a)

Graphical user interface

Description automatically generated

b)

The final test accuracy is 0.995. There is overfitting as test loss converges to a larger value than train loss beyond approximately 200 epochs; this suggests that the classifier has been overtrained, becoming optimised specifically for the training samples and failing to fully generalise to the test set. Stopping training early (after ~200 epochs) would reduce overfitting but compromise performance, since the test accuracy at this point is only 0.95 (see zoomed plot below). However, training could be stopped at a later point (e.g. ~1000) epochs in an attempt to limit the extent of overfitting.

Chart, histogram

Description automatically generated

c)

A picture containing chart

Description automatically generated

Final training accuracy is unaffected, converging to an accuracy of 1. Final test accuracy is minorly compromised, converging to a value that oscillates between 0.990 and 0.995. Training appears to be much faster than with a smaller initialisation, as loss (and accuracy) converges more rapidly; this is likely because gradients are larger and therefore gradient descent takes larger steps towards the local minimum. Stopping training early (e.g. around 400 epochs) would yield the same final test accuracy but decrease the amount of overfitting.

For the purpose of generalisation however, smaller weight initialisation appears more appropriate due to better test performance. Aside from slight gains in test accuracy, the final test loss is significantly lower (0.02 vs. 0.04) with smaller weight initialisation, implying that the network generalises better.

import scipy.io as sc

import numpy as np

import csv

import matplotlib.pyplot as plt

def runNet(data, n\_h1=100, n\_h2=80, weight\_scale=0.01, save=True):

    n\_inp = data['X\_train'].shape[0]

    w3 = weight\_scale\*np.random.normal(size=(1,n\_h2))

    w2 = weight\_scale\*np.random.normal(size=(n\_h2,n\_h1))

    w1 = weight\_scale\*np.random.normal(size=(n\_h1,n\_inp))

    L\_train = np.zeros(max\_itr)

    L\_test = np.zeros(max\_itr)

    acc\_train = np.zeros(max\_itr)

    acc\_test = np.zeros(max\_itr)

    print('Training network...')

    for i in range(max\_itr):

        z1 = w1 @ data['X\_train']

        h1 = np.maximum(z1, 0)

        z2 = w2 @ h1

        h2 = np.maximum(z2, 0)

        yh = w3 @ h2

        \_z1 = w1 @ data['X\_test']

        \_h1 = np.maximum(\_z1, 0)

        \_z2 = w2 @ \_h1

        \_h2 = np.maximum(\_z2, 0)

        \_yh = w3 @ \_h2

        L\_train[i] = 0.5 \* np.linalg.norm(data['y\_train'] - yh)\*\*2

        acc\_train[i] = np.mean(np.sign(yh) == np.sign(data['y\_train']))

        L\_test[i] = 0.5 \* np.linalg.norm(data['y\_test'] - \_yh)\*\*2

        acc\_test[i] = np.mean(np.sign(\_yh) == np.sign(data['y\_test']))

        if i % 100 == 0:

            print('Epoch:\t', i)

            print('Training loss:\t', L\_train[i])

            print('Training acc:\t', acc\_train[i])

            print('Test loss:\t', L\_test[i])

            print('Test acc:\t', acc\_test[i])

        e = data['y\_train'] - yh

        d3 = e

        d2 = w3.T @ d3 \* (z2 > 0)

        d1 = w2.T @ d2 \* (z1 > 0) #check z or h

        g3 = -d3 @ h2.T

        g2 = -d2 @ h1.T

        g1 = -d1 @ data['X\_train'].T

        w3 -= alpha\*g3

        w2 -= alpha\*g2

        w1 -= alpha\*g1

    print('Done training.')

    t = np.arange(1,max\_itr+1)

    if save:

        print('Saving data...')

        data\_out = zip(t, L\_train/P, L\_test/data['X\_test'].shape[1], acc\_train, acc\_test)

        filename = 'logs/B2\_4\_data\_w{x}.csv'.format(x = weight\_scale)

        np.savetxt(filename, [p for p in data\_out], delimiter=',', fmt='%f')

        print('Saved.')

    return data\_out

#LOAD DATA

data = sc.loadmat('data/mnist\_sevens\_nines.mat')

#DATA VISUALISATION

plt.figure(1)

for i in range(16):

    plt.subplot(4,4,i+1)

    sample = data['X\_train'][:,i]

    plt.imshow(sample.reshape(28,28).T, cmap='gray')

#plt.axes('off')

plt.tight\_layout()

plt.show()

plt.figure(2)

for i in range(16):

    plt.subplot(4,4,i+1)

    sample = data['X\_train'][:,i+900]

    plt.imshow(sample.reshape(28,28).T, cmap='gray')

#plt.axes('off')

plt.tight\_layout()

plt.show()

P = data['X\_train'].shape[1]

max\_itr = 5000

alpha = 0.1/P

#WEIGHTSCALE 0.01 EXPERIMENT

runNet(data)

##WEIGHTSCALE 0.1 EXPERIMENT

runNet(data, weight\_scale=0.1)