Amartejas Manjunath

axm2606

1001742606

Homework 3

I will be using my late day submission for this homework

To run the programs:

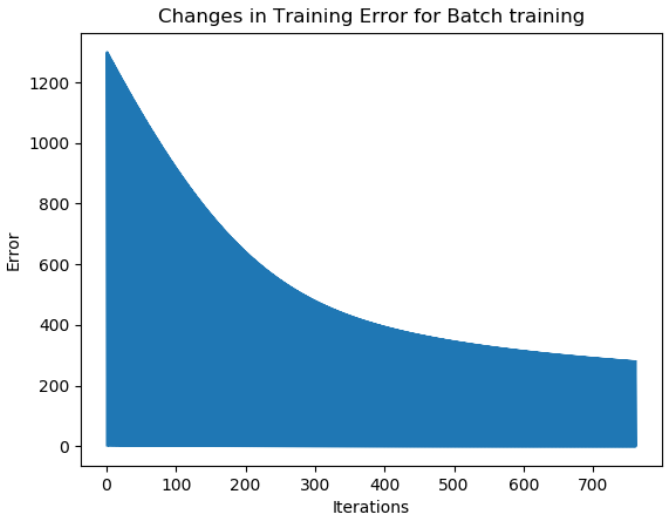
Language used: python3

Running problem 1: python3 task1.py

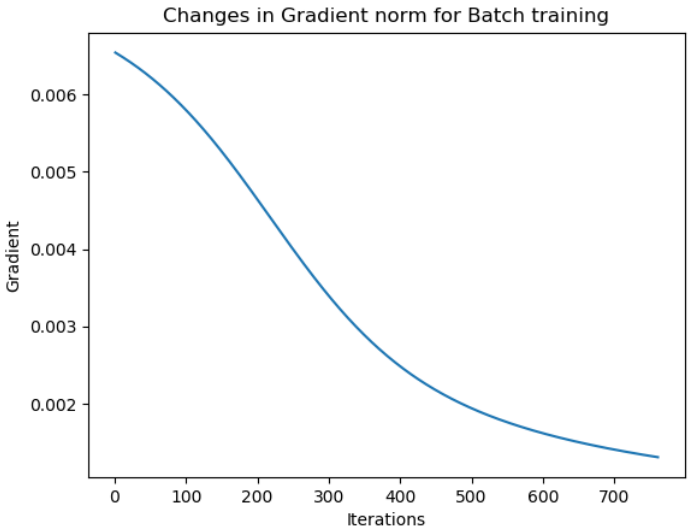
Problem 1

Batch Training:

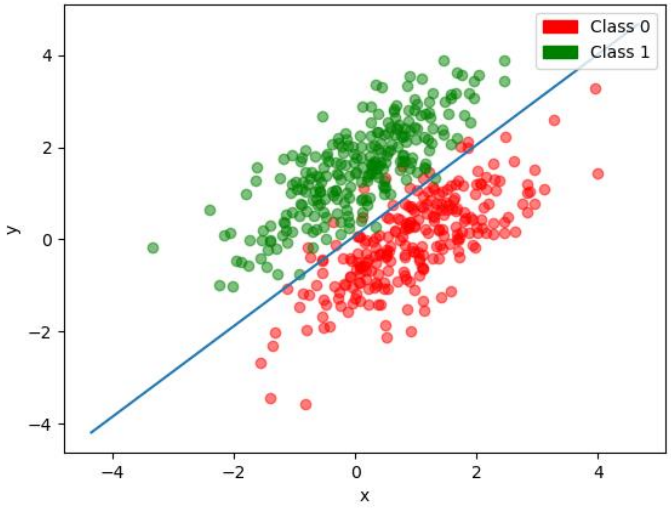
Error plot for learning rate 0.01:



Gradient Plot for learning rate 0.01:



Scatter Plot and Decision boundary for learning rate 0.01:



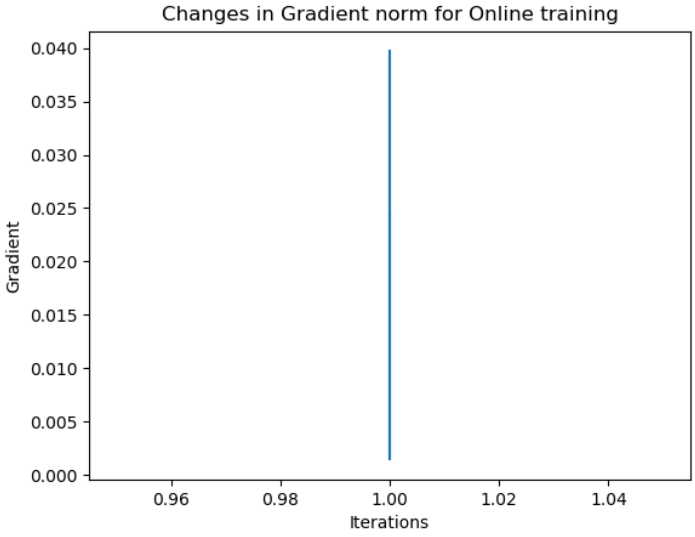
Batch Training Output:

```
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
```

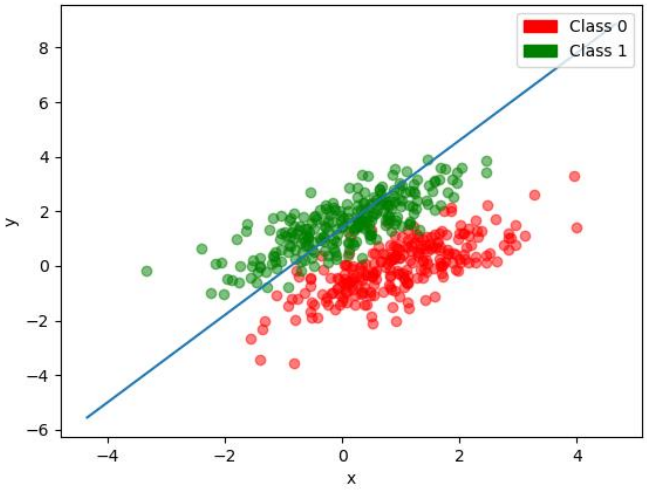
Online Training:

Error plot for learning rate 0.01:

Gradient Plot for learning rate 0.01:



Scatter Plot and Decision boundary for learning rate 0.01:



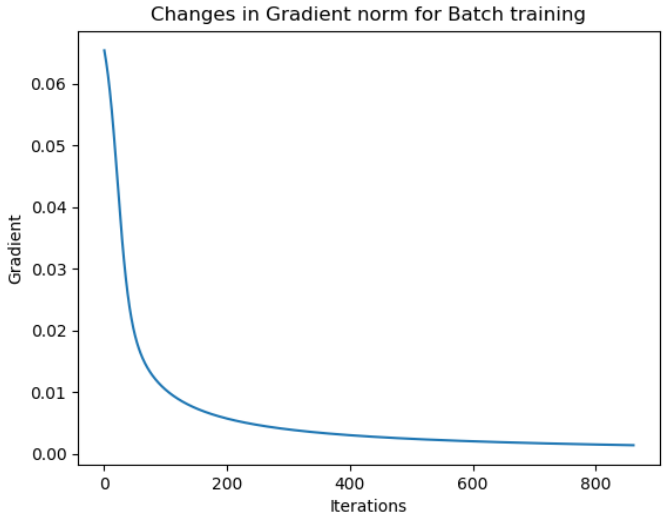
Online Training Output:

```
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
```

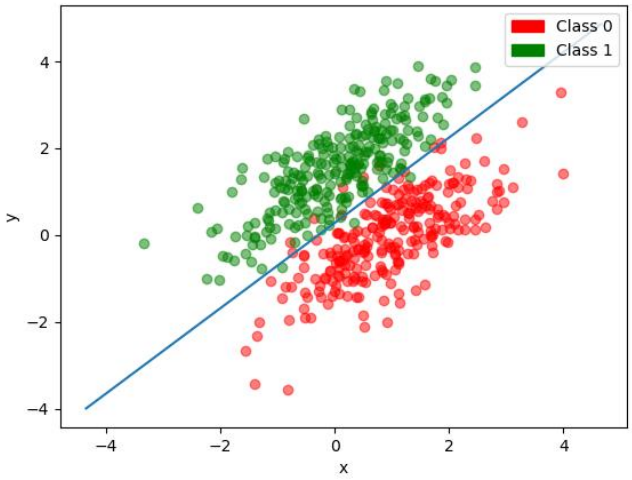
Batch Training:

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:

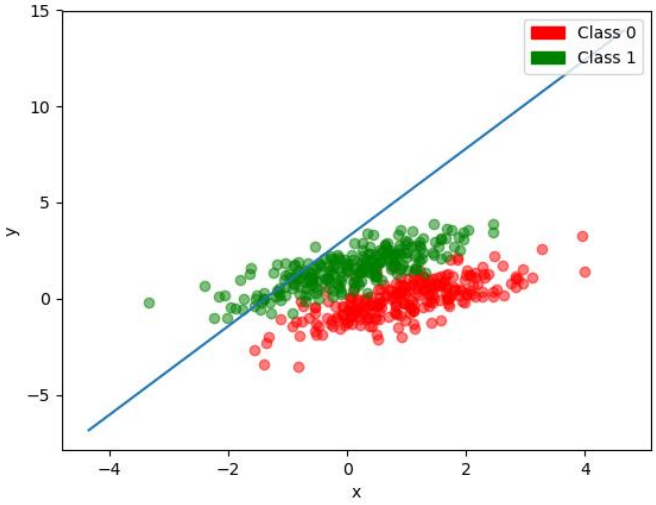
```
Type: Batch Processing with learning rate: 0.1
Iteration(s): 861
Error: 131.66676703505715
861
Time(s): 125.054
Weights: [-0.85546964 -3.02842921  3.07998324]
Accuracy = 97.8
```

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:

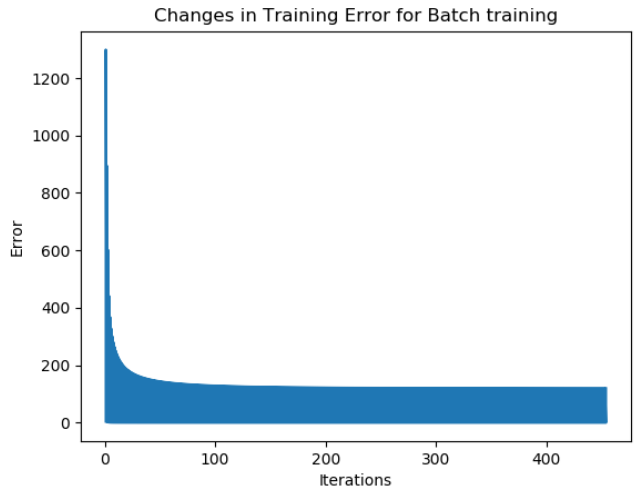


Online Training Output:

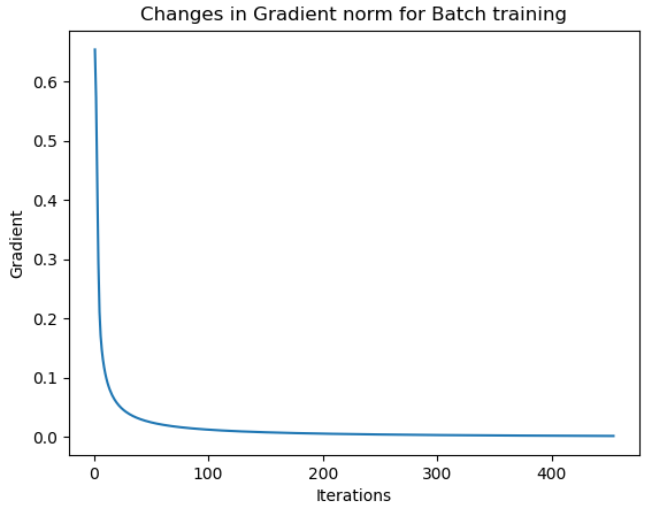
```
Type: Online Processing with learning rate: 0.1
Iteration(s): 1
Time(s): 47.031
Weights: [-2.08554468 -1.51506644  0.6556488 ]
Accuracy = 56.39999999999999
```

Batch Training:

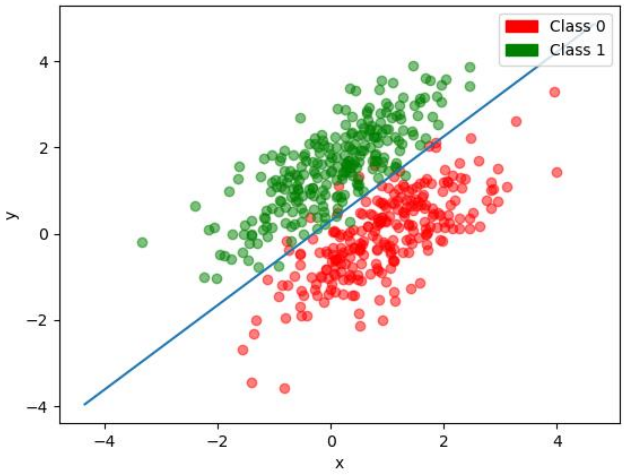
Error plot for learning rate 1:



Gradient Plot for learning rate 1:



Scatter Plot and Decision boundary for learning rate 1:



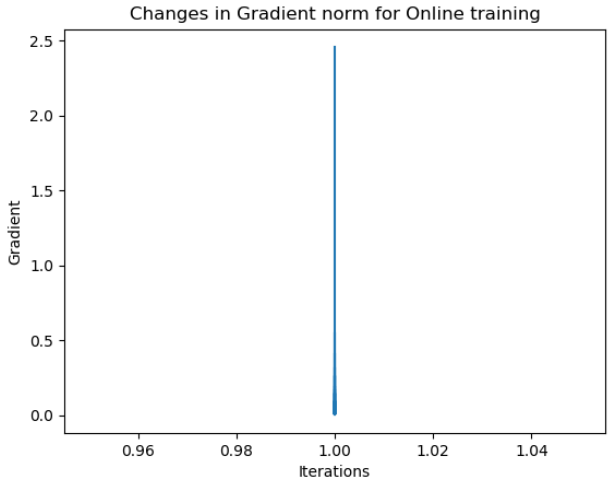
Batch Training Output:

```
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
```

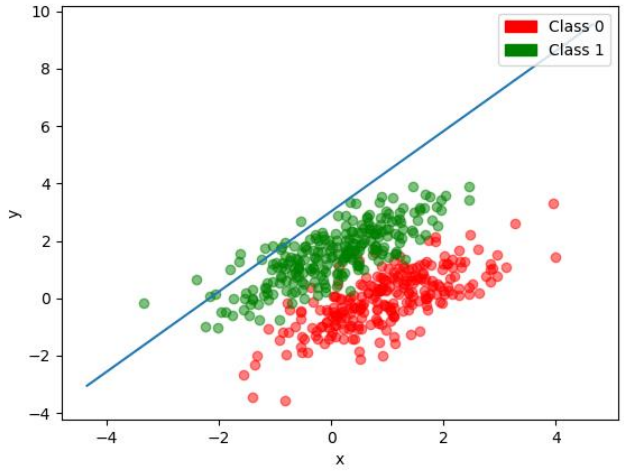
Online Training:

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
```

Problem 2

2.1) Training and loading data from the dataset. With at least one hidden layer. Check task2.1 folder

```
    train_data, test_data: type -- list; You need to transform them into numpy if you want to use them for logistic regression.
    '''

    ###### Modify/Add your code here to implement "transform" used below for data transform/normalization ######
    transform = transforms.Compose(
        [transforms.Grayscale(num_output_channels=1),
         transforms.ToTensor(),
         transforms.Normalize(mean=[0.485],
                              std=[0.229])])

    cifar10_class_list = [0,1,2]
    ## The index of the classes selected, for example: [0, 1]. If none indices are selected, the whole dataset will be used.
    fashion_mnist_class_list = [0,1]  ## You may need to change the index number if you want to select different classes.


    ###### Do NOT modify the code below
    ## generate data loader for CIFAR10 and Fashion MNIST
    if args.dataset_name == "cifar10":
        train_data_raw = torchvision.datasets.CIFAR10(root=args.data_path, train=True, download=True,
                                                      transform=transform)
        test_data_raw = torchvision.datasets.CIFAR10(root=args.data_path, train=False, download=True,
                                                     transform=transform)

        if len(cifar10_class_list) > 0:
            train_data = data_subset(train_data_raw, cifar10_class_list)
            test_data = data_subset(test_data_raw, cifar10_class_list)
```

2.2) Adding one hidden layer with 64 nodes

```
class Net(torch.nn.Module):
    def __init__(self, args):
        h_sizes = 64
        super(Net, self).__init__()
        self.args = args
        self.dropout = args.dropout
        self.fc = torch.nn.Linear(1024, 64)
        #self.out_cifar10 = torch.nn.Linear(args.image_cifar10_width * args.image_cifar10_height,
                                           #args.class_number_cifar10)

        #self.out_fashion_mnist = torch.nn.Linear(args.image_fashion_mnist_width * args.image_fashion_mnist_height,
                                                 #    args.class_number_fashion_mnist)

    def forward(self, x):
        #print(x.shape)
        if self.args.dataset_name == "cifar10":
            x = x.view(-1, self.args.image_cifar10_width * self.args.image_cifar10_height)
            #x = self.dropout(x)
            x = F.relu(self.fc(x))  # instead of Heaviside step fn
            #x = self.out_cifar10(x)

            return F.log_softmax(x)
```

The Output Accuracy file and loss file is in the folder named Task2.2

2.3) When we change the dropout to 0.2 there is a increase in accuracy and decrease in loss for the test dataset. This can be seen in fig below

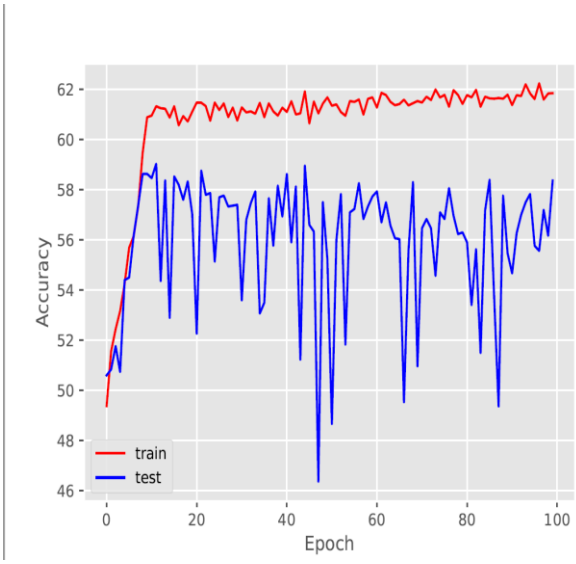


Fig. Dropout 0.0

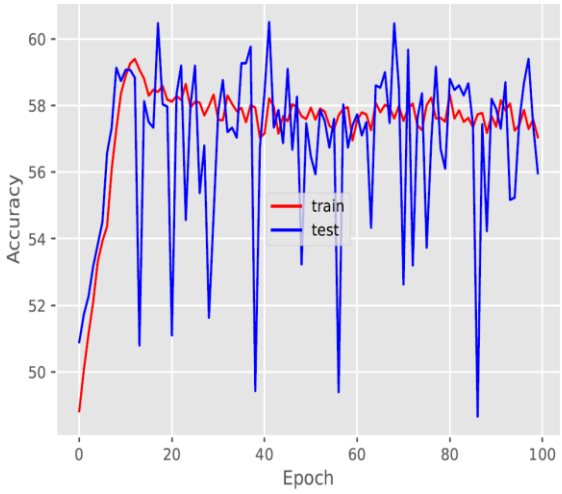


Fig. Dropout 0.2

2.4) Adding multiple hidden layers and using ReLu . Changing learning rate and dropout parameters

```
class Net(torch.nn.Module):
    def __init__(self, args):
        h_sizes = 64
        super(Net, self).__init__()
        self.args = args
        self.dropout = args.dropout
        self.fc = torch.nn.Linear(1024, 256)
        self.fc1 = torch.nn.Linear(256, 128)
        self.fc2 = torch.nn.Linear(128, 64)
        self.out_cifar10 = torch.nn.Linear(64,
                                           args.class_number_cifar10)

        #self.out_fashion_mnist = torch.nn.Linear(args.image_fashion_mnist_width * args.image_fashion_mnist_height,
                                           #      args.class_number_fashion_mnist)

    def forward(self, x):
        #print(x.shape)
        if self.args.dataset_name == "cifar10":
            x = x.view(-1, self.args.image_cifar10_width * self.args.image_cifar10_height)
            #x = self.dropout(x)
            x = F.relu(self.fc(x))  # instead of Heaviside step fn
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = self.out_cifar10(x)

            return F.log_softmax(x)
```

Increases the accuracy to 85%

2.5) Adding the same hidden layer as before and loading and training the Fashion MNIST dataset

```
elif self.args.dataset_name == "fashion_mnist":
    x = x.view(-1, self.args.image_fashion_mnist_width * self.args.image_fashion_mnist_height)


    if self.args.visual_flag:
        x = F.relu(self.fc(x))  # instead of Heaviside step fn

        network_weight = self.fc.weight.data




        return x, network_weight
    else:
        x = self.drop(x)
        x = F.relu(self.fc(x))  # instead of Heaviside step fn
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.out_fashion_mnist(x)
        x = F.log_softmax(x)
        ###### Mofify/Add your code here ######




        return x
```

Accuracy was 99% can be found in “task2.5” folder

2.6) The output for this can be found in the folder “task2.6”