Aditya Das

axd5763

1001675762

Assignment 3

To run the programs:

Language used: python3

Running problem 1: python3 problem1.py

Running problem2: python3 tf.py

Problem 2 has the hidden node commented by default. To run and change the hidden node, kindly change the snippet at the line.

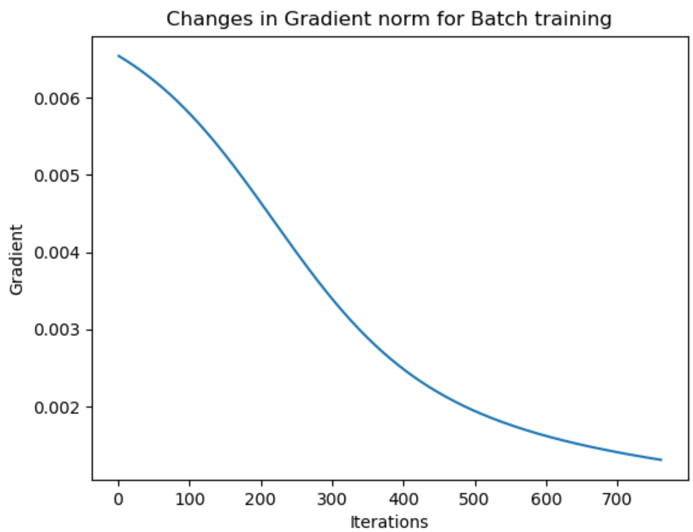
Problem 1

1.Perform batch training using gradient descent. Divide the derivative with the total number of training dataset as you go through iteration (it is very likely that you will get NaN if you don't do this.). Change your learning rate as η = {1, 0.1, 0.01, 0.001}. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training error (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number iterations it took for training and the accuracy that you have.

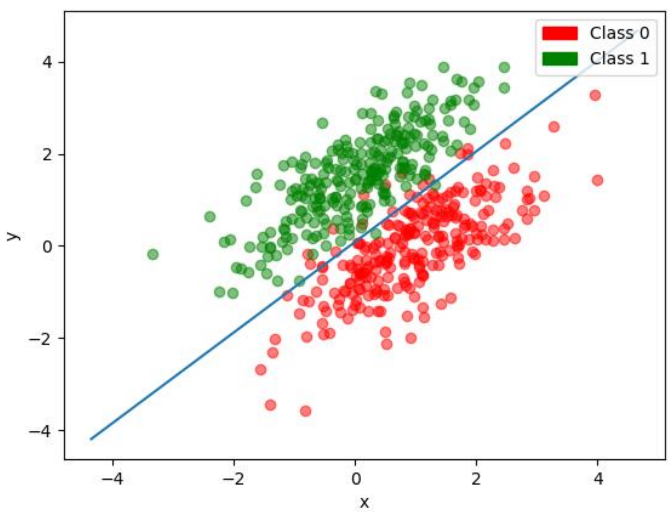
Batch Training:

Error plot for learning rate 0.01:

Gradient Plot for learning rate 0.01:



Scatter Plot and Decision boundary for learning rate 0.01:



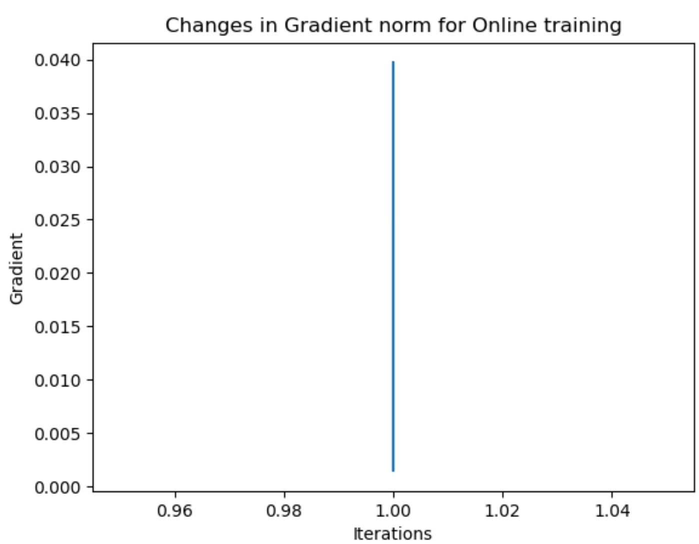
Batch Training Output:

```
Type: Batch Processing with learning rate: 0.01
Iteration(s): 761
Error: 279.812712755828
761
Time(s): 72.295
Weights: [-0.09207623 -1.13280927  1.15048507]
Accuracy = 96.6
```

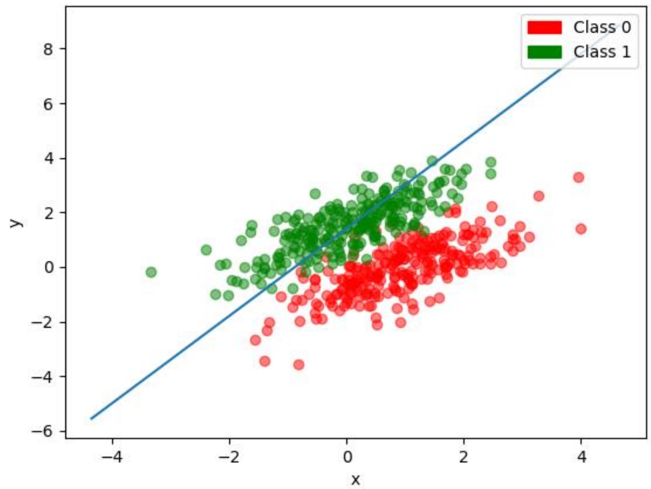
Online Training:

Error plot for learning rate 0.01:

Gradient Plot for learning rate 0.01:



Scatter Plot and Decision boundary for learning rate 0.01:



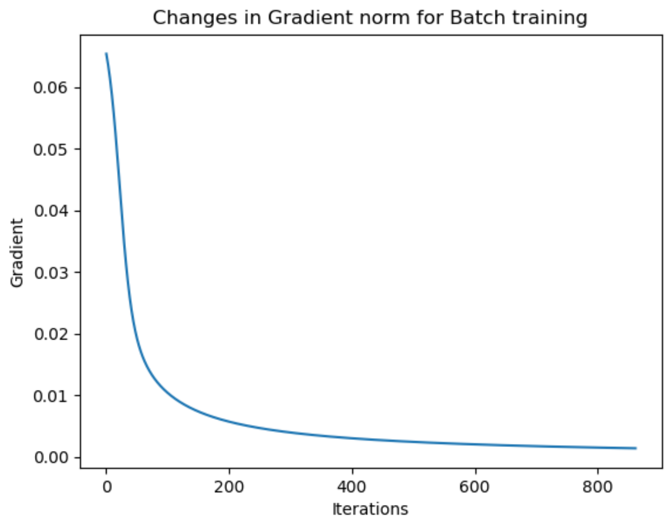
Online Training Output:

```
Type: Online Processing with learning rate: 0.01
Iteration(s): 1
Time(s): 59.431
Weights: [-0.7595109  -0.8684908   0.54235604]
Accuracy = 75.4
```

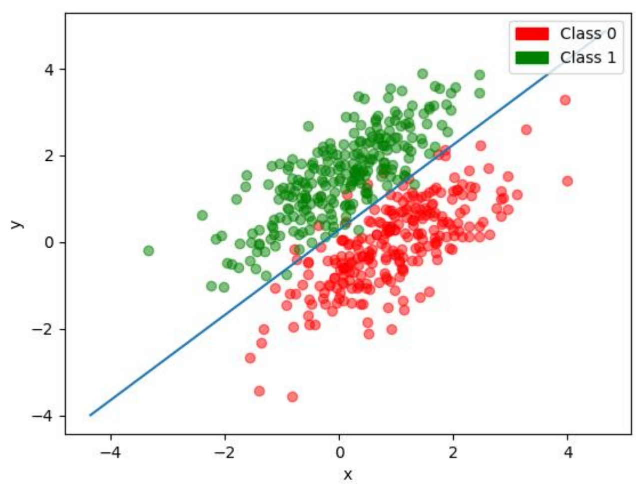
Batch Training:

Error plot for learning rate 0.1:

Gradient Plot for learning rate 0.1:



Scatter Plot and Decision boundary for learning rate 0.1:



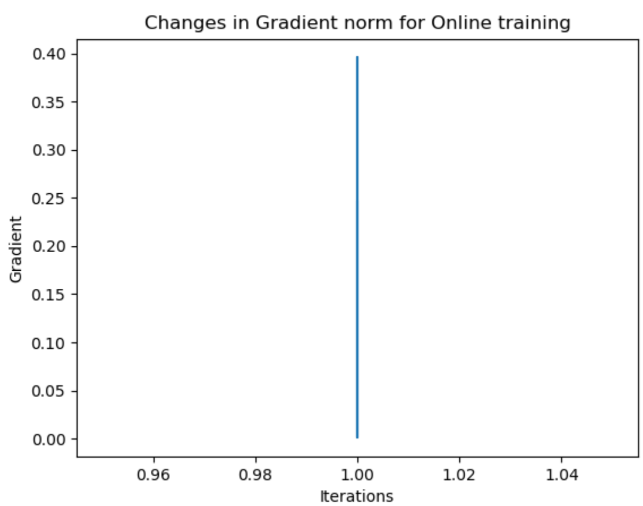
Batch Training Output:

```
Type: Batch Processing with learning rate: 0.1
Iteration(s): 861
Error: 131.66676703505715
861
Time(s): 125.054
Weights: [-0.85546964 -3.02842921  3.07998324]
Accuracy = 97.8
```

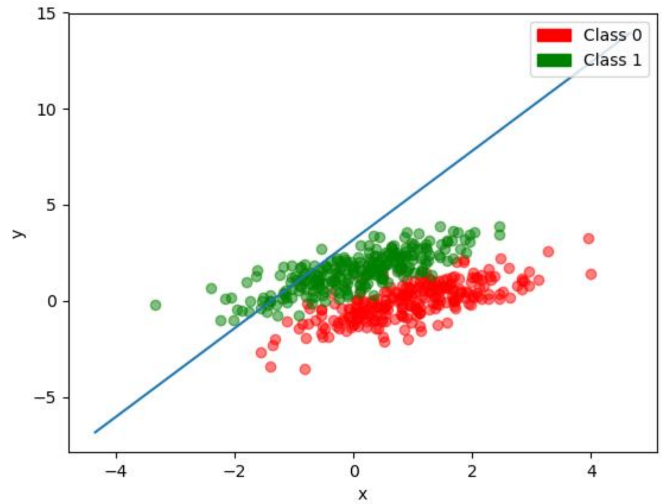
Online Training:

Error plot for learning rate 0.1:

Gradient Plot for learning rate 0.1:



Scatter Plot and Decision boundary for learning rate 0.1:



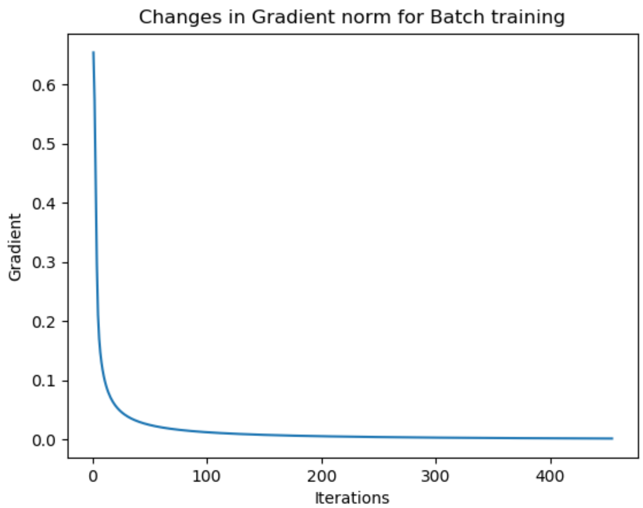
Online Training Output:

```
Type: Online Processing with learning rate: 0.1
Iteration(s): 1
Time(s): 47.031
Weights: [-2.08554468 -1.51506644  0.6556488 ]
Accuracy = 56.39999999999999
```

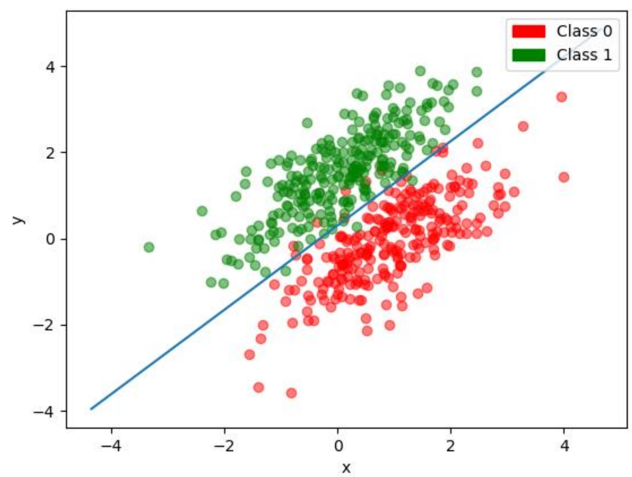
Batch Training:

Error plot for learning rate 1:

Gradient Plot for learning rate 1:



Scatter Plot and Decision boundary for learning rate 1:



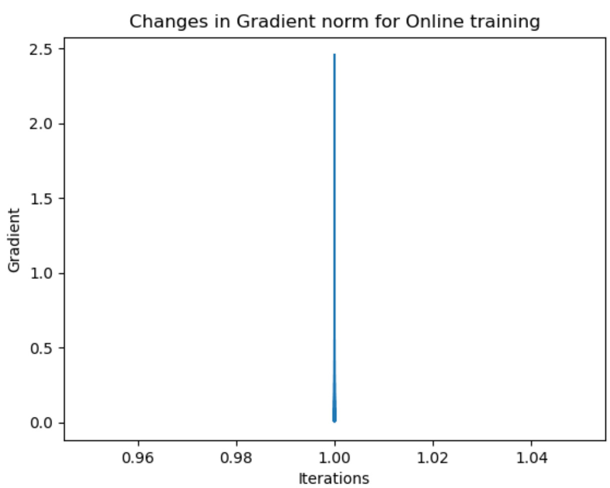
Batch Training Output:

```
Type: Batch Processing with learning rate: 1
Iteration(s): 454
Error: 120.06509771035618
454
Time(s): 50.283
Weights: [-1.26708894 -4.18480858  4.27750628]
Accuracy = 97.8
```

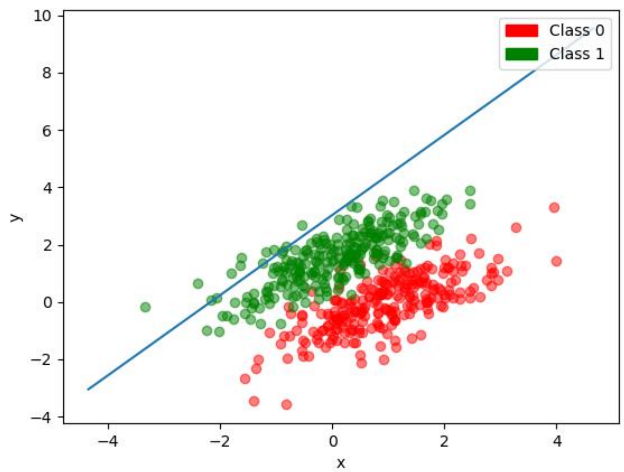
Online Training:

Error plot for learning rate 1:

Gradient Plot for learning rate 1:



Scatter Plot and Decision boundary for learning rate 1:



Online Training Output:

```
Type: Online Processing with learning rate: 1
Iteration(s): 1
Time(s): 44.573
Weights: [-3.87629175 -1.79304478  1.28221432]
Accuracy = 52.2
```

The images are attached in a separate folder images under subfolder problem

Problem 2

1. import tensorflow as tf

2. mnist = tf.keras.datasets.mnist

3.

4. (x_train, y_train),(x_test, y_test) = mnist.load_data()

5. x_train, x_test = x_train / 255.0, x_test / 255.0

6.

7. model = tf.keras.models.Sequential([

8. tf.keras.layers.Flatten(),

9.tf.keras.layers.Dense(512, activation=tf.nn.relu),

10. tf.keras.layers.Dropout(0.2),

11. tf.keras.layers.Dense(10, activation=tf.nn.softmax)

12.])

13.

14. model.compile(

```
15. optimizer='adam',
16. loss='sparse_categorical_crossentropy',
17. metrics=['accuracy'] )
18.
19. model.fit(x_train, y_train, epochs=5)
20. model.evaluate(x_test, y_test)
```

1. In the report, write comments for each line of code given above and explain what this framework is doing.

Line no.	Comments
1	Importing the **tensorflow** library under the alias **tf** to use in the program
2	importing the **mnist** dataset which consists of image of **handwritten digits** of size 28*28 pixels.
4	Loading the **mnist** training data and testing data of size **6000** and **1000** respectively. x_train and x_test represents the matrix of 28*28, while y_train and y_test is the output.
5	**Normalizing** the dataset with values between 0 to 1
7	**tf.keras.models.Sequential** creates a deep learning model with linear stack of layers.
8	tf.keras.layers.Flatten() Flattens the input.
9	**tf.keras.layers.Dense(512, activation = tf.nn.relu)** This creates a layer in the network which has 512 nodes and all with the activation function of 'Rectified Linear Unit'.
10	**tf.keras.layers.Dropout(0.2)** This helps to evaluate the network better by droping out 20% of nodes for each iterations. This helps to deviate the model from over fitting.
11	**tf.keras.layers.Dense(10, activation = tf.nn.softmax)** This creates a layer in the network which has 10 nodes and all with the activation function of 'Softmax'. Since this is the last layer, we can say the same as output layer.
14	**model.compile()** helps us to add configurations to the model. They include learning rates, error rate methods, different optimizer types and many more.
15	Model to use Adam Optimizer Algorithm
16	Model to use Sparse Categorical Cross Entropy to calculate the loss
17	Metrics to be evaluated by the model during training and testing. Here, accuracy
19	**model.fit(x_train, y_train, epochs=5)** helps to train the model with input, required output. Here epochs is number of batch iterations.

20	**Model.evaluate(x_test, y_test)** helps to test the model with input and required output.

2. Change the number of hidden nodes to 5, 10, 128 and 512. Report how the testing accuracy changes for the testing data. Report the result and your observation in the report.

Hidden nodes 5

```
Epoch 1/5
60000/60000 [==============================] - 6s 96us/sample - loss: 1.3109 - acc: 0.5325
Epoch 2/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.9968 - acc: 0.6524
Epoch 3/5
60000/60000 [==============================] - 6s 94us/sample - loss: 0.9489 - acc: 0.6692
Epoch 4/5
60000/60000 [==============================] - 6s 93us/sample - loss: 0.9294 - acc: 0.6773
Epoch 5/5
60000/60000 [==============================] - 6s 93us/sample - loss: 0.9198 - acc: 0.6829
10000/10000 [==============================] - 1s 51us/sample - loss: 0.5562 - acc: 0.8529
[0.5561643992900849, 0.8529]
```

Hidden notes 10

```
Epoch 1/5
60000/60000 [==============================] - 6s 107us/sample - loss: 0.8600 - acc: 0.7121
Epoch 2/5
60000/60000 [==============================] - 6s 96us/sample - loss: 0.5897 - acc: 0.8051
Epoch 3/5
60000/60000 [==============================] - 6s 96us/sample - loss: 0.5509 - acc: 0.8195
Epoch 4/5
60000/60000 [==============================] - 6s 97us/sample - loss: 0.5298 - acc: 0.8259
Epoch 5/5
60000/60000 [==============================] - 6s 96us/sample - loss: 0.5207 - acc: 0.8280
10000/10000 [==============================] - 1s 56us/sample - loss: 0.2797 - acc: 0.9217
[0.27966173109412196, 0.9217]
```

Hidden notes 128

```
Epoch 1/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.2910 - acc: 0.9159
Epoch 2/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.1396 - acc: 0.9587
Epoch 3/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.1053 - acc: 0.9679
Epoch 4/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0888 - acc: 0.9721
Epoch 5/5
60000/60000 [==============================] - 6s 104us/sample - loss: 0.0745 - acc: 0.9763
10000/10000 [==============================] - 1s 60us/sample - loss: 0.0739 - acc: 0.9773
[0.07388623829467687, 0.9773]
```

Hidden notes 512

```
Epoch 1/5
60000/60000 [==============================] - 6s 98us/sample - loss: 0.2219 - acc: 0.9356
Epoch 2/5
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0991 - acc: 0.9696
Epoch 3/5
60000/60000 [==============================] - 7s 113us/sample - loss: 0.0677 - acc: 0.9786
Epoch 4/5
60000/60000 [==============================] - 6s 104us/sample - loss: 0.0528 - acc: 0.9836
Epoch 5/5
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0435 - acc: 0.9862
10000/10000 [==============================] - 1s 54us/sample - loss: 0.0608 - acc: 0.9810
[0.06079931019728538, 0.981]
```

3. Now, remove the hidden layer in the code and train the model. The trained model contains the weights that it has learned from training. Plot the "new representation" that it has learned for each number from training and include them in the report. That is, reshape the learned weights (i.e.,vector) to the image dimension (in 2D, i.e., 28x28) and show them. You will see some number-like features.

The program was changed, and the following code was added.

a = model.get_weights()[0]

for i in range(0,10):

b = tf.reshape(a[:,i], [28,28]) # Choosing column i to procude 28x28 color matrix

plt.title("Image " + str(i) + ":")

plt.imshow(tf.Session().run(b), cmap='gray')

plt.show()

```
10000/10000 [==============================] - 0s 26us/sample - loss: 0.2752 - acc: 0.9246
```

The images are attached in a separate folder images under the subfolder problem 2.