])
```

6 mini batch sgd

[60]:
```
def sgd(params, grads, eta):

     params['w1']-=eta*grads['dj_dw1']
     params['w2']-=eta*grads['dj_dw2']
     return params
```

[61]:
```
sgd(params,grads,0.1)
```

[61]:
```
{'w1': array([[-0.03460347,  0.11373863, -0.55844361],
        [-0.15615428, -0.19579762, -0.37018598]]),
 'w2': array([[ 0.03773087, -0.40785053, -0.19940116],
        [ 0.77430291,  0.15868703, -0.13580156],
        [-0.43950451, -0.11853768, -0.49035344],
        [-0.33796912, -0.15012654,  0.13429234]])}
```

[62]:
```python
import matplotlib.pyplot as plt
def mini_batch_sgd(X, y, params, batch_size=128, epochs=20, eta=0.01):
    loss_history = []
    accuracy_history = []

    for epoch in range(epochs):
        total_loss = 0
        correct_predictions = 0
        #random data
        shuffle_indices = np.random.permutation(len(X))
        X= X[shuffle_indices]
        y = y[shuffle_indices]


        for i in range(0, len(X), batch_size):
            #Load a batch of data
            x_batch = X[i:i + batch_size]
            y_batch = y[i:i + batch_size]

            outputs, y_hat = forward(params, x_batch)
            grads = backward(x_batch, params, outputs, y_hat, y_batch)
            params = sgd(params, grads, eta)

            batch_loss, batch_accuracy = loss_accuracy(y_batch, y_hat)
            total_loss += batch_loss
            correct_predictions += batch_accuracy * len(y_batch)

        average_loss = total_loss / (len(X) // batch_size)
        accuracy = correct_predictions / len(X)

        loss_history.append(average_loss)
        accuracy_history.append(accuracy)

        print(f"Epoch {epoch + 1}/{epochs} - Loss: {average_loss:.4f} -␣
↪Accuracy: {accuracy:.4f}")

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(loss_history, label='Training Loss')
```

```
    plt.title('Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(accuracy_history, label='Training Accuracy')
    plt.title('Training Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.show()
```

Parameters: learning rate= 0.1, batch size= 128, nbr of iterations= 50

7 MNIST dataset import

[63]:
```
import tensorflow as tf
import tensorflow.keras as keras
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

[64]:
```
print('X_train: ' + str(x_train.shape))
print('Y_train: ' + str(y_train.shape))
print('X_test:  '  + str(x_test.shape))
print('Y_test:  '  + str(y_test.shape))
```

```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

8 Reshaping

[65]:
```
x_train=x_train.reshape(x_train.shape[0],x_train.shape[1]*x_train.shape[2])
x_test=x_test.reshape(x_test.shape[0],x_test.shape[1]*x_test.shape[2])
```

[66]:
```
print('x_train: ' + str(x_train.shape))
print('y_train: ' + str(y_train.shape))
print('x_test:  '  + str(x_test.shape))
print('y_test:  '  + str(y_test.shape))
```

```
x_train: (60000, 784)
y_train: (60000,)
x_test:  (10000, 784)
y_test:  (10000,)
```

[67]:
```
x_train[0]
```

[67]:
```
array([  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,   3,  18,  18,  18,
       126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
       253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
       253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
       253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,  14,   1, 154, 253,  90,   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, 139, 253, 190,   2,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,  70,
         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,  35,
       241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,  45, 186, 253, 253, 150,  27,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  16,  93, 252, 253, 187,
         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, 249,
       253, 249,  64,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  46, 130,
       183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
       229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114,
       221, 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,
```

```
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  23,  66,
213, 253, 253, 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 171,
219, 253, 253, 253, 253, 195,  80,   9,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  55, 172,
226, 253, 253, 253, 253, 244, 133,  11,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
136, 253, 253, 253, 212, 135, 132,  16,   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], dtype=uint8)
```

9 Normalization –> (X — Xmin)/(Xmax-Xmin) = X/255

[68]:
```
x_train=x_train/255
x_test=x_test/255
```

[69]:
```
x_train[0]
```

[69]:
```
array([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.        , 0.01176471, 0.07058824, 0.07058824,
0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.11764706, 0.14117647, 0.36862745, 0.60392157,
0.66666667, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.88235294, 0.6745098 , 0.99215686, 0.94901961,
0.76470588, 0.25098039, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.19215686, 0.93333333,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.98431373, 0.36470588,
0.32156863, 0.32156863, 0.21960784, 0.15294118, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.07058824, 0.85882353, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.77647059, 0.71372549,
0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.31372549, 0.61176471, 0.41960784, 0.99215686, 0.99215686,
0.80392157, 0.04313725, 0.        , 0.16862745, 0.60392157,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.05490196,
0.00392157, 0.60392157, 0.99215686, 0.35294118, 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.        , 0.54509804,
0.99215686, 0.74509804, 0.00784314, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.04313725, 0.74509804, 0.99215686,
0.2745098 , 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.        , 0.1372549 , 0.94509804, 0.88235294, 0.62745098,
0.42352941, 0.00392157, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.31764706, 0.94117647, 0.99215686, 0.99215686, 0.46666667,
0.09803922, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.17647059,
0.72941176, 0.99215686, 0.99215686, 0.58823529, 0.10588235,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.0627451 , 0.36470588,
0.98823529, 0.99215686, 0.73333333, 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.        , 0.97647059, 0.99215686,
0.97647059, 0.25098039, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.18039216, 0.50980392,
0.71764706, 0.99215686, 0.99215686, 0.81176471, 0.00784314,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.15294118,
0.58039216, 0.89803922, 0.99215686, 0.99215686, 0.99215686,
0.98039216, 0.71372549, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.09019608, 0.25882353, 0.83529412, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.77647059, 0.31764706,
0.00784314, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.07058824, 0.67058824, 0.85882353,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.76470588,
0.31372549, 0.03529412, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.21568627, 0.6745098 ,
0.88627451, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.95686275, 0.52156863, 0.04313725, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.53333333, 0.99215686, 0.99215686, 0.99215686,
0.83137255, 0.52941176, 0.51764706, 0.0627451 , 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.        ])
```

10 Labeling : 5–> [0,0,0,0,0,1,0,0,0,0] (one hot encoding)

[70]:
```
from keras.utils import to_categorical
print("class label for first image",y_train[0])
y_train=tf.keras.utils.to_categorical(y_train,10)
#y_test=tf.keras.utils.to_categorical(y_test,10)
print("class label for first image after labeling",y_train[0])
```

```
class label for first image 5
class label for first image after labeling [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

[71]:
```
y_train.shape
```

[71]:
```
(60000, 10)
```

[72]:
```
nx = x_train.shape[1]
nh = 32
ny = y_train.shape[1]
```

[73]:
```
params=init_params(nx,nh,ny)
```

[74]:
```
params['w1'].shape
```

[74]:
```
(32, 785)
```

[75]:
```
params['w2'].shape
```

[75]:
```
(10, 33)
```

11 adding the bias term for x_train and x_test in order to be passed to the forward function

[76]:
```
m = x_train.shape[0]
x_train=np.c_[np.ones((m,1)),x_train]
x_train
```

[76]:
```
array([[1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       …,
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.]])
```

[77]:
```
m = x_test.shape[0]
x_test=np.c_[np.ones((m,1)),x_test]
```

```
x_test
```

[77]:
```
array([[1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       …,
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.],
       [1., 0., 0., …, 0., 0., 0.]])
```

[78]:
```
mini_batch_sgd(x_train,y_train,params,batch_size=128,epochs=50,eta=0.01)
```

```
Epoch 1/50 - Loss: 0.7081 - Accuracy: 0.7869
Epoch 2/50 - Loss: 0.3733 - Accuracy: 0.8946
Epoch 3/50 - Loss: 0.3178 - Accuracy: 0.9131
Epoch 4/50 - Loss: 0.2843 - Accuracy: 0.9243
Epoch 5/50 - Loss: 0.2606 - Accuracy: 0.9318
Epoch 6/50 - Loss: 0.2425 - Accuracy: 0.9359
Epoch 7/50 - Loss: 0.2276 - Accuracy: 0.9403
Epoch 8/50 - Loss: 0.2158 - Accuracy: 0.9441
Epoch 9/50 - Loss: 0.2064 - Accuracy: 0.9470
Epoch 10/50 - Loss: 0.1975 - Accuracy: 0.9496
Epoch 11/50 - Loss: 0.1897 - Accuracy: 0.9519
Epoch 12/50 - Loss: 0.1830 - Accuracy: 0.9539
Epoch 13/50 - Loss: 0.1770 - Accuracy: 0.9562
Epoch 14/50 - Loss: 0.1716 - Accuracy: 0.9581
Epoch 15/50 - Loss: 0.1662 - Accuracy: 0.9597
Epoch 16/50 - Loss: 0.1618 - Accuracy: 0.9612
Epoch 17/50 - Loss: 0.1575 - Accuracy: 0.9625
Epoch 18/50 - Loss: 0.1533 - Accuracy: 0.9639
Epoch 19/50 - Loss: 0.1503 - Accuracy: 0.9647
Epoch 20/50 - Loss: 0.1468 - Accuracy: 0.9656
Epoch 21/50 - Loss: 0.1433 - Accuracy: 0.9666
Epoch 22/50 - Loss: 0.1407 - Accuracy: 0.9678
Epoch 23/50 - Loss: 0.1373 - Accuracy: 0.9687
Epoch 24/50 - Loss: 0.1353 - Accuracy: 0.9696
Epoch 25/50 - Loss: 0.1326 - Accuracy: 0.9703
Epoch 26/50 - Loss: 0.1304 - Accuracy: 0.9711
Epoch 27/50 - Loss: 0.1277 - Accuracy: 0.9718
Epoch 28/50 - Loss: 0.1256 - Accuracy: 0.9724
Epoch 29/50 - Loss: 0.1238 - Accuracy: 0.9736
Epoch 30/50 - Loss: 0.1219 - Accuracy: 0.9736
Epoch 31/50 - Loss: 0.1199 - Accuracy: 0.9744
Epoch 32/50 - Loss: 0.1183 - Accuracy: 0.9752
Epoch 33/50 - Loss: 0.1167 - Accuracy: 0.9757
Epoch 34/50 - Loss: 0.1150 - Accuracy: 0.9764
Epoch 35/50 - Loss: 0.1131 - Accuracy: 0.9766
```

```
Epoch 36/50 - Loss: 0.1119 - Accuracy: 0.9773
Epoch 37/50 - Loss: 0.1105 - Accuracy: 0.9776
Epoch 38/50 - Loss: 0.1087 - Accuracy: 0.9783
Epoch 39/50 - Loss: 0.1076 - Accuracy: 0.9785
Epoch 40/50 - Loss: 0.1070 - Accuracy: 0.9788
Epoch 41/50 - Loss: 0.1053 - Accuracy: 0.9793
Epoch 42/50 - Loss: 0.1042 - Accuracy: 0.9798
Epoch 43/50 - Loss: 0.1030 - Accuracy: 0.9803
Epoch 44/50 - Loss: 0.1017 - Accuracy: 0.9804
Epoch 45/50 - Loss: 0.1005 - Accuracy: 0.9807
Epoch 46/50 - Loss: 0.0998 - Accuracy: 0.9812
Epoch 47/50 - Loss: 0.0986 - Accuracy: 0.9816
Epoch 48/50 - Loss: 0.0981 - Accuracy: 0.9819
Epoch 49/50 - Loss: 0.0972 - Accuracy: 0.9822
Epoch 50/50 - Loss: 0.0958 - Accuracy: 0.9823
```