Name: Venkata Meghana Achanta

USC ID: 2578990261

Q1.

Q1. $W^{(1)} = \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix}$ $b^{(1)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

$W^{(2)} = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}$ $b^{(2)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$W^{(3)} = \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix}$ $b^{(3)} = \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$

$X = [+1 \quad -1 \quad +1]^T$

$s^{(1)} = W^1 X + b^1$

$= \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

$= \begin{bmatrix} 1+2+1 \\ 3-4-2 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

$= \begin{bmatrix} 4 \\ -3 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

$= \begin{bmatrix} 5 \\ -5 \end{bmatrix}$

$h(s^{(1)}) = \text{relu} \begin{bmatrix} 5 \\ -5 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$

$a'(s) = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$; $\dot{a}^{(1)} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$

$s^{(2)} = W^{(2)} a^{(1)} + b^{(2)}$

$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$= \begin{bmatrix} 5 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$= \begin{bmatrix} 6 \\ 15 \end{bmatrix}$

$a^{(2)} = h(s^{(2)}) = \begin{bmatrix} 6 \\ 15 \end{bmatrix}$

$\dot{a}^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$g^{(3)} = W^3 a^{(2)} + b^3$

$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 15 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$

$= \begin{bmatrix} 12+30 \\ 18-45 \\ 12+15 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$

$= \begin{bmatrix} 42 \\ -27 \\ 27 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$

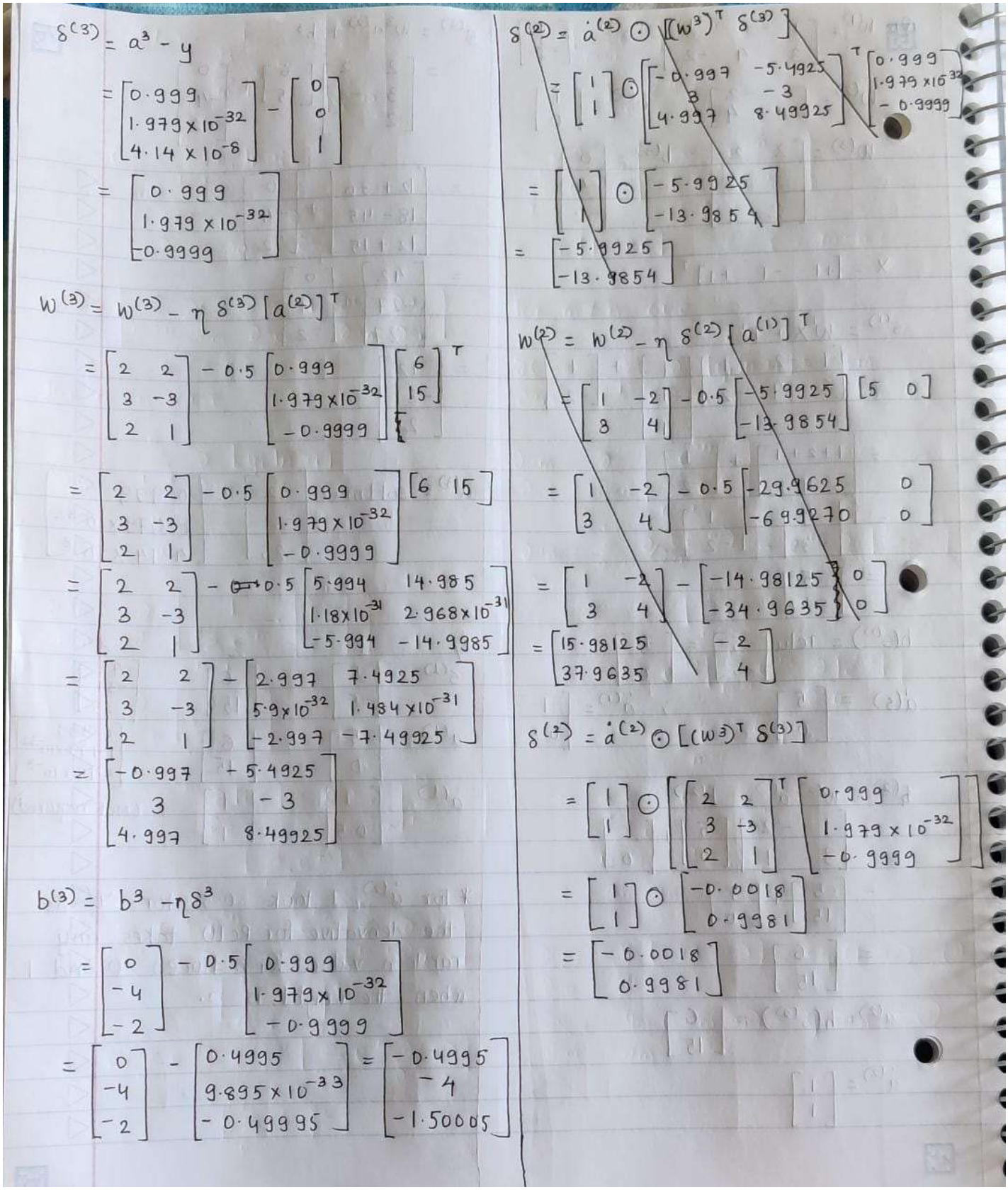
$= \begin{bmatrix} 42 \\ -31 \\ 25 \end{bmatrix}$

$a^{(3)} = \text{softmax} \begin{bmatrix} 42 \\ -31 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ 4.14 \times 10^{-8} \end{bmatrix}$

a)

	1	2	3
$s^{(1)}$	$\begin{bmatrix} 5 \\ -5 \end{bmatrix}$	$\begin{bmatrix} 6 \\ 15 \end{bmatrix}$	$\begin{bmatrix} 42 \\ -31 \\ 25 \end{bmatrix}$
$a^{(1)}$	$\begin{bmatrix} 5 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 6 \\ 15 \end{bmatrix}$	$\begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ 4.14 \times 10^{-8} \end{bmatrix}$
$\dot{a}^{(1)}$	$\begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	(Not required)

* For $\dot{a}^{(2)}$, I took 0.5 for '0' as the derivative for ReLU takes any random value ~~for~~ between 0 and 1 when the weight is 0.


$\delta^{(3)} = a^3 - y$
$= \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ 4.14 \times 10^{-8} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
$= \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix}$
$w^{(3)} = w^{(3)} - \eta\, \delta^{(3)} [a^{(2)}]^T$
$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix} \begin{bmatrix} 6 \\ 15 \end{bmatrix}^T$
$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix} \begin{bmatrix} 6 & 15 \end{bmatrix}$
$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \begin{bmatrix} 5.994 & 14.985 \\ 1.18 \times 10^{-31} & 2.968 \times 10^{-31} \\ -5.994 & -14.9985 \end{bmatrix}$
$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 2.997 & 7.4925 \\ 5.9 \times 10^{-32} & 1.484 \times 10^{-31} \\ -2.997 & -7.49925 \end{bmatrix}$
$= \begin{bmatrix} -0.997 & +5.4925 \\ 3 & -3 \\ 4.997 & 8.49925 \end{bmatrix}$
$b^{(3)} = b^3 - \eta \delta^3$
$= \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix} - 0.5 \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix}$
$= \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix} - \begin{bmatrix} 0.4995 \\ 9.895 \times 10^{-33} \\ -0.49995 \end{bmatrix} = \begin{bmatrix} -0.4995 \\ -4 \\ -1.50005 \end{bmatrix}$
$\delta^{(2)} = \dot{a}^{(2)} \odot [(w^3)^T \delta^{(3)}]$
$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} -0.997 & -5.4925 \\ 3 & -3 \\ 4.997 & 8.49925 \end{bmatrix}^T \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix}$
$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} -5.9925 \\ -13.9854 \end{bmatrix}$
$= \begin{bmatrix} -5.9925 \\ -13.9854 \end{bmatrix}$
$w^{(2)} = w^{(2)} - \eta\, \delta^{(2)} [a^{(1)}]^T$
$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - 0.5 \begin{bmatrix} -5.9925 \\ -13.9854 \end{bmatrix} \begin{bmatrix} 5 & 0 \end{bmatrix}$
$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - 0.5 \begin{bmatrix} -29.9625 & 0 \\ -69.9270 & 0 \end{bmatrix}$
$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} -14.98125 & 0 \\ -34.9635 & 0 \end{bmatrix}$
$= \begin{bmatrix} 15.98125 & -2 \\ 37.9635 & 4 \end{bmatrix}$
$\delta^{(2)} = \dot{a}^{(2)} \odot [(w^3)^T \delta^{(3)}]$
$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \left[\begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix}^T \begin{bmatrix} 0.999 \\ 1.979 \times 10^{-32} \\ -0.9999 \end{bmatrix} \right]$
$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} -0.0018 \\ 0.9981 \end{bmatrix}$
$= \begin{bmatrix} -0.0018 \\ 0.9981 \end{bmatrix}$

Q2.

2. $a = h(s) = \frac{1}{\sum_{m=1}^{M} e^{s_m}} \begin{bmatrix} e^{s_1} \\ e^{s_2} \\ \vdots \\ e^{s_M} \end{bmatrix}$

$C = -\sum_{i=1}^{n} y_i \ln a_i$

$\delta = \frac{\partial C}{\partial s_n} = \frac{\partial C}{\partial a_n} \cdot \frac{\partial a_n}{\partial s_n}$

$\delta = \left[\sum_{m \neq n} \frac{\partial C}{\partial a_m} \cdot \frac{\partial a_m}{\partial s_n} + \frac{\partial C}{\partial a_n} \cdot \frac{\partial a_n}{\partial s_n} \right]$ —①

For $m = n$ case

$\frac{\partial C}{\partial a_n} = -\sum_{i=1}^{n} \frac{\partial}{\partial a_n} [y_i \ln a_i]$

$= -\frac{y_n}{a_n}$

$\frac{\partial a_n}{\partial s_n} = \frac{\partial}{\partial s_n} \left[\frac{1}{\sum_{m=1}^{M} e^{s_m}} \begin{bmatrix} e^{s_1} \\ \vdots \\ e^{s_M} \end{bmatrix} \right]$

$= \frac{\partial}{\partial s_n} \left[\frac{e^{s_n}}{\sum_{m=1}^{M} e^{s_m}} \right]$

$= \frac{\sum_{m=1}^{M} e^{s_m} \frac{\partial [e^{s_n}]}{\partial s} - e^{s_n} \frac{\partial}{\partial s_n} \left[\sum_{m=1}^{M} e^{s_m} \right]}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{\sum_{m=1}^{M} e^{s_m} \cdot e^{s_n} - e^{s_n} \left[\sum_{m=1}^{M} \frac{\partial e^{s_m}}{\partial s_n} \right]}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{e^{s_n} \sum_{m=1}^{M} e^{s_m} - e^{s_n} \left[\sum_{m=1}^{M} e^{s_m} \frac{\partial (s_m)}{\partial s_n} \right]}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{e^{s_n} \left[\sum_{m=1}^{M} e^{s_m} \right] - e^{s_n} \cdot e^{s_m}}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{e^{s_n}}{\sum_{m=1}^{M} e^{s_m}} \left[1 - \frac{e^{s_m}}{\sum_{m=1}^{M} e^{s_m}} \right]$

$= a_n [1 - a_n]$ —② {∵ $m = n$}

For $m \neq n$ case

$\frac{\partial C}{\partial a_m} = -\sum_{i=1}^{n} \frac{\partial}{\partial a_m} [y_i \ln a_i] = -\frac{y_m}{a_m}$

$\frac{\partial a_m}{\partial s_n} = \frac{\partial}{\partial s_n} \left[\frac{e^{s_m}}{\sum_{m=1}^{M} e^{s_m}} \right]$

$\frac{\partial a_m}{\partial s_n} = \frac{\sum_{m=1}^{M} \partial/\partial s_n (e^{s_m}) - e^{s_m} \frac{\partial}{\partial s_n} \left[\sum_{m=1}^{M} e^{s_m} \right]}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{0 - e^{s_m} \left[\sum_{m=1}^{M} e^{s_m} \frac{\partial [e^{s_n}]}{\partial s_n} \right]}{\left[\sum_{m=1}^{M} e^{s_m} \right]^2}$

$= \frac{-e^{s_m} \cdot e^{s_n}}{\sum_{m=1}^{M} e^{s_m} \cdot \sum_{m=1}^{M} e^{s_m}}$

$= -a_m \cdot a_n$ —②

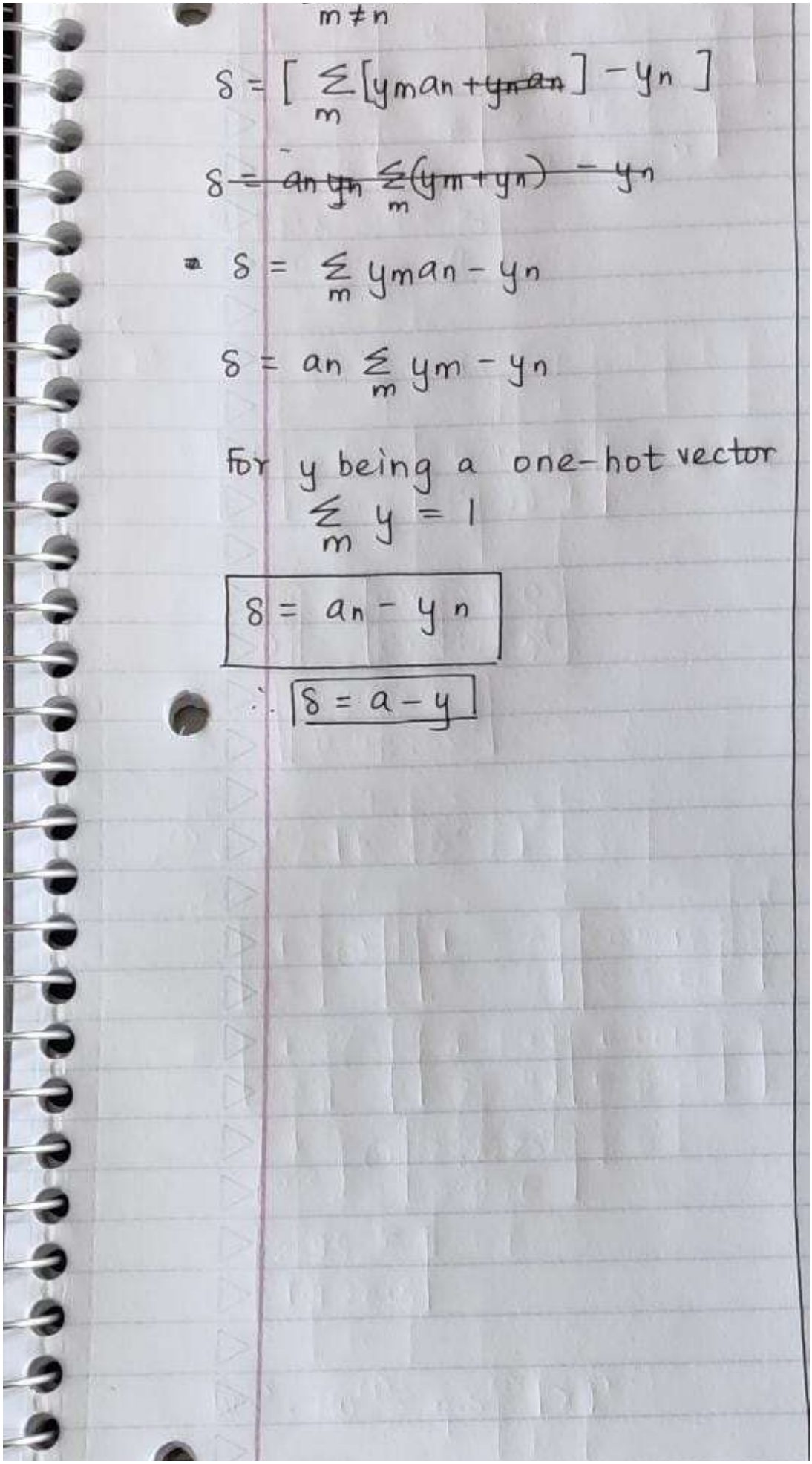
$\frac{\partial C}{\partial s_n} = -\frac{y_n}{a_n} [a_n (1 - a_n)] = -y_n (1 - a_n)$ —③

$\frac{\partial C}{\partial s_m} = -\frac{y_m}{a_m} (-a_m \cdot a_n) = y_m a_n$ —④

Substituting ③ and ④ in ①

$\delta = \left[\sum_{m \neq n} y_m a_n - y_n (1 - a_n) \right]$

$\delta = \left[\sum y_m a_n - y_n + y_n a_n \right]$



$m \neq n$

$$\delta = \left[\sum_m [y_m a_n + \cancel{y_n a_n}] - y_n \right]$$

$$\cancel{\delta = a_n y_n \sum_m (y_m + y_n) - y_n}$$

$$\delta = \sum_m y_m a_n - y_n$$

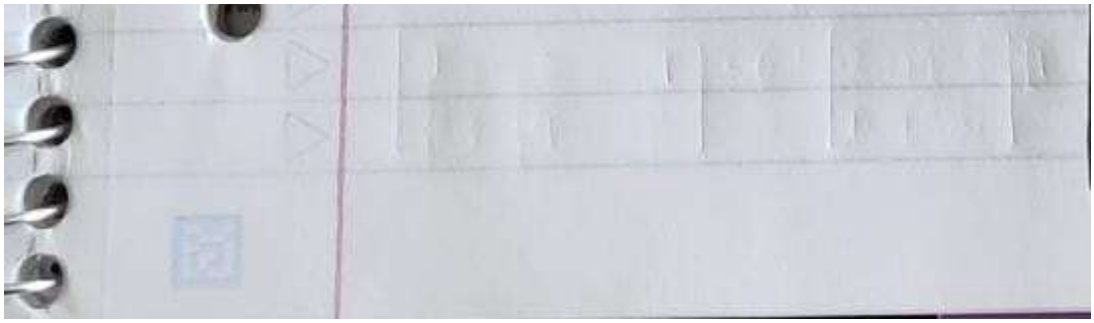
$$\delta = a_n \sum_m y_m - y_n$$

for y being a one-hot vector

$$\sum_m y = 1$$

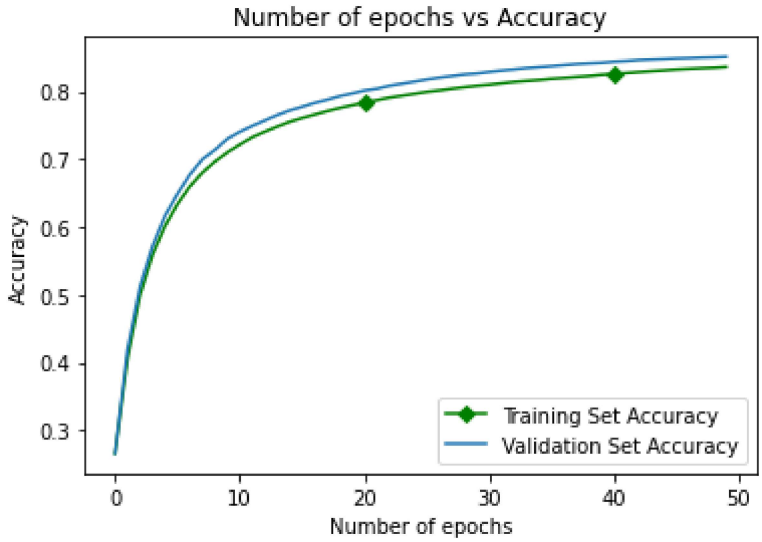
$$\boxed{\delta = a_n - y_n}$$

$$\therefore \boxed{\delta = a - y}$$

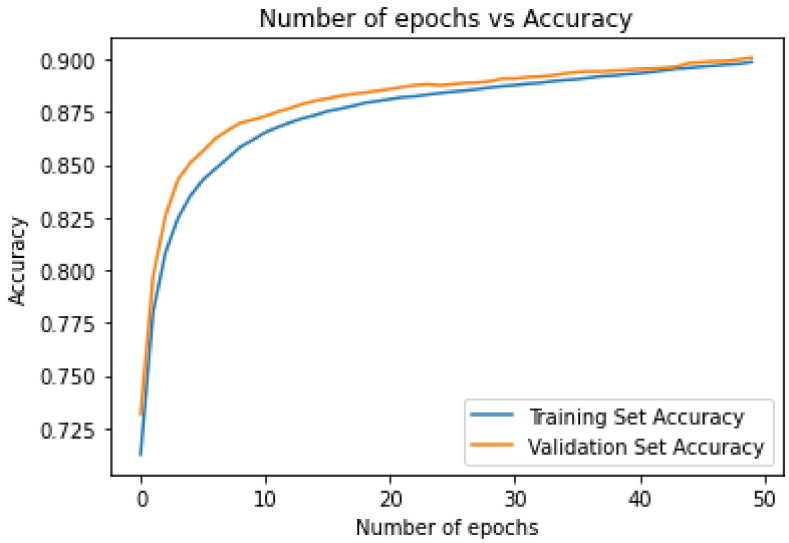


Q3. a) I developed my model using a hidden layer with activation function ReLU and an output layer with softmax. The three learning rates that I used were 0.00001,0.0001 and 0.001 with learning rate decay and a minibatch size of 100.

After training my model for 50 epochs, I observed that the accuracy was about 83.6% for the training set and 85.6% for the validation set with a learning rate of 0.00001.



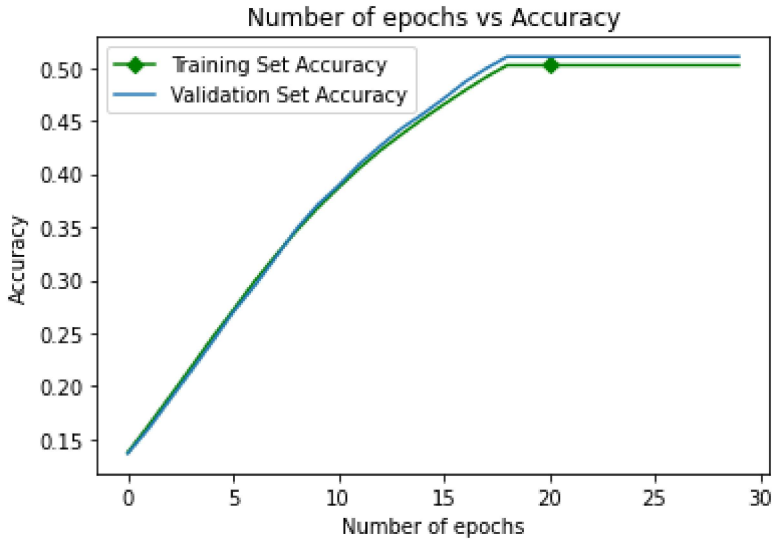
When I used the learning rate of 0.0001, the accuracy obtained for training dataset was about 89.97% and for the validation dataset, the accuracy was 90.07%.



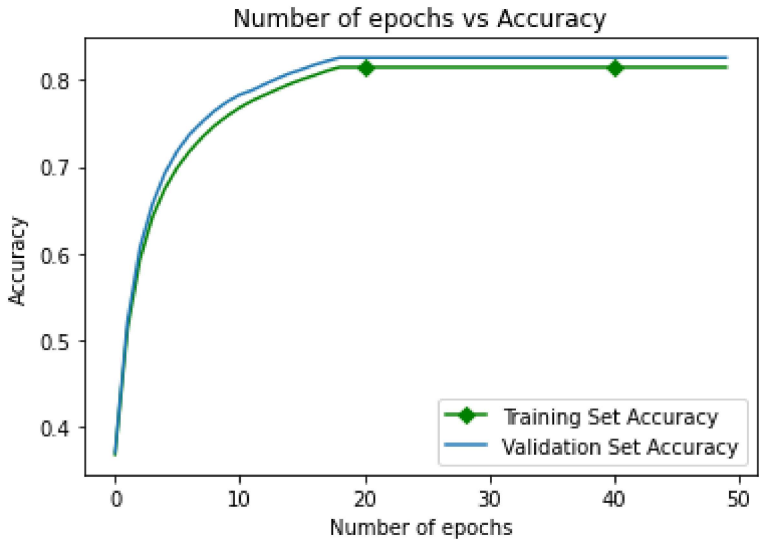
Lastly, when I used the learning rate of 0.001, I obtained an accuarcy of 95.36% on the training dataset and 94.36% on the validation dataset.

b) Similarly, while using tanh as activation function, I used a single hidden layer and an output layer with learning rates 0.00001, 0.0001 and 0.001.

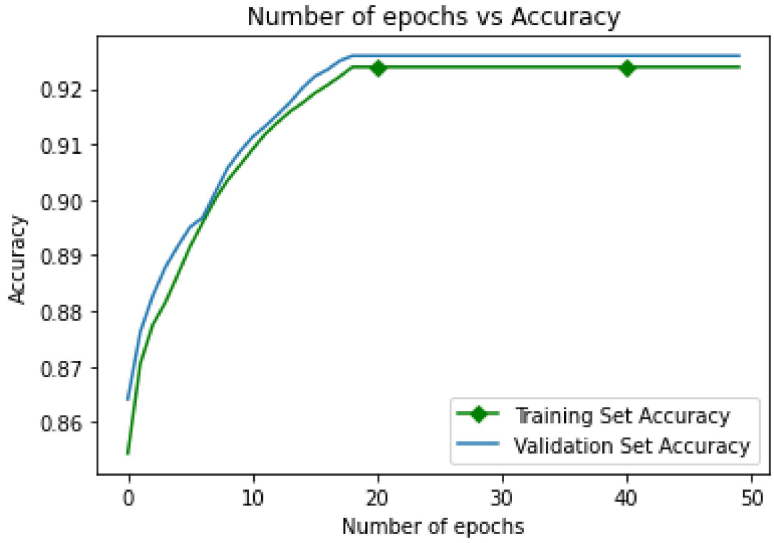
For the learning rate 0.00001, the model obtained an accuracy of 51.04% on the training dataset and 50.04% on the validation dataset.



The learning rate of 0.0001 produced an accuracy of 81% on the training dataset and 82.57% on the validation dataset.



When used a learning rate of 0.001, the model obtained an accuracy of 92.39% on training dataset and 92.59% on validation dataset.



Hence, observing the above values, I used the weights obtained after training the model with a learning rate of 0.001 with ReLU as activation function to test the testing dataset. The testing dataset gave an accuracy of 93.5% when the trained weights were used.

Appendix:

```
#TANH with 0.00001

import h5py
import numpy as np
import matplotlib.pyplot as plt

data1 = h5py.File('mnist_traindata.hdf5', 'r')
```

```
list(data1.keys())

xdata = np.asarray(data1['xdata'])
ydata = np.asarray(data1['ydata'])

#Training Dataset

train_ind = int((5/6)*xdata.shape[0])
train_x = xdata[:train_ind]
train_y = ydata[:train_ind]

#Validation Dataset

valid_x = xdata[train_ind:]
valid_y = ydata[train_ind:]

#importing test dataset
test = h5py.File('mnist_testdata.hdf5', 'r')

xtest = np.asarray(test['xdata'])
ytest = np.asarray(test['ydata'])

#Activation Functions

def softmax(x):
  a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
  return a/np.sum(a).reshape(-1,1)

def linear(x,w,b):
  return np.matmul(x, w) + b.reshape(1,-1)

def relu(x):
  return np.maximum(x,0)

def tanh(x):
  a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
  return a

def d(x):
  return (x > 0).astype(float)

def tanh_d(x):
  a = 1- x**2
  return a

def acc(y_pred, y_true):
  z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
  a = np.equal(y_true,z)
  acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
  return acc/y_true.shape[0]

learning_rate = 0.00001
```

```
learning_rate = 0.00001
mini_batch = 1
n_epochs = 50
n_iters = int(train_x.shape[0]/mini_batch)
loss_train_list = []
loss_test_list = []
loss_valid_list = []

acc_train_list = []
acc_test_list = []
acc_valid_list = []


#Initialising the weights
np.random.seed(seed = 243)
w1 = np.random.normal(0,1, size =(784,100))
b1 = np.random.normal(0,1, size =(100,1))

w2 = np.random.normal(0,1, size = (100,50))
b2 = np.random.normal(0,1, (50,1))

w3 = np.random.normal(0,1, size = (100,10))
b3 = np.random.normal(0,1, size = (10,1))


for i in range(n_epochs):
  iters = 1
  gradient_w3 = np.zeros(w3.shape)
  # gradient_w2 = np.zeros(w2.shape)
  gradient_w1 = np.zeros(w1.shape)

  gradient_b3 = np.zeros(b3.shape)
  # gradient_b2 = np.zeros(b2.shape)
  gradient_b1 = np.zeros(b1.shape)
  # In each epoch store the loss and y_pred
  for j in range(n_iters):
    iters += 1

    #Learning Rate Decay
    if ((i+1)%20==0):
       learning_rate = learning_rate/2

    #Defining the dataset according to the minibatch
    x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
    y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

    #Layer 1
    a1 = tanh(linear(x_train_batch, w1, b1))

    #Layer 2
    # a2 = relu(linear(a1, w2, b2))
```

```
    #Layer 3
    y_train_pred = softmax(linear(a1, w3, b3))

    #Backprop

    #Calculating deltas
    d3 = y_train_pred - y_train_batch
    # d2 = d(a2)*(np.matmul(d3, w3.T))
    d1 = (tanh_d(a1)*(np.matmul(d3,w3.T)))

    #Updating weights and biases
    gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
    gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)

    # gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
    # gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

    gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
    gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

    if iters % 100 == 0:
      w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
      # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
      w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

      b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
      # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
      b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
      # print('b3',b3.T)

      gradient_w3 = np.zeros(w3.shape)
      # gradient_w2 = np.zeros(w2.shape)
      gradient_w1 = np.zeros(w1.shape)

      gradient_b3 = np.zeros(b3.shape)
      # gradient_b2 = np.zeros(b2.shape)
      gradient_b1 = np.zeros(b1.shape)

  #for validation
  y1 = tanh(linear(valid_x, w1, b1))
  # y2 = relu(linear(y1, w2, b2))
  valid_pred = softmax(linear(y1, w3, b3))

  loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
  loss_valid_list.append(loss_valid)
  acc_valid_list.append(acc(valid_pred, valid_y))

  #for training
  yt1 = tanh(linear(train_x, w1, b1))
  # yt2 = relu(linear(yt1, w2, b2))
  train_pred = softmax(linear(yt1, w3, b3))
```

```
    loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
    loss_train_list.append(loss_train)
    acc_train_list.append(acc(train_pred, train_y))

    print("Epoch : ", i)
    print(loss_train, loss_valid)

  markers_on = [20,40]
  plt.plot(acc_train_list,'-gD',markevery=markers_on)
  plt.plot(acc_valid_list)
  plt.xlabel('Number of epochs')
  plt.ylabel('Accuracy')
  plt.title('Number of epochs vs Accuracy')
  plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])

  #TANH with 0.0001

  import h5py
  import numpy as np
  import matplotlib.pyplot as plt

  data1 = h5py.File('mnist_traindata.hdf5', 'r')
  list(data1.keys())

  xdata = np.asarray(data1['xdata'])
  ydata = np.asarray(data1['ydata'])

  #Training Dataset

  train_ind = int((5/6)*xdata.shape[0])
  train_x = xdata[:train_ind]
  train_y = ydata[:train_ind]

  #Validation Dataset

  valid_x = xdata[train_ind:]
  valid_y = ydata[train_ind:]

  #importing test dataset
  test = h5py.File('mnist_testdata.hdf5', 'r')

  xtest = np.asarray(test['xdata'])
  ytest = np.asarray(test['ydata'])

  #Activation Functions

  def softmax(x):
    a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
    return a/np.sum(a).reshape(-1,1)

  def linear(x,w,b):
```

```
    return np.matmul(x, w) + b.reshape(1,-1)

  def relu(x):
    return np.maximum(x,0)

  def tanh(x):
    a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
    return a

  def d(x):
    return (x > 0).astype(float)

  def tanh_d(x):
    a = 1- x**2
    return a

  def acc(y_pred, y_true):
    z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
    a = np.equal(y_true,z)
    acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
    return acc/y_true.shape[0]

  learning_rate = 0.0001
  mini_batch = 1
  n_epochs = 50
  n_iters = int(train_x.shape[0]/mini_batch)
  loss_train_list = []
  loss_test_list = []
  loss_valid_list = []

  acc_train_list = []
  acc_test_list = []
  acc_valid_list = []


  #Initialising the weights
  np.random.seed(seed = 243)
  w1 = np.random.normal(0,1, size =(784,100))
  b1 = np.random.normal(0,1, size =(100,1))

  w2 = np.random.normal(0,1, size = (100,50))
  b2 = np.random.normal(0,1, (50,1))

  w3 = np.random.normal(0,1, size = (100,10))
  b3 = np.random.normal(0,1, size = (10,1))


  for i in range(n_epochs):
    iters = 1
    gradient_w3 = np.zeros(w3.shape)
    # gradient_w2 = np.zeros(w2.shape)
    gradient_w1 = np.zeros(w1.shape)
```

```
    gradient_b3 = np.zeros(b3.shape)
    # gradient_b2 = np.zeros(b2.shape)
    gradient_b1 = np.zeros(b1.shape)
    # In each epoch store the loss and y_pred
    for j in range(n_iters):
      iters += 1

      #Learning Rate Decay
      if ((i+1)%20==0):
         learning_rate = learning_rate/2

      #Defining the dataset according to the minibatch
      x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
      y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

      #Layer 1
      a1 = tanh(linear(x_train_batch, w1, b1))

      #Layer 2
      # a2 = relu(linear(a1, w2, b2))

      #Layer 3
      y_train_pred = softmax(linear(a1, w3, b3))

      #Backprop

      #Calculating deltas
      d3 = y_train_pred - y_train_batch
      # d2 = d(a2)*(np.matmul(d3, w3.T))
      d1 = (tanh_d(a1)*(np.matmul(d3,w3.T)))

      #Updating weights and biases
      gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
      gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)

      # gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
      # gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

      gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
      gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

      if iters % 100 == 0:
        w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
        # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
        w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

        b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
        # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
        b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
        # print('b3',b3.T)
```

```
        gradient_w3 = np.zeros(w3.shape)
        # gradient_w2 = np.zeros(w2.shape)
        gradient_w1 = np.zeros(w1.shape)

        gradient_b3 = np.zeros(b3.shape)
        # gradient_b2 = np.zeros(b2.shape)
        gradient_b1 = np.zeros(b1.shape)

    #for validation
    y1 = tanh(linear(valid_x, w1, b1))
    # y2 = relu(linear(y1, w2, b2))
    valid_pred = softmax(linear(y1, w3, b3))

    loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
    loss_valid_list.append(loss_valid)
    acc_valid_list.append(acc(valid_pred, valid_y))

    #for training
    yt1 = tanh(linear(train_x, w1, b1))
    # yt2 = relu(linear(yt1, w2, b2))
    train_pred = softmax(linear(yt1, w3, b3))

    loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
    loss_train_list.append(loss_train)
    acc_train_list.append(acc(train_pred, train_y))

    print("Epoch : ", i)
    print(loss_train, loss_valid)

markers_on = [20,40]
plt.plot(acc_train_list,'-gD',markevery=markers_on)
plt.plot(acc_valid_list)
plt.xlabel('Number of epochs')
plt.ylabel('Accuracy')
plt.title('Number of epochs vs Accuracy')
plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])

#TANH with 0.001

import h5py
import numpy as np
import matplotlib.pyplot as plt

data1 = h5py.File('mnist_traindata.hdf5', 'r')
list(data1.keys())

xdata = np.asarray(data1['xdata'])
ydata = np.asarray(data1['ydata'])

#Training Dataset
```

```
train_ind = int((5/6)*xdata.shape[0])
train_x = xdata[:train_ind]
train_y = ydata[:train_ind]

#Validation Dataset

valid_x = xdata[train_ind:]
valid_y = ydata[train_ind:]

#importing test dataset
test = h5py.File('mnist_testdata.hdf5', 'r')

xtest = np.asarray(test['xdata'])
ytest = np.asarray(test['ydata'])

#Activation Functions

def softmax(x):
  a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
  return a/np.sum(a).reshape(-1,1)

def linear(x,w,b):
  return np.matmul(x, w) + b.reshape(1,-1)

def relu(x):
  return np.maximum(x,0)

def tanh(x):
  a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
  return a

def d(x):
  return (x > 0).astype(float)

def tanh_d(x):
  a = 1- x**2
  return a

def acc(y_pred, y_true):
  z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
  a = np.equal(y_true,z)
  acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
  return acc/y_true.shape[0]

learning_rate = 0.001
mini_batch = 1
n_epochs = 50
n_iters = int(train_x.shape[0]/mini_batch)
loss_train_list = []
loss_test_list = []
loss_valid_list = []
```

```
acc_train_list = []
acc_test_list = []
acc_valid_list = []


#Initialising the weights
np.random.seed(seed = 243)
w1 = np.random.normal(0,1, size =(784,100))
b1 = np.random.normal(0,1, size =(100,1))

w2 = np.random.normal(0,1, size = (100,50))
b2 = np.random.normal(0,1, (50,1))

w3 = np.random.normal(0,1, size = (100,10))
b3 = np.random.normal(0,1, size = (10,1))


for i in range(n_epochs):
  iters = 1
  gradient_w3 = np.zeros(w3.shape)
  # gradient_w2 = np.zeros(w2.shape)
  gradient_w1 = np.zeros(w1.shape)

  gradient_b3 = np.zeros(b3.shape)
  # gradient_b2 = np.zeros(b2.shape)
  gradient_b1 = np.zeros(b1.shape)
  # In each epoch store the loss and y_pred
  for j in range(n_iters):
    iters += 1

    #Learning Rate Decay
    if ((i+1)%20==0):
       learning_rate = learning_rate/2

    #Defining the dataset according to the minibatch
    x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
    y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

    #Layer 1
    a1 = tanh(linear(x_train_batch, w1, b1))

    #Layer 2
    # a2 = relu(linear(a1, w2, b2))

    #Layer 3
    y_train_pred = softmax(linear(a1, w3, b3))

    #Backprop

    #Calculating deltas
    d3 = y_train_pred - y_train_batch
```

```
    # d2 = d(a2)*(np.matmul(d3, w3.T))
    d1 = (tanh_d(a1)*(np.matmul(d3,w3.T)))

    #Updating weights and biases
    gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
    gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)

    # gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
    # gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

    gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
    gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

    if iters % 100 == 0:
      w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
      # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
      w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

      b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
      # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
      b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
      # print('b3',b3.T)

      gradient_w3 = np.zeros(w3.shape)
      # gradient_w2 = np.zeros(w2.shape)
      gradient_w1 = np.zeros(w1.shape)

      gradient_b3 = np.zeros(b3.shape)
      # gradient_b2 = np.zeros(b2.shape)
      gradient_b1 = np.zeros(b1.shape)

  #for validation
  y1 = tanh(linear(valid_x, w1, b1))
  # y2 = relu(linear(y1, w2, b2))
  valid_pred = softmax(linear(y1, w3, b3))

  loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
  loss_valid_list.append(loss_valid)
  acc_valid_list.append(acc(valid_pred, valid_y))

  #for training
  yt1 = tanh(linear(train_x, w1, b1))
  # yt2 = relu(linear(yt1, w2, b2))
  train_pred = softmax(linear(yt1, w3, b3))

  loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
  loss_train_list.append(loss_train)
  acc_train_list.append(acc(train_pred, train_y))

  print("Epoch : ", i)
  print(loss_train, loss_valid)
```

```
markers_on = [20,40]
plt.plot(acc_train_list,'-gD',markevery=markers_on)
plt.plot(acc_valid_list)
plt.xlabel('Number of epochs')
plt.ylabel('Accuracy')
plt.title('Number of epochs vs Accuracy')
plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])

#ReLU with 0.00001

import h5py
import numpy as np
import matplotlib.pyplot as plt

data1 = h5py.File('mnist_traindata.hdf5', 'r')
list(data1.keys())

xdata = np.asarray(data1['xdata'])
ydata = np.asarray(data1['ydata'])

#Training Dataset

train_ind = int((5/6)*xdata.shape[0])
train_x = xdata[:train_ind]
train_y = ydata[:train_ind]

#Validation Dataset

valid_x = xdata[train_ind:]
valid_y = ydata[train_ind:]

#importing test dataset
test = h5py.File('mnist_testdata.hdf5', 'r')

xtest = np.asarray(test['xdata'])
ytest = np.asarray(test['ydata'])

#Activation Functions

def softmax(x):
  a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
  return a/np.sum(a).reshape(-1,1)

def linear(x,w,b):
  return np.matmul(x, w) + b.reshape(1,-1)

def relu(x):
  return np.maximum(x,0)

def tanh(x):
  a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
```

```
    return a

  def d(x):
    return (x > 0).astype(float)

  def tanh_d(x):
    a = 1- x**2
    return a

  def acc(y_pred, y_true):
    z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
    a = np.equal(y_true,z)
    acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
    return acc/y_true.shape[0]

  learning_rate = 0.00001
  mini_batch = 1
  n_epochs = 50
  n_iters = int(train_x.shape[0]/mini_batch)
  loss_train_list = []
  loss_test_list = []
  loss_valid_list = []

  acc_train_list = []
  acc_test_list = []
  acc_valid_list = []


  #Initialising the weights
  np.random.seed(seed = 243)
  w1 = np.random.normal(0,1, size =(784,100))
  b1 = np.random.normal(0,1, size =(100,1))

  w2 = np.random.normal(0,1, size = (100,50))
  b2 = np.random.normal(0,1, (50,1))

  w3 = np.random.normal(0,1, size = (100,10))
  b3 = np.random.normal(0,1, size = (10,1))


  for i in range(n_epochs):
    iters = 1
    gradient_w3 = np.zeros(w3.shape)
    # gradient_w2 = np.zeros(w2.shape)
    gradient_w1 = np.zeros(w1.shape)

    gradient_b3 = np.zeros(b3.shape)
    # gradient_b2 = np.zeros(b2.shape)
    gradient_b1 = np.zeros(b1.shape)
    # In each epoch store the loss and y_pred
    for j in range(n_iters):
      iters += 1
```

```
#Learning Rate Decay
if ((i+1)%20==0):
   learning_rate = learning_rate/2

#Defining the dataset according to the minibatch
x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

#Layer 1
a1 = relu(linear(x_train_batch, w1, b1))

#Layer 2
# a2 = relu(linear(a1, w2, b2))

#Layer 3
y_train_pred = softmax(linear(a1, w3, b3))

#Backprop

#Calculating deltas
d3 = y_train_pred - y_train_batch
# d2 = d(a2)*(np.matmul(d3, w3.T))
d1 = (d(a1)*(np.matmul(d3,w3.T)))

#Updating weights and biases
gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)

# gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
# gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

if iters % 100 == 0:
  w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
  # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
  w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

  b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
  # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
  b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
  # print('b3',b3.T)

  gradient_w3 = np.zeros(w3.shape)
  # gradient_w2 = np.zeros(w2.shape)
  gradient_w1 = np.zeros(w1.shape)

  gradient_b3 = np.zeros(b3.shape)
  # gradient_b2 = np.zeros(b2.shape)
```

```
        gradient_b1 = np.zeros(b1.shape)

    #for validation
    y1 = relu(linear(valid_x, w1, b1))
    # y2 = relu(linear(y1, w2, b2))
    valid_pred = softmax(linear(y1, w3, b3))

    loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
    loss_valid_list.append(loss_valid)
    acc_valid_list.append(acc(valid_pred, valid_y))

    #for training
    yt1 = relu(linear(train_x, w1, b1))
    # yt2 = relu(linear(yt1, w2, b2))
    train_pred = softmax(linear(yt1, w3, b3))

    loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
    loss_train_list.append(loss_train)
    acc_train_list.append(acc(train_pred, train_y))

    print("Epoch : ", i)
    print(loss_train, loss_valid)

markers_on = [20,40]
plt.plot(acc_train_list,'-gD',markevery=markers_on)
plt.plot(acc_valid_list)
plt.xlabel('Number of epochs')
plt.ylabel('Accuracy')
plt.title('Number of epochs vs Accuracy')
plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])

#ReLU with 0.0001

import h5py
import numpy as np
import matplotlib.pyplot as plt

data1 = h5py.File('mnist_traindata.hdf5', 'r')
list(data1.keys())

xdata = np.asarray(data1['xdata'])
ydata = np.asarray(data1['ydata'])

#Training Dataset

train_ind = int((5/6)*xdata.shape[0])
train_x = xdata[:train_ind]
train_y = ydata[:train_ind]

#Validation Dataset

valid_x = xdata[train_ind:]
```

```
valid_y = ydata[train_ind:]

#importing test dataset
test = h5py.File('mnist_testdata.hdf5', 'r')

xtest = np.asarray(test['xdata'])
ytest = np.asarray(test['ydata'])

#Activation Functions

def softmax(x):
  a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
  return a/np.sum(a).reshape(-1,1)

def linear(x,w,b):
  return np.matmul(x, w) + b.reshape(1,-1)

def relu(x):
  return np.maximum(x,0)

def tanh(x):
  a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
  return a

def d(x):
  return (x > 0).astype(float)

def tanh_d(x):
  a = 1- x**2
  return a

def acc(y_pred, y_true):
  z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
  a = np.equal(y_true,z)
  acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
  return acc/y_true.shape[0]

learning_rate = 0.0001
mini_batch = 1
n_epochs = 50
n_iters = int(train_x.shape[0]/mini_batch)
loss_train_list = []
loss_test_list = []
loss_valid_list = []

acc_train_list = []
acc_test_list = []
acc_valid_list = []


#Initialising the weights
np.random.seed(seed = 243)
```

```
np.random.seed(seed = 243)
w1 = np.random.normal(0,1, size =(784,100))
b1 = np.random.normal(0,1, size =(100,1))

w2 = np.random.normal(0,1, size = (100,50))
b2 = np.random.normal(0,1, (50,1))

w3 = np.random.normal(0,1, size = (100,10))
b3 = np.random.normal(0,1, size = (10,1))


for i in range(n_epochs):
  iters = 1
  gradient_w3 = np.zeros(w3.shape)
  # gradient_w2 = np.zeros(w2.shape)
  gradient_w1 = np.zeros(w1.shape)

  gradient_b3 = np.zeros(b3.shape)
  # gradient_b2 = np.zeros(b2.shape)
  gradient_b1 = np.zeros(b1.shape)
  # In each epoch store the loss and y_pred
  for j in range(n_iters):
    iters += 1

    #Learning Rate Decay
    if ((i+1)%20==0):
       learning_rate = learning_rate/2

    #Defining the dataset according to the minibatch
    x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
    y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

    #Layer 1
    a1 = relu(linear(x_train_batch, w1, b1))

    #Layer 2
    # a2 = relu(linear(a1, w2, b2))

    #Layer 3
    y_train_pred = softmax(linear(a1, w3, b3))

    #Backprop

    #Calculating deltas
    d3 = y_train_pred - y_train_batch
    # d2 = d(a2)*(np.matmul(d3, w3.T))
    d1 = (d(a1)*(np.matmul(d3,w3.T)))

    #Updating weights and biases
    gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
    gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)
```

```
      # gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
      # gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

      gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
      gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

      if iters % 100 == 0:
        w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
        # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
        w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

        b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
        # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
        b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
        # print('b3',b3.T)

        gradient_w3 = np.zeros(w3.shape)
        # gradient_w2 = np.zeros(w2.shape)
        gradient_w1 = np.zeros(w1.shape)

        gradient_b3 = np.zeros(b3.shape)
        # gradient_b2 = np.zeros(b2.shape)
        gradient_b1 = np.zeros(b1.shape)

    #for validation
    y1 = relu(linear(valid_x, w1, b1))
    # y2 = relu(linear(y1, w2, b2))
    valid_pred = softmax(linear(y1, w3, b3))

    loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
    loss_valid_list.append(loss_valid)
    acc_valid_list.append(acc(valid_pred, valid_y))

    #for training
    yt1 = relu(linear(train_x, w1, b1))
    # yt2 = relu(linear(yt1, w2, b2))
    train_pred = softmax(linear(yt1, w3, b3))

    loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
    loss_train_list.append(loss_train)
    acc_train_list.append(acc(train_pred, train_y))

    print("Epoch : ", i)
    print(loss_train, loss_valid)

  markers_on = [20,40]
  plt.plot(acc_train_list,'-gD',markevery=markers_on)
  plt.plot(acc_valid_list)
  plt.xlabel('Number of epochs')
  plt.ylabel('Accuracy')
  plt.title('Number of epochs vs Accuracy')
  plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])
```

```
plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])

#ReLU with 0.001

import h5py
import numpy as np
import matplotlib.pyplot as plt

data1 = h5py.File('mnist_traindata.hdf5', 'r')
list(data1.keys())

xdata = np.asarray(data1['xdata'])
ydata = np.asarray(data1['ydata'])

#Training Dataset

train_ind = int((5/6)*xdata.shape[0])
train_x = xdata[:train_ind]
train_y = ydata[:train_ind]

#Validation Dataset

valid_x = xdata[train_ind:]
valid_y = ydata[train_ind:]

#importing test dataset
test = h5py.File('mnist_testdata.hdf5', 'r')

xtest = np.asarray(test['xdata'])
ytest = np.asarray(test['ydata'])

#Activation Functions

def softmax(x):
  a = np.exp(x - np.max(x, axis = 1).reshape(-1,1))
  return a/np.sum(a).reshape(-1,1)

def linear(x,w,b):
  return np.matmul(x, w) + b.reshape(1,-1)

def relu(x):
  return np.maximum(x,0)

def tanh(x):
  a = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))
  return a

def d(x):
  return (x > 0).astype(float)

def tanh_d(x):
  a = 1- x**2
```

```
    return a

  def acc(y_pred, y_true):
    z = y_pred >= np.max(y_pred, axis = 1).reshape(-1,1)
    a = np.equal(y_true,z)
    acc = np.sum(np.sum(a, axis = 1)==y_pred.shape[1])
    return acc/y_true.shape[0]

  learning_rate = 0.001
  mini_batch = 1
  n_epochs = 50
  n_iters = int(train_x.shape[0]/mini_batch)
  loss_train_list = []
  loss_test_list = []
  loss_valid_list = []

  acc_train_list = []
  acc_test_list = []
  acc_valid_list = []


  #Initialising the weights
  np.random.seed(seed = 243)
  w1 = np.random.normal(0,1, size =(784,100))
  b1 = np.random.normal(0,1, size =(100,1))

  w2 = np.random.normal(0,1, size = (100,50))
  b2 = np.random.normal(0,1, (50,1))

  w3 = np.random.normal(0,1, size = (100,10))
  b3 = np.random.normal(0,1, size = (10,1))


  for i in range(n_epochs):
    iters = 1
    gradient_w3 = np.zeros(w3.shape)
    # gradient_w2 = np.zeros(w2.shape)
    gradient_w1 = np.zeros(w1.shape)

    gradient_b3 = np.zeros(b3.shape)
    # gradient_b2 = np.zeros(b2.shape)
    gradient_b1 = np.zeros(b1.shape)
    # In each epoch store the loss and y_pred
    for j in range(n_iters):
      iters += 1

      #Learning Rate Decay
      if ((i+1)%20==0):
         learning_rate = learning_rate/2

      #Defining the dataset according to the minibatch
      x_train_batch = train_x[j*mini_batch: (j+1)*mini_batch]
```

```
    y_train_batch = train_y[j*mini_batch: (j+1)*mini_batch]

    #Layer 1
    a1 = relu(linear(x_train_batch, w1, b1))

    #Layer 2
    # a2 = relu(linear(a1, w2, b2))

    #Layer 3
    y_train_pred = softmax(linear(a1, w3, b3))

    #Backprop

    #Calculating deltas
    d3 = y_train_pred - y_train_batch
    # d2 = d(a2)*(np.matmul(d3, w3.T))
    d1 = (d(a1)*(np.matmul(d3,w3.T)))

    #Updating weights and biases
    gradient_w3 = gradient_w3 + np.matmul(a1.T, d3)
    gradient_b3 = gradient_b3 + np.sum(d3, axis = 0).reshape(-1,1)

    # gradient_w2 = gradient_w2 + np.matmul(a1.T, d2)
    # gradient_b2 = gradient_b2 + np.sum(d2, axis = 0).reshape(-1,1)

    gradient_w1 = gradient_w1 + np.matmul(x_train_batch.T, d1)
    gradient_b1 = gradient_b1 + np.sum(d1, axis = 0).reshape(-1,1)

    if iters % 100 == 0:
      w3 = w3 - learning_rate * (1/mini_batch) * gradient_w3
      # w2 = w2 - learning_rate * (1/mini_batch) * gradient_w2
      w1 = w1 - learning_rate * (1/mini_batch) * gradient_w1

      b3 = b3 - learning_rate * (1/mini_batch) * gradient_b3
      # b2 = b2 - learning_rate * (1/mini_batch) * gradient_b2
      b1 = b1 - learning_rate * (1/mini_batch) * gradient_b1
      # print('b3',b3.T)

      gradient_w3 = np.zeros(w3.shape)
      # gradient_w2 = np.zeros(w2.shape)
      gradient_w1 = np.zeros(w1.shape)

      gradient_b3 = np.zeros(b3.shape)
      # gradient_b2 = np.zeros(b2.shape)
      gradient_b1 = np.zeros(b1.shape)

  #for validation
  y1 = relu(linear(valid_x, w1, b1))
  # y2 = relu(linear(y1, w2, b2))
  valid_pred = softmax(linear(y1, w3, b3))

  loss_valid = (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-20))
```

```
    loss_valid = - (1/10000)*np.sum(valid_y*np.log(valid_pred + 1e-39))
    loss_valid_list.append(loss_valid)
    acc_valid_list.append(acc(valid_pred, valid_y))

    #for training
    yt1 = relu(linear(train_x, w1, b1))
    # yt2 = relu(linear(yt1, w2, b2))
    train_pred = softmax(linear(yt1, w3, b3))

    loss_train = - (1/60000)*np.sum(train_y*np.log(train_pred  + 1e-39))
    loss_train_list.append(loss_train)
    acc_train_list.append(acc(train_pred, train_y))

    print("Epoch : ", i)
    print(loss_train, loss_valid)

markers_on = [20,40]
plt.plot(acc_train_list,'-gD',markevery=markers_on)
plt.plot(acc_valid_list)
plt.xlabel('Number of epochs')
plt.ylabel('Accuracy')
plt.title('Number of epochs vs Accuracy')
plt.legend(['Training Set Accuracy', 'Validation Set Accuracy'])
```