ECE 421 Assignment 1 Report

Yiren Zhao
1005092527

Part 1: Logistic Regression with Numpy

1. **Loss Function and Gradient**
2. **Gradient Descent Implementation**
   (code is at the end of this report or see another attach file starter.py)

3. Tuning the Learning Rate

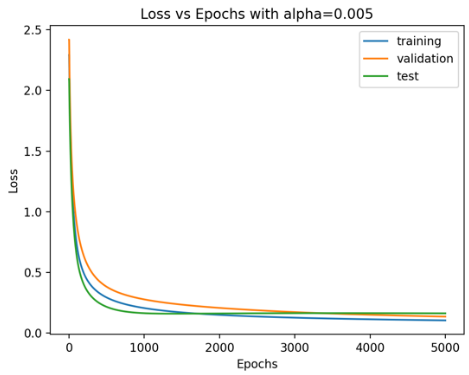

Figure 1: loss vs epochs with α = 0.005

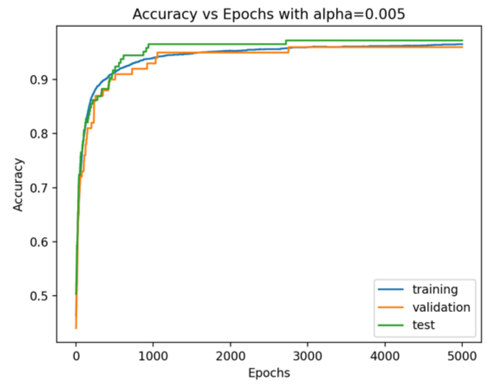

Figure 2: accuracy vs epochs with α = 0.005

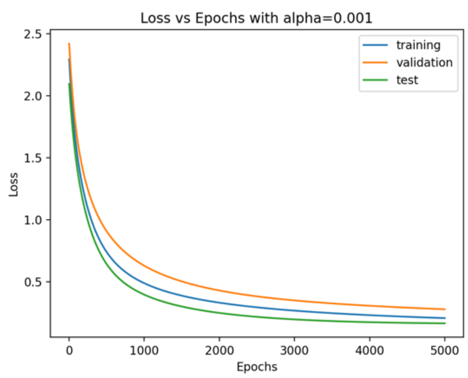

Figure 3: loss vs epochs with α = 0.001

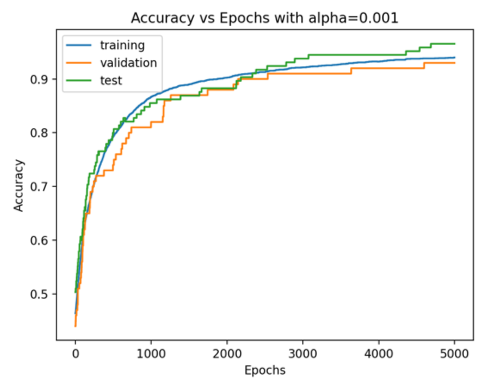

Figure 4: accuracy vs epochs with α = 0.001

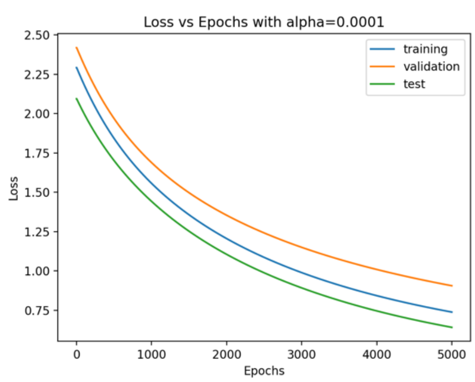

Figure 5: loss vs epochs with α = 0.0001

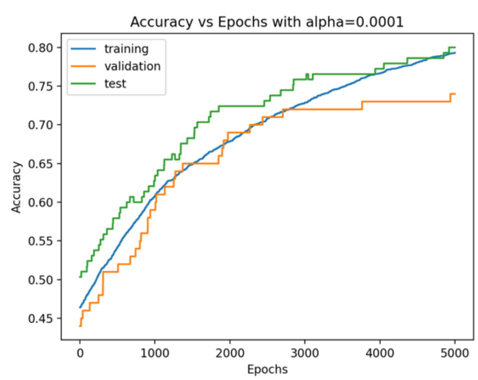

Figure 6: accuracy vs epochs with α = 0.0001

Table1: using learning rate α = {0.005, 0.001, 0.0001} and keep λ = 0

	α = 0.005	α = 0.001	α = 0.0001
Training Accuracy	0.965429	0.94	0.793143
Validation Accuracy	0.96	0.93	0.74
Test Accuracy	0.972414	0.965517	0.8

Figure 1, 3 and 5 shows the training and validation loss vs. number of passed epochs with α=0.005, 0.001, 0.00001 respectively and λ = 0. All three loss plots shows a decreasing trend and a convex shape, as passed more epochs, the loss decreases speed becomes much slowly. The plot with smaller learning rate is less convex and takes more epochs to converge than the larger one. It is obvious that Figure 5 with α=0.0001 need more time to approach the global minimum value than other two loss plots and has a relatively large gap between the training and validation curve. Curves in figure 1 with α = 0.005 fit the best, since the gap between training and validation curve is smaller than figure 3 with α = 0.001, and almost overlap for most of the time.

Figure 2, 4, 6 and table1 shows the training, validation and test accuracy with different learning rate. Since validation data is a separate data set not used for training, we use validation data to track validation accuracy and make decision about model hyperparameters (The model is already trained by training data many times, the training accuracy is unreliable and may causes overfit) The accuracy for α = 0.0001 is the lowest, which is 0.74, around 0.15 lower than other two learning rate. Learning rate with 0.005 has highest validation accuracy, which is 0.96, slightly high than figure 2 with α=0.001.

Figure 1 with 0.005 learning rate is the best fitted model so far with the highest accuracy and converges fastest. Therefore, I choose α = 0.005 as the best learning rate, and the validation accuracy for this learning rate is 0.96, the training accuracy is 0.965429.

4. Generalization

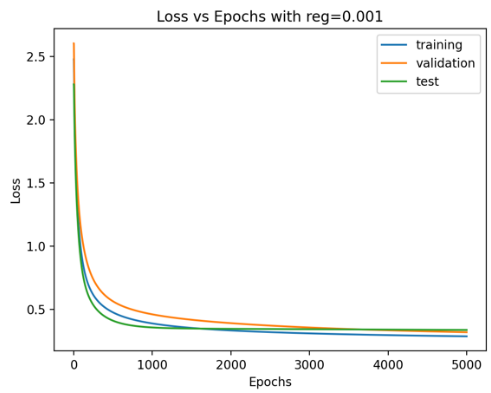

Figure 7: loss vs epochs with reg = 0.001

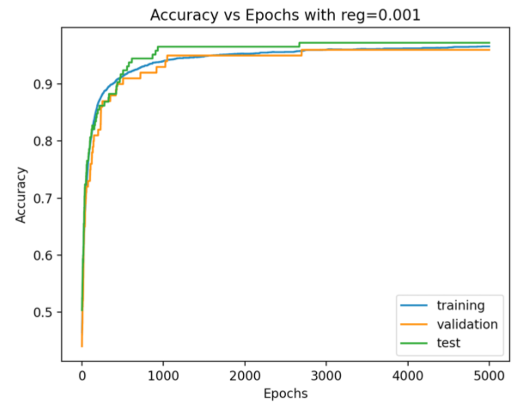

Figure 8: accuracy vs epochs with reg=0.001

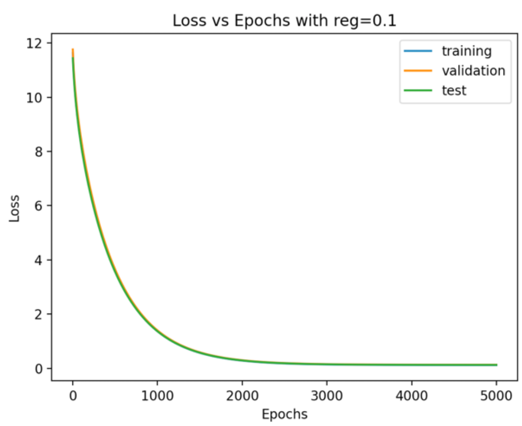

Figure 9: loss vs epochs with reg = 0.1

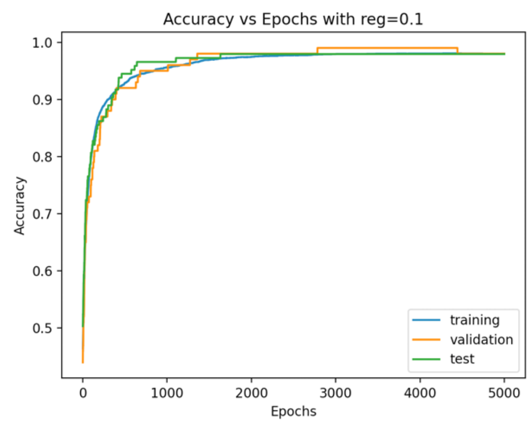

Figure 10: accuracy vs epochs with reg=0.1

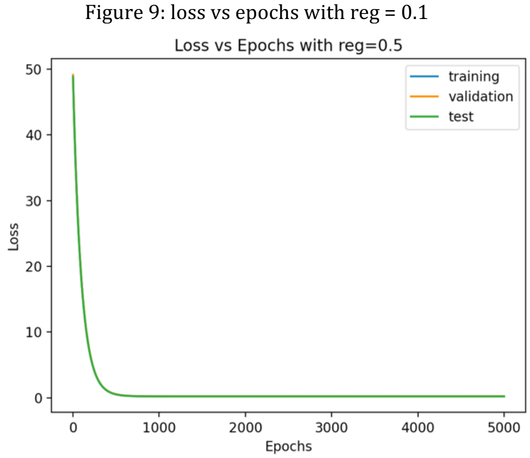

Figure 11: loss vs epochs with reg = 0.5

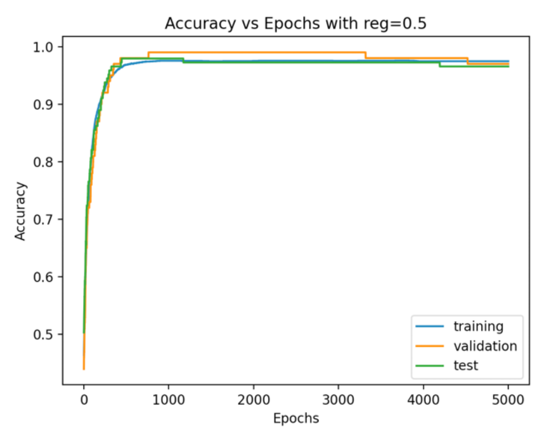

Figure 12: accuracy vs epochs with reg=0.5

Table2: regularization parameter λ = {0.001, 0.1, 0.5} and keep α = 0.005

	λ = 0.001	λ = 0.1	λ = 0.5
Training Accuracy	0.9657142857142857	0.9808571428571429	0.9754285714285714
Validation Accuracy	0.96	0.99	0.99
Test Accuracy	0.9724137931034482	0.9793103448275862	0.9724137931034482

Figure 7, 9 and 11 shows the training and validation loss vs. number of passed epochs with λ = 0.001, 0.1, 0.5 respectively and α = 0.005.

There is a gap between curves in figure 7 with λ = 0.001, then the gap becomes smaller in figure 9 with λ = 0.1, and in figure 11 with λ = 0.5, the curves are overlapping. We can see as λ increases, the model is better fitted to prevent overfitting, also, the loss curve will converge faster to approach zero.

Figure 11 with 0.5 regularization converges fastest and has the lowest validation loss among three different value of λ. Therefore, I choose regularization parameter λ = 0.5 as the best parameter and the training accuracy for this model is 0.9754285714285714, the validation accuracy is 0.99.

Part 2 Logistic Regression in TensorFlow

1. Building the Computational Graph
(code is at the end of this report or see another attach file starter.py)

2. Implementing Stochastic Gradient Descent

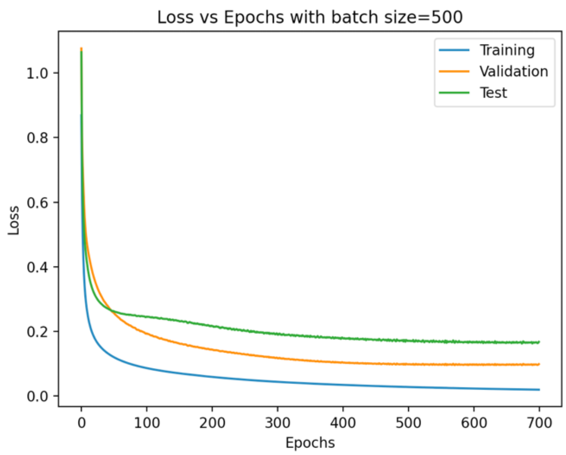

Figure 13: loss vs epochs with batch size=500

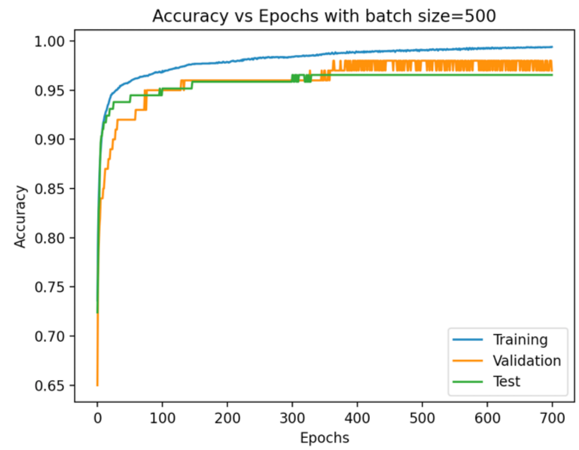

Figure 14: accuracy vs epochs with batch size=500

Table3: using a minibatch size of 500 optimizing over 700 epochs, set $\lambda = 0$ and $\alpha = 0.001$

	Accuracy	Loss
Training	0.994	0.018850597
Validation	0.97	0.098015755
Test	0.9655172413793104	0.16710205

3. Batch Size Investigation

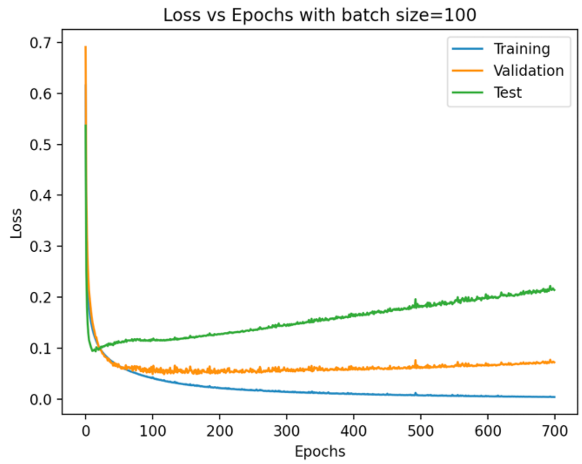

Figure 15: loss vs epochs with batch size=100

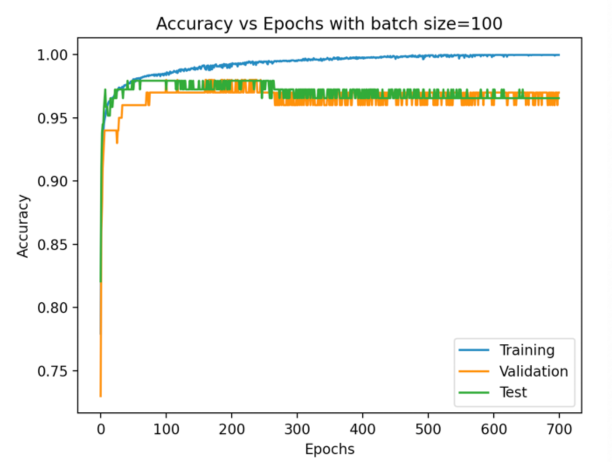

Figure 16: accuracy vs epochs with batch size=100

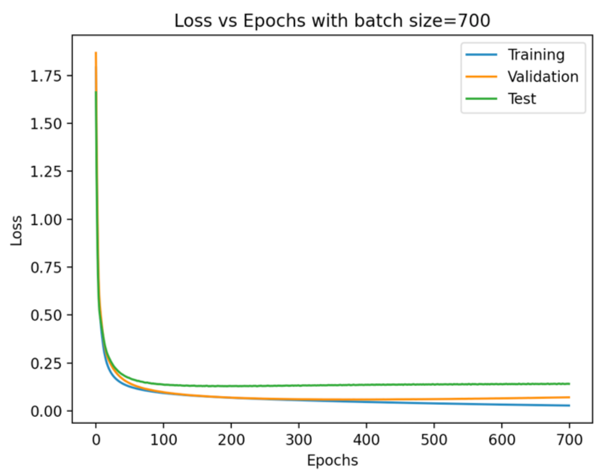

Figure 17: loss vs epochs with batch size=700

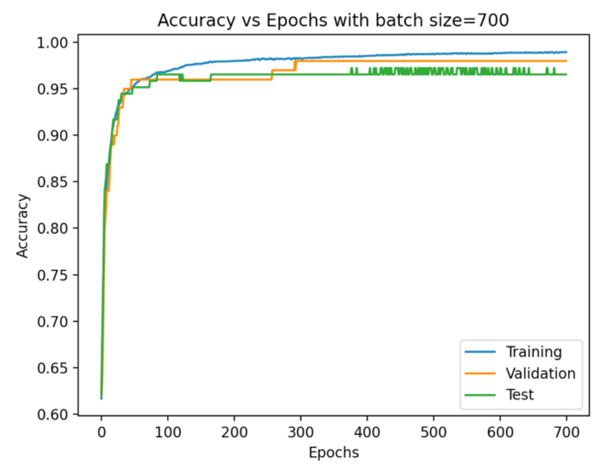

Figure 18: accuracy vs epochs with batch size=700

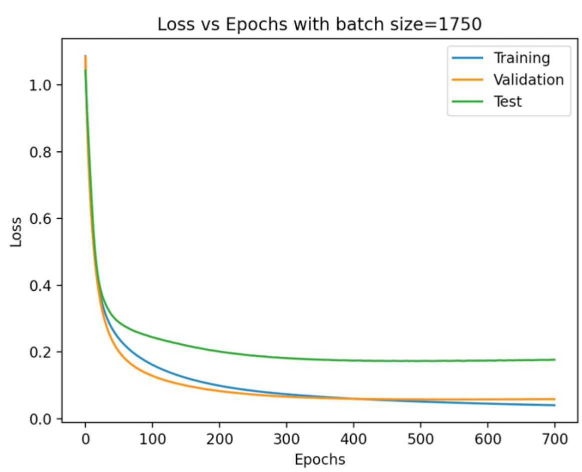

Figure 19: loss vs epochs with batch size=1750

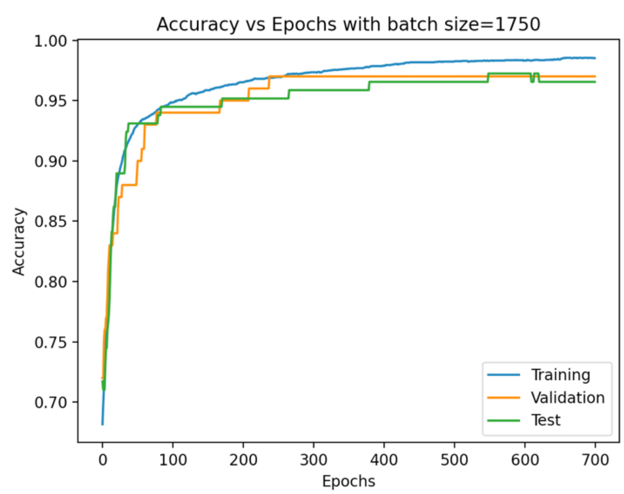

Figure 20: accuracy vs epochs with batch size=1750

Table4: using batch sizes of B = {100,700,1750}, set $\lambda = 0$ and $\alpha = 0.001$

	Batch size=100	Batch size=700	Batch size=1750
Training Accuracy	0.999623	0.992867	0.978286
Validation Accuracy	0.98	0.97	0.97

Figure 15, 17 and 19 shows the training and validation loss vs. number of passed epochs with batch size = 100, 700 and 1750 respectively and epoch = 500. We can see with a large batch size, the loss curves are less convex. Larger batch size is supposed be has lower accuracy. But the table 4 indicates as the batch size increases, the accuracy do not has significant change, batch size of 1750 has slightly lower accuracy. That might because the model is already optimal earlier before 700 epochs. But since larger batch size has larger gradient steps, it has the advantage of escaping the local minimum and reduce the noise, while the smaller batch sizes provide a regularization effect. We can see the loss plot for batch size 1750 is smoother than other two.

4. Hyperparameter Investigation

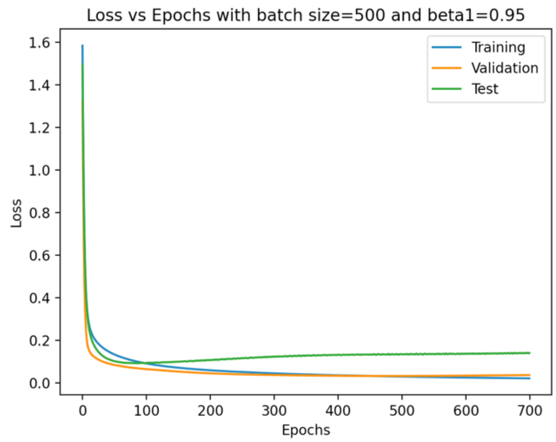

Figure 21: loss vs epochs with beta1=0.95

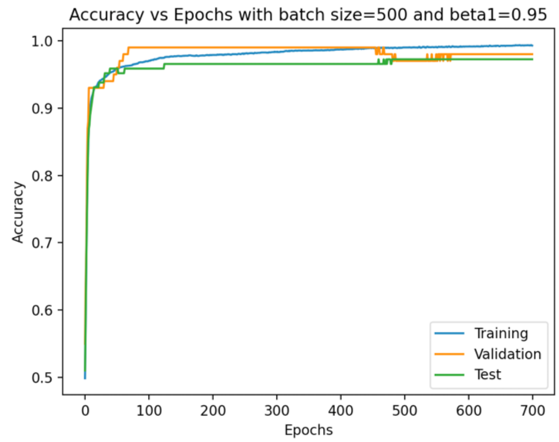

Figure 22: accuracy vs epochs with beta1=0.95

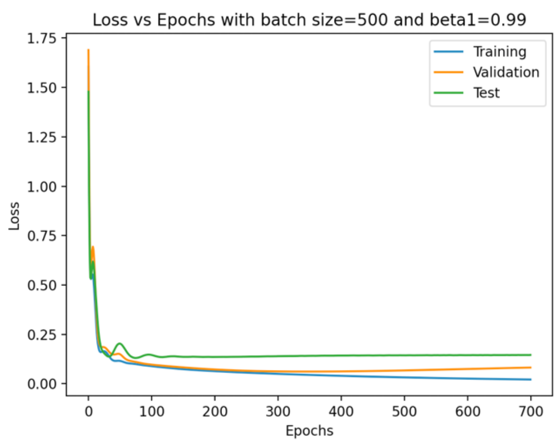

Figure 23: loss vs epochs with beta1=0.99

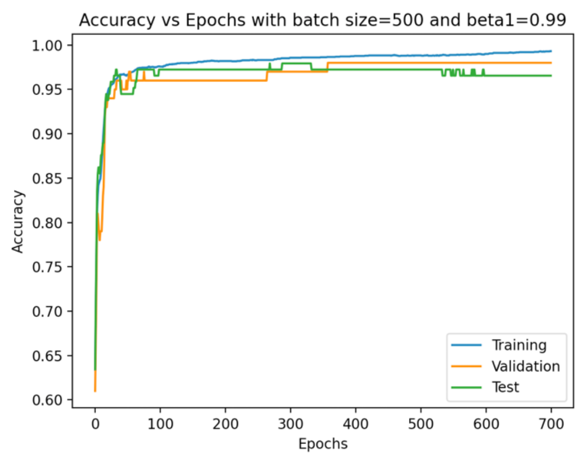

Figure 24: accuracy vs epochs with beta1=0.99

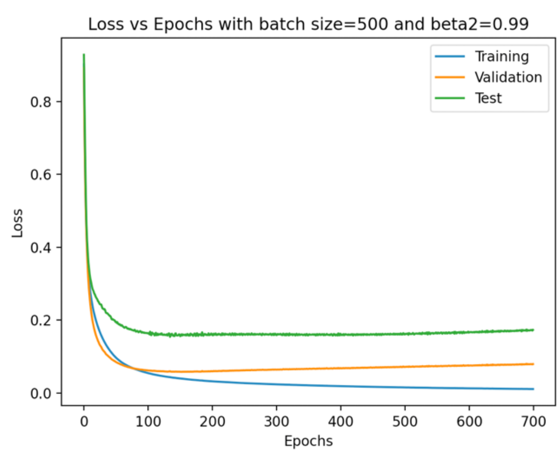

Figure 25: loss vs epochs with beta2=0.99

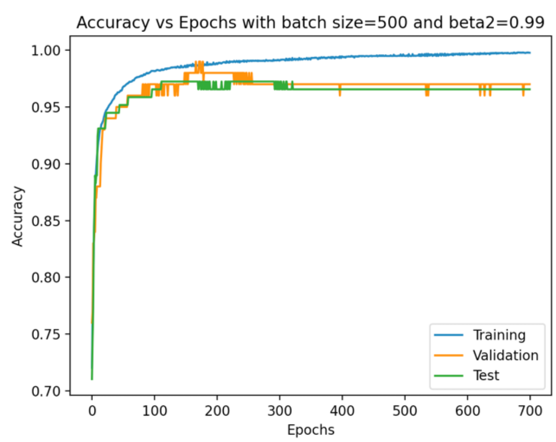

Figure 26: accuracy vs epochs with beta2=0.99

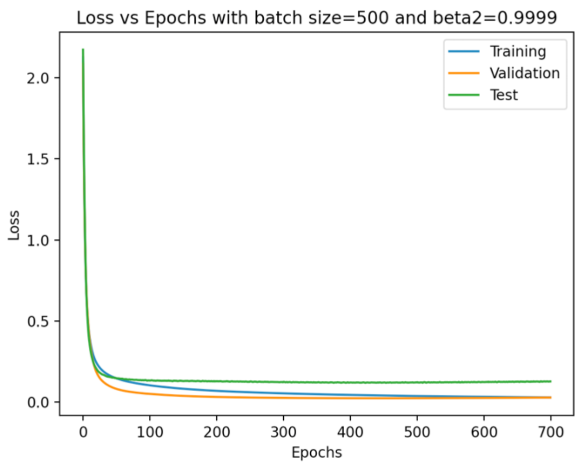

Figure 27: loss vs epochs with beta2=0.9999

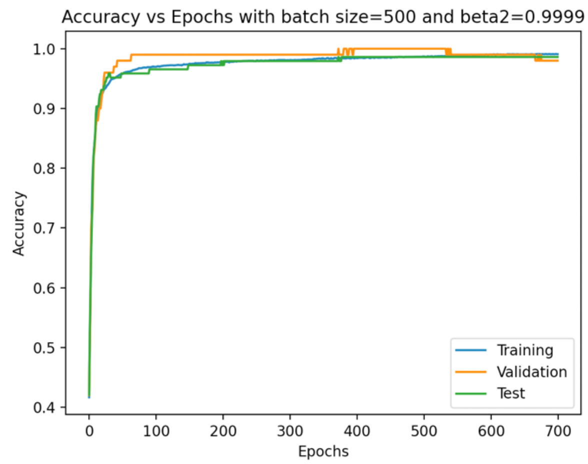

Figure 28: accuracy vs epochs with beta2=0.9999

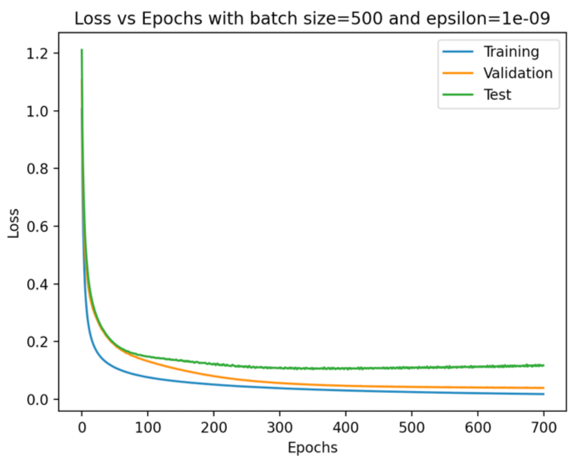

Figure 29: loss vs epochs with epsilon=1e-09

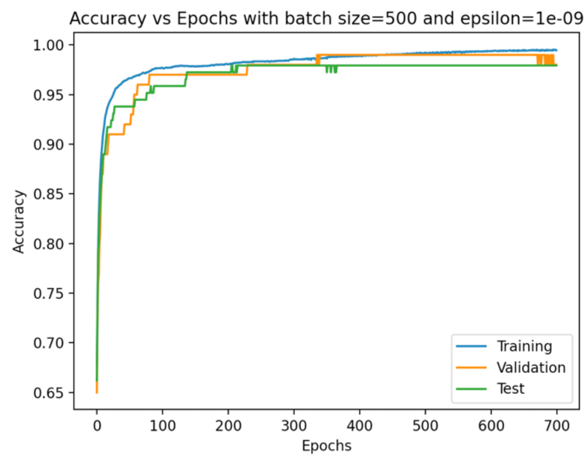

Figure 30: accuracy vs epochs with epsilon=1e-09

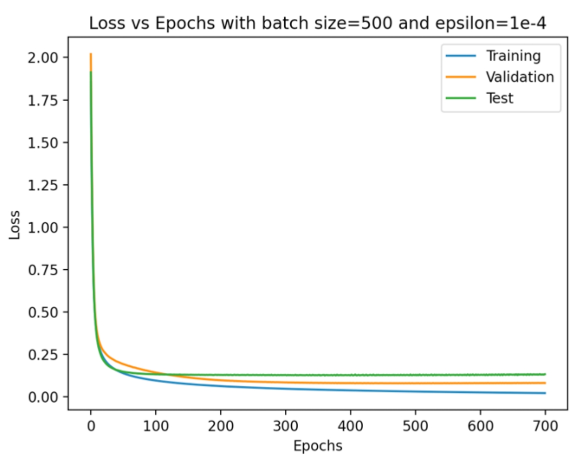

Figure 31: loss vs epochs with epsilon=1e-4

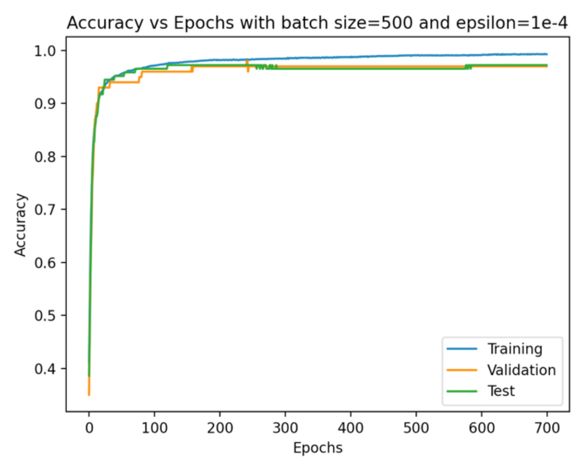

Figure 32: accuracy vs epochs with epsilon=1e-4

Table5: different value of beta1, which keep beta2=default and epsilon=default

	bata1=0.95	bata1=0.99
Training Accuracy	0.9938571438571429	0.9921428571428571
Validation Accuracy	0.98	0.97
Test Accuracy	0.9724137931034482	0.9655172413793104

(a) Bata1
Beta1 represents the first moment and is used to compute running averages of gradient. Higher beta1 will results in lower accuracy. By comparing loss plot figure 21 with beta1=0.95 and figure 23 with beta1=0.99, we found the larger the beta1, the less the noise, the curves are smoother. From table 5, both bata1 values have very high and similarly accuracy, but beta1=0.95 has slightly higher accuracy than beta1=0.99. I would choose beta1=0.95 with 0.9938571438571429 training accuracy and 0.98 validation accuracy.

Table6: different value of beta2, with beta1=default and epsilon=default

	bata2=0.99	bata2=0.9999
Training Accuracy	0.9977142857142857	0.9908571428571429
Validation Accuracy	0.98	0.97
Test Accuracy	0.9862068965517241	0.9655172413793104

(b) Bata2
Beta2 represents the second moment and is used to compute running averages of square of gradient. Higher beta2 will results in lower accuracy and higher loss. By comparing loss plot figure 25 with beta2=0.99 and figure 27 with beta2=0.9999, we found that the larger beta2, the less the noise, the curves are smoother. From table 6, both bata2 values have very high and similarly accuracy, but beta2=0.99 has slightly higher accuracy than beta2=0.9999, which means a slightly. I would choose beta2=0.99 with 0.9977142857142857 training accuracy and 0.98 validation accuracy.

Table7: different value of epsilon, which keep beta1=default and beta2=default

	epsilon =1e-09	epsilon =1e-4
Training Accuracy	0.9945714285714286	0.9928571428571429
Validation Accuracy	0.98	0.97
Test Accuracy	0.9793103448275862	0.9724137931034482

(c) Epsilon
Epsilon is used to improve numerical stability and prevent dividing by zero error while updating the Adam step when the gradient is almost zero. The epsilon value is supposed to be small, but having a small epsilon will make large weight updates and thus fast the training progress. However, in our plot in figure 29 and 31, I found the epsilon with 1e-4

converges faster table 7 indicate both epsilon values have very high and similarly accuracy. I would choose epsilon=1e-09 and the training accuracy is 0. 9945714285714286, and the validation accuracy is 0.98.

5. Comparison against Batch GD

The overall performance of SGD algorithm with Adam is better than batch gradient descent algorithm. Two method both reach high accuracy that approach to 1, with SGD algorithm has slightly higher. The SGD algorithm with Adam passed less epochs than gradient descent algorithm to reach loss of 0. This shows SGD algorithm with Adam converges much faster than Adam for loss and need less time to minimize the error. Also Adam optimizer has beta1 and beta2 parameters that can speed up the gradient decent and can help the model escape the local minimums.

Code

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

def loadData():
    with np.load('notMNIST.npz') as dataset:
        Data, Target = dataset['images'], dataset['labels']
        posClass = 2
        negClass = 9
        dataIndx = (Target==posClass) + (Target==negClass)
        Data = Data[dataIndx]/255.
        Target = Target[dataIndx].reshape(-1, 1)
        Target[Target==posClass] = 1
        Target[Target==negClass] = 0
        np.random.seed(421)
        randIndx = np.arange(len(Data))
        np.random.shuffle(randIndx)
        Data, Target = Data[randIndx], Target[randIndx]
        trainData, trainTarget = Data[:3500], Target[:3500]
        validData, validTarget = Data[3500:3600], Target[3500:3600]
        testData, testTarget = Data[3600:], Target[3600:]
    return trainData, validData, testData, trainTarget, validTarget, testTarget



def loss(W, b, x, y, reg):
    # Your implementation here
    N = np.shape(y)[0]
    z = np.matmul(x,W) + b
    y_hat= 1.0/(1.0+np.exp(-z))

    #loss_ce= (1/N)*(-np.sum(y*np.log(y_hat)+(1-y)*np.log(1-y_hat)))
    #loss_reg=(reg/2)*(np.linalg.norm(W)**2)
    #loss_reg=(reg/2)*(np.sum(W*W))
    #loss_total=loss_ce+loss_reg
    loss_total=(np.sum(-(y*np.log(y_hat)+(1-y)*np.log(1-y_hat))))/(np.shape(y)[0]) +
reg/2*np.sum(W*W)
    return loss_total


def grad_loss(W, b, x, y, reg):
    # Your implementation here
    N = np.shape(y)[0]

    z = np.matmul(x,W) + b
    y_hat= 1.0/(1.0+np.exp(-1.0*z))
```

```
    grad_loss_w=np.matmul(np.transpose(x), (y_hat - y))/(np.shape(y)[0]) + reg*W
    grad_loss_b=(np.sum(y_hat - y)) / N
    return grad_loss_w, grad_loss_b


def grad_descent(W, b, x, y, alpha, epochs, reg, error_tol, vali, vali_target, test,
test_target):
    # Your implementation here

    train_loss_array=[]
    train_acc_array=[]
    train_loss_array.append(loss(W, b, x, y, reg))
    train_acc_array.append(accuracy(W, b, x, y))

    validation_loss_array=[]
    validation_acc_array=[]
    validation_loss_array.append(loss(W, b, vali, vali_target, reg))
    validation_acc_array.append(accuracy(W, b, vali, vali_target))

    test_loss_array=[]
    test_acc_array=[]
    test_loss_array.append(loss(W, b, test, test_target, reg))
    test_acc_array.append(accuracy(W, b, test, test_target))
    for epoch in range(epochs):
        grad_loss_w, grad_loss_b = grad_loss(W, b, x, y, reg)
        W_new = W - alpha * grad_loss_w
        b_new = b - alpha * grad_loss_b

        #calculate loss
        train_loss_array.append(loss(W_new, b_new, x, y, reg))
        #add accuracy to array
        train_acc_array.append(accuracy(W_new, b_new, x, y))


        #calculate loss
        validation_loss_array.append(loss(W_new, b_new, vali, vali_target, reg))
        #add accuracy to array
        validation_acc_array.append(accuracy(W_new, b_new, vali, vali_target))

        #calculate loss
        test_loss_array.append(loss(W_new, b_new, test, test_target, reg))
        #add accuracy to array
        test_acc_array.append(accuracy(W_new, b_new, test, test_target))

        if(np.linalg.norm(W_new - W) < error_tol):
            break
```

```
        W = W_new
        b = b_new


    print("When alpha = ", alpha,"and reg=", reg)
    print("Training Accuracy", train_acc_array[-1])
    print("Validation Accuracy", validation_acc_array[-1])
    print("Test Accuracy", test_acc_array[-1])

    return W_new, b_new, train_loss_array, train_acc_array, validation_loss_array,
validation_acc_array, test_loss_array, test_acc_array


def accuracy(W, b, x, y):
    N = np.shape(y)[0]
    z = np.matmul(x,W) + b
    y_hat= 1.0/(1.0+np.exp(-z))

    accuracy = np.sum( (y_hat >= 0.5) ==y ) / N
    return accuracy

def buildGraph(beta1, beta2, epsilon,learning_rate ):
    W = tf.Variable(tf.truncated_normal([784, 1],mean=0.0, stddev=0.5,
dtype=tf.float32))
    b = tf.Variable(tf.zeros(1))

    x = tf.placeholder(tf.float32, [None, 784])
    y = tf.placeholder(tf.float32, [None, 1])
    reg = tf.placeholder(tf.float32)
    tf.set_random_seed(421)

    logits = (tf.matmul(x, W) + b)
    loss_total=tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=y,
logits=logits)) + reg * tf.nn.l2_loss(W)
    if (beta1!=0 and beta2==0 and epsilon==0):
        optimizer = tf.train.AdamOptimizer(learning_rate=0.001,
beta1=beta1).minimize(loss_total)
    elif(beta1==0 and beta2!=0 and epsilon==0):
        optimizer = tf.train.AdamOptimizer(learning_rate=0.001,
beta2=beta2).minimize(loss_total)
    elif(beta1==0 and beta2==0 and epsilon!=0):
        optimizer = tf.train.AdamOptimizer(learning_rate=0.001,
epsilon=epsilon).minimize(loss_total)
    elif(beta1==0 and beta2==0 and epsilon==0):
        optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss_total)
    else:
        optimizer = tf.train.AdamOptimizer(learning_rate=0.001, beta1=beta1,
beta2=beta2, epsilon=epsilon).minimize(loss_total)
```

```
    return x, y, W, b, reg, loss_total, optimizer



def SGD(batchSize, trainData, trainTarget, beta1, beta2, epsilon,
learning_rate,epochs):
    N=3500
    x, y, W, b, reg, loss_total, optimizer = buildGraph(beta1, beta2,
epsilon,learning_rate)
    loop = N // batchSize

    train_loss_array=[]
    train_acc_array=[]
    validation_loss_array=[]
    validation_acc_array=[]
    test_loss_array=[]
    test_acc_array=[]

    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        for i in range(epochs):
            index = np.arange(N)
            np.random.shuffle(index)
            trainData = trainData[index]
            trainTarget = trainTarget[index]

            for j in range(loop):
                batch_x = trainData[j*batchSize:(j+1)*batchSize, :]
                batch_y = trainTarget[j*batchSize:(j+1)*batchSize, :]
                _, train_W, train_b = sess.run([optimizer, W, b], feed_dict={x:
batch_x, y: batch_y, reg: 0})


            train_acc_array.append(accuracy(train_W, train_b, trainData, trainTarget))
            train_loss_array.append(sess.run(loss_total, feed_dict={x: trainData, y:
trainTarget, reg: 0}))

            validation_acc_array.append(accuracy(train_W, train_b, validData,
validTarget))
            validation_loss_array.append(sess.run(loss_total, feed_dict={x: validData,
y: validTarget, reg: 0}))


            test_acc_array.append(accuracy(train_W, train_b, testData, testTarget))
            test_loss_array.append(sess.run(loss_total, feed_dict={x: testData, y:
testTarget, reg: 0}))
```

```
    print('Batch size:',batchSize)
    if (beta1!=0 and beta2==0 and epsilon==0):
        print('beta1:',beta1)
    elif(beta1==0 and beta2!=0 and epsilon==0):
        print('beta2:',beta2)
    elif(beta1==0 and beta2==0 and epsilon!=0):
        print('epsilon:',epsilon)

    print("Trainning Loss",train_loss_array[-1])
    print("Validation Loss",validation_loss_array[-1])
    print("Test Loss",test_loss_array[-1])
    print("Training Accuracy", train_acc_array[-1])
    print("Validation Accuracy", validation_acc_array[-1])
    print("Test Accuracy", test_acc_array[-1])


    return train_loss_array, train_acc_array, validation_loss_array,
validation_acc_array, test_loss_array, test_acc_array





#testing
trainData, validData, testData, trainTarget, validTarget, testTarget = loadData()
#Reshape trainData to pass into functions

trainData=trainData.reshape(3500,784)
validData = validData.reshape(100,784)
testData = testData.reshape(145,784)
W= np.random.normal(0,0.5,(trainData.shape[1],1))
b=0

#trainData = trainData.reshape(trainData.shape[0], -1)
#validData = validData.reshape(validData.shape[0], -1)
#testData = testData.reshape(testData.shape[0], -1)

#================================Part 1=====================================
#part 1 Q3 tuning the learning rate
#initialize parameters
epochs = 5000
alpha=[0.005, 0.001, 0.0001]
reg = 0
error_tol = 1e-7


#when alpha=0.005, plot train loss and validation loss--------------------------
```

```
W_train, b_train,train_loss_array1, train_acc_array1,val_loss_array1, val_acc_array1,
test_loss_array1, test_acc_array1  = grad_descent(W, b, trainData, trainTarget,
alpha[0], epochs, reg, error_tol, validData, validTarget,testData, testTarget)

plt.figure(1)
plt.plot(train_loss_array1, label='training')
plt.plot(val_loss_array1, label='validation')
plt.plot(test_loss_array1, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with alpha=0.005")
plt.legend(loc='best')


#when alpha=0.005, plot train accuracy and validation accuracy------------------
plt.figure(2)
plt.plot(train_acc_array1, label='training')
plt.plot(val_acc_array1, label='validation')
plt.plot(test_acc_array1, label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with alpha=0.005")
plt.legend(loc='best')


#-------------------------------------------------------------------------------

#when alpha=0.001, plot train loss and validation loss--------------------------
W_train, b_train,train_loss_array2, train_acc_array2,val_loss_array2, val_acc_array2,
test_loss_array2, test_acc_array2  = grad_descent(W, b, trainData, trainTarget,
alpha[1], epochs, reg, error_tol, validData, validTarget,testData, testTarget)

plt.figure(3)
plt.plot(train_loss_array2, label='training')
plt.plot(val_loss_array2, label='validation')
plt.plot(test_loss_array2, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with alpha=0.001")
plt.legend(loc='best')


#when alpha=0.001, plot train accuracy and validation accuracy------------------
plt.figure(4)
plt.plot(train_acc_array2, label='training')
plt.plot(val_acc_array2, label='validation')
plt.plot(test_acc_array2, label='test')
plt.xlabel("Epochs")
```

```
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with alpha=0.001")
plt.legend(loc='best')

#-------------------------------------------------------------------------------
#when alpha=0.0001, plot train accuracy and validation accuracy------------------
W_train, b_train,train_loss_array3, train_acc_array3,val_loss_array3, val_acc_array3,
test_loss_array3, test_acc_array3  = grad_descent(W, b, trainData, trainTarget,
alpha[2], epochs, reg, error_tol, validData, validTarget,testData, testTarget)

plt.figure(5)
plt.plot(train_loss_array3, label='training')
plt.plot(val_loss_array3, label='validation')
plt.plot(test_loss_array3, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with alpha=0.0001")
plt.legend(loc='best')


#when alpha=0.0001, plot train accuracy and validation accuracy------------------
plt.figure(6)
plt.plot(train_acc_array3, label='training')
plt.plot(val_acc_array3, label='validation')
plt.plot(test_acc_array3, label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with alpha=0.0001")
plt.legend(loc='best')

#=====================================================================================
======================
#part 1 Q4 Generalization
epochs = 5000
reg = [0.001, 0.1, 0.5]
alpha=0.005
error_tol = 1e-7

#-------------------------------------------------------------------------------
#when reg=0.001, plot train loss and validation loss--------------------------

W_train, b_train,train_loss_array4, train_acc_array4,val_loss_array4, val_acc_array4,
test_loss_array4, test_acc_array4  = grad_descent(W, b, trainData, trainTarget, alpha,
epochs, reg[0], error_tol, validData, validTarget,testData, testTarget)

plt.figure(7)
plt.plot(train_loss_array4, label='training')
plt.plot(val_loss_array4, label='validation')
```

```
plt.plot(test_loss_array4, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with reg=0.001")
plt.legend(loc='best')


#when reg=0.001, plot train accuracy and validation accuracy------------------
plt.figure(8)
plt.plot(train_acc_array4, label='training')
plt.plot(val_acc_array4, label='validation')
plt.plot(test_acc_array4, label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with reg=0.001")
plt.legend(loc='best')

#-------------------------------------------------------------------------------

#when reg=0.1, plot train loss and validation loss---------------------------
W_train, b_train,train_loss_array5, train_acc_array5,val_loss_array5, val_acc_array5,
test_loss_array5, test_acc_array5  = grad_descent(W, b, trainData, trainTarget, alpha,
epochs, reg[1], error_tol, validData, validTarget,testData, testTarget)

plt.figure(9)
plt.plot(train_loss_array5, label='training')
plt.plot(val_loss_array5, label='validation')
plt.plot(test_loss_array5, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with reg=0.1")
plt.legend(loc='best')


#when reg=0.1, plot train accuracy and validation accuracy------------------
plt.figure(10)
plt.plot(train_acc_array5, label='training')
plt.plot(val_acc_array5, label='validation')
plt.plot(test_acc_array5, label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with reg=0.1")
plt.legend(loc='best')

#-------------------------------------------------------------------------------
#when reg=0.5, plot train accuracy and validation accuracy------------------
```

```
W_train, b_train,train_loss_array6, train_acc_array6,val_loss_array6, val_acc_array6,
test_loss_array6, test_acc_array6  = grad_descent(W, b, trainData, trainTarget, alpha,
epochs, reg[2], error_tol, validData, validTarget,testData, testTarget)

plt.figure(11)
plt.plot(train_loss_array6, label='training')
plt.plot(val_loss_array6, label='validation')
plt.plot(test_loss_array6, label='test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with reg=0.5")
plt.legend(loc='best')


#when reg=0.5, plot train accuracy and validation accuracy------------------
plt.figure(12)
plt.plot(train_acc_array6, label='training')
plt.plot(val_acc_array6, label='validation')
plt.plot(test_acc_array6, label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with reg=0.5")
plt.legend(loc='best')

#===========================Part2================================================
epochs=700
beta1 =[0,0.95, 0.99]
beta2 = [0,0.99, 0.9999]
epsilon = [0,1e-09, 1e-4]
batchSize=[500, 100, 700, 1750]
learning_rate=0.001

#Q2 minibatch_size=500, epochs=700, reg=0, alpha=0.001 ----------------------
train_loss_array7, train_acc_array7,validation_loss_array7, validation_acc_array7,
test_loss_array7, test_acc_array7=SGD(batchSize[0], trainData, trainTarget, beta1[0],
beta2[0], epsilon[0], learning_rate, epochs)

plt.figure(13)
plt.plot(train_loss_array7, label='Training')
plt.plot(validation_loss_array7, label='Validation')
plt.plot(test_loss_array7, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500")
plt.legend(loc='best')

plt.figure(14)
plt.plot(train_acc_array7, label='Training')
```

```
plt.plot(validation_acc_array7, label='Validation')
plt.plot(test_acc_array7, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500")
plt.legend(loc='best')

#Q3 epochs=700, reg=0, alpha=0.001 ----------------------
#when minibatch_size=100 -----------------------
train_loss_array8, train_acc_array8,validation_loss_array8, validation_acc_array8,
test_loss_array8, test_acc_array8=SGD(batchSize[1], trainData, trainTarget, beta1[0],
beta2[0], epsilon[0], learning_rate, epochs)

plt.figure(15)
plt.plot(train_loss_array8, label='Training')
plt.plot(validation_loss_array8, label='Validation')
plt.plot(test_loss_array8, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=100")
plt.legend(loc='best')

plt.figure(16)
plt.plot(train_acc_array8, label='Training')
plt.plot(validation_acc_array8, label='Validation')
plt.plot(test_acc_array8, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=100")
plt.legend(loc='best')

train_loss_array9, train_acc_array9,validation_loss_array9, validation_acc_array9,
test_loss_array9, test_acc_array9=SGD(batchSize[2], trainData, trainTarget, beta1[0],
beta2[0], epsilon[0], learning_rate, epochs)

#when minibatch_size=700 -----------------------
plt.figure(17)
plt.plot(train_loss_array9, label='Training')
plt.plot(validation_loss_array9, label='Validation')
plt.plot(test_loss_array9, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=700")
plt.legend(loc='best')

plt.figure(18)
plt.plot(train_acc_array9, label='Training')
plt.plot(validation_acc_array9, label='Validation')
```

```python
plt.plot(test_acc_array9, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=700")
plt.legend(loc='best')


#when minibatch_size=1750 -----------------------
train_loss_array10, train_acc_array10,validation_loss_array10, validation_acc_array10,
test_loss_array10, test_acc_array10=SGD(batchSize[3], trainData, trainTarget,
beta1[0], beta2[0], epsilon[0], learning_rate, epochs)
plt.figure(19)
plt.plot(train_loss_array10, label='Training')
plt.plot(validation_loss_array10, label='Validation')
plt.plot(test_loss_array10, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=1750")
plt.legend(loc='best')

plt.figure(20)
plt.plot(train_acc_array10, label='Training')
plt.plot(validation_acc_array10, label='Validation')
plt.plot(test_acc_array10, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=1750")
plt.legend(loc='best')


#Q4 epochs=700, reg=0, alpha=0.001, batchSize=500 ----------------------
#when beta1=0.95, beta2=default, epsilon=default-------------
train_loss_array11, train_acc_array11,validation_loss_array11, validation_acc_array11,
test_loss_array11, test_acc_array11=SGD(batchSize[0], trainData, trainTarget,
beta1[1], beta2[0], epsilon[0], learning_rate, epochs)
plt.figure(21)
plt.plot(train_loss_array11, label='Training')
plt.plot(validation_loss_array11, label='Validation')
plt.plot(test_loss_array11, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and beta1=0.95")
plt.legend(loc='best')

plt.figure(22)
plt.plot(train_acc_array11, label='Training')
plt.plot(validation_acc_array11, label='Validation')
plt.plot(test_acc_array11, label='Test')
```

```
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and beta1=0.95")
plt.legend(loc='best')

#when beta1=0.99, beta2=default, epsilon=default----------
train_loss_array12, train_acc_array12,validation_loss_array12, validation_acc_array12,
test_loss_array12, test_acc_array12=SGD(batchSize[0], trainData, trainTarget,
beta1[2], beta2[0], epsilon[0], learning_rate, epochs)
plt.figure(23)
plt.plot(train_loss_array12, label='Training')
plt.plot(validation_loss_array12, label='Validation')
plt.plot(test_loss_array12, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and beta1=0.99")
plt.legend(loc='best')

plt.figure(24)
plt.plot(train_acc_array12, label='Training')
plt.plot(validation_acc_array12, label='Validation')
plt.plot(test_acc_array12, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and beta1=0.99")
plt.legend(loc='best')


#when beta1=default, beta2=0.99, epsilon=default----------
train_loss_array13, train_acc_array13,validation_loss_array13, validation_acc_array13,
test_loss_array13, test_acc_array13=SGD(batchSize[0], trainData, trainTarget,
beta1[0], beta2[1], epsilon[0], learning_rate, epochs)
plt.figure(25)
plt.plot(train_loss_array13, label='Training')
plt.plot(validation_loss_array13, label='Validation')
plt.plot(test_loss_array13, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and beta2=0.99")
plt.legend(loc='best')

plt.figure(26)
plt.plot(train_acc_array13, label='Training')
plt.plot(validation_acc_array13, label='Validation')
plt.plot(test_acc_array13, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and beta2=0.99")
```

```
plt.legend(loc='best')


#when beta1=default, beta2=0.9999, epsilon=default----------
train_loss_array14, train_acc_array14,validation_loss_array14, validation_acc_array14,
test_loss_array14, test_acc_array14=SGD(batchSize[0], trainData, trainTarget,
beta1[0], beta2[2], epsilon[0], learning_rate, epochs)
plt.figure(27)
plt.plot(train_loss_array14, label='Training')
plt.plot(validation_loss_array14, label='Validation')
plt.plot(test_loss_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and beta2=0.9999")
plt.legend(loc='best')

plt.figure(28)
plt.plot(train_acc_array14, label='Training')
plt.plot(validation_acc_array14, label='Validation')
plt.plot(test_acc_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and beta2=0.9999")
plt.legend(loc='best')

#when beta1=default, beta2=default, epsilon=1e-09----------
train_loss_array14, train_acc_array14,validation_loss_array14, validation_acc_array14,
test_loss_array14, test_acc_array14=SGD(batchSize[0], trainData, trainTarget,
beta1[0], beta2[0], epsilon[1], learning_rate, epochs)
plt.figure(29)
plt.plot(train_loss_array14, label='Training')
plt.plot(validation_loss_array14, label='Validation')
plt.plot(test_loss_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and epsilon=1e-09")
plt.legend(loc='best')

plt.figure(30)
plt.plot(train_acc_array14, label='Training')
plt.plot(validation_acc_array14, label='Validation')
plt.plot(test_acc_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and epsilon=1e-09")
plt.legend(loc='best')

#when beta1=default, beta2=default, epsilon=1e-4----------
```

```
train_loss_array14, train_acc_array14,validation_loss_array14, validation_acc_array14,
test_loss_array14, test_acc_array14=SGD(batchSize[0], trainData, trainTarget,
beta1[0], beta2[0], epsilon[2], learning_rate, epochs)
plt.figure(31)
plt.plot(train_loss_array14, label='Training')
plt.plot(validation_loss_array14, label='Validation')
plt.plot(test_loss_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs with batch size=500 and epsilon=1e-4")
plt.legend(loc='best')

plt.figure(32)
plt.plot(train_acc_array14, label='Training')
plt.plot(validation_acc_array14, label='Validation')
plt.plot(test_acc_array14, label='Test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Epochs with batch size=500 and epsilon=1e-4")
plt.legend(loc='best')

plt.show()
```