Homework Assignment - 1

Name: Sagar Patel

NETID: sp5894

Question 1

Fun with vector calculus. This question has two parts.

1.a

If x is a d-dimensional vector variable, write down the gradient of the function $f(x) = ||x||_2^2$.

Solution:

We know that $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{bmatrix}$ and it is given that $f(x) = ||x||_2^2$

$\Rightarrow f(x) = ||x||_2^2 \Rightarrow f(x) = x^T x$

$\Rightarrow f(x) = \sum_{i=1}^{d} x_i^2$

Therefore, the gradient of $f(x) = f'(x) = 2x$

1.b

Suppose we have n data points are real d-dimensional vectors. Analytically derive a constant vector μ for which the MSE loss function $L(\mu) = \sum_{i=1}^{n} ||x_i - \mu||_2^2$ is minimized.

Solution:

Let $x_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_i^{(3)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$ and it is given that $L(\mu) = \sum_{i=1}^{n} ||x_i - \mu||_2^2$

$\Rightarrow L(\mu) = \sum_{i=1}^{n} ||x_i - \mu||_2^2$

$\Rightarrow L(\mu) = \sum_{i=1}^{n} (x_i^{(1)} - \mu^{(1)})^2 + (x_i^{(2)} - \mu^{(2)})^2 + (x_i^{(3)} - \mu^{(3)})^2 + ... + (x_i^{(d)} - \mu^{(d)})^2$

The gradient can be calculated by differentiating $L(\mu)$ with respect to $\mu^{(j)}$

$\Rightarrow \frac{\partial L(\mu)}{\partial \mu^{(j)}} = \frac{\partial((x_i^{(1)}-\mu^{(1)})^2+(x_i^{(2)}-\mu^{(2)})^2+(x_i^{(3)}-\mu^{(3)})^2+...+(x_i^{(d)}-\mu^{(d)})^2)}{\partial u^{(j)}}$

$\Rightarrow \frac{\partial L(\mu)}{\partial \mu^{(j)}} = -2\sum_{i=1}^{n}(x_i^{(j)} - n.\mu^{(j)})$

Equating that to zero, we get $\mu = \frac{\sum_{i=1}^{n} x_i}{n}$

If we differentiate the given equation one more time, we get -

MSE' $= \frac{\partial L(\mu)}{\partial \mu^{(j)}} = -2\sum_{i=1}^{n}(x_i^{(j)} - n.\mu^{(j)})$

MSE" $= 2.n$

Now, $2.n > 0$ which means that the equation will minimize.

Question 2

Linear regression with non-standard losses. In class we derived an analytical expression for the optimal linear regression model using the least squares loss. If X is the matrix of training data points (stacked row-wise) and y is the vector of labels, then:

2.a

Using matrix/vector notation, write down a loss function that measures the training error in terms of the l_1-norm.

Solution:

Loss function that measures the training error in terms of l_1 norm can be called Mean Absolute Error (MAE).

Mean Absolute Error $= \frac{\sum_{i=1}^{n} |y_i - <\omega, x_i>|}{n}$

2.b

Can you write down the optimal linear model in closed form? If not, why not?

Solution:

The closed-form solution is preferred for smaller data - if computing a matrix inverse is not a concern. For very large datasets, or datasets where the inverse of $X^T X$ may not exist (the matrix is non-invertible), the Gradient Descent or Stochastic Gradient Descent approaches are to be preferred.

The weight of the model is calculated by -

$\omega = (X^T X)^{-1} X^T y$

So in conclusion, there can not be an optimal linear model in closed form.

2.c

If the answer to b is no, can you think of an alternative algorithm to optimize the loss function? Comment on its pros and cons.

Solution:

Integrate a Regularization paramater.

Pros:

- Simple and intuitive.
- Both the systems then tend to mix equally and that would perform better.

Cons:

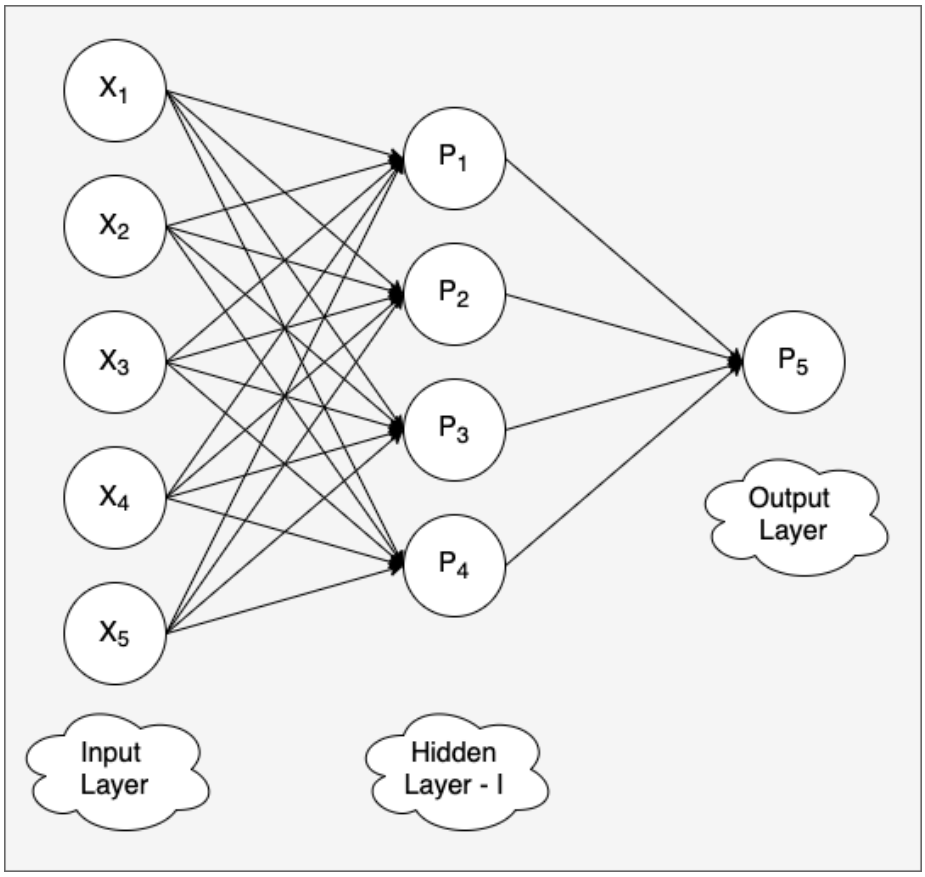
- Depends on the problem we are dealing with. It is not a very generalized solution in my opinion.

Question 3

Hard coding a multi-layer perceptron. The functional form for a single perceptron is given by $y = \text{sign}(< \omega, x > +b)$, where x is the data point and y is the predicted label. Suppose your data is 5-dimensional (i.e., $x = (x_1, x_2, x_3, x_4, x_5)$) and real-valued. Find a simple 2-layer network of perceptrons that implements the *Decreasing-order* function, i.e., it returns +1 if $x_1 > x_2 > x_3 > x_4 > x_5$ and -1 otherwise. Your network should have two layers: the input nodes, feeding into 4 hidden perceptrons, which in turn feed into 1 output perceptron. Clearly indicate all the weights and biases of all the 5 perceptrons in your network.

Solution:

Let us consider that a simple 2-layer perceptron network looks like this.



In this diagram, we have considered P_5 to be the output perceptron (Also called y).

We know that $y = sign(< \omega, x > +b)$ and that $x = (x_1, x_2, x_3, x_4, x_5)$

$Z =< \omega, x > +b \Rightarrow y = sign < Z >$

Now, let us consider Perceptron 1 P_1. In this case, we compare x_1 and x_2.

Therefore, $w_1 = -1, w_2 = 1, w_3 = 0, w_4 = 0, w_5 = 0$ and $b_1 = 0$

$\Rightarrow Z_1 = x_2 - x_1$

$$\Rightarrow y_1 = sign < Z_1 >= \begin{cases} -1, & \text{if } Z_1 \leq 0 \\ 1, & \text{if } Z_1 > 0 \end{cases}$$

Now, let us consider Perceptron 2 P_2. In this case, we compare x_2 and x_3.

Therefore, $w_1 = 0, w_2 = -1, w_3 = 1, w_4 = 0, w_5 = 0$ and $b_2 = 0$

$\Rightarrow Z_2 = x_3 - x_2$

$$\Rightarrow y_2 = sign < Z_2 >= \begin{cases} -1, & \text{if } Z_2 \leq 0 \\ 1, & \text{if } Z_2 > 0 \end{cases}$$

Now, let us consider Perceptron 3 P_3. In this case, we compare x_3 and x_4.

Therefore, $w_1 = 0, w_2 = 0, w_3 = -1, w_4 = 1, w_5 = 0$ and $b_3 = 0$

$\Rightarrow Z_3 = x_4 - x_3$

$$\Rightarrow y_3 = sign < Z_3 >= \begin{cases} -1, & \text{if } Z_3 \leq 0 \\ 1, & \text{if } Z_3 > 0 \end{cases}$$

Now, let us consider Perceptron 4 P_4. In this case, we compare x_4 and x_5.

Therefore, $w_1 = 0, w_2 = 0, w_3 = 0, w_4 = -1, w_5 = 1$ and $b_4 = 0$

$\Rightarrow Z_4 = x_5 - x_4$

$$\Rightarrow y_4 = sign < Z_4 >= \begin{cases} -1, & \text{if } Z_4 \leq 0 \\ 1, & \text{if } Z_4 > 0 \end{cases}$$

Finally, for the output layer, we consider the Perceptron 5 P_5. In this case, we compare all the previous four perceptrons.

Therefore, $w_1 = 1, w_2 = 1, w_3 = 1, w_4 = 1, w_5 = 1$ and $b_5 = -3$

$$\Rightarrow y = sign < Z_5 >= \begin{cases} -1, & \text{if } Z_5 \leq 0 \\ 1, & \text{if } Z_5 > 0 \end{cases}$$

Question 4

This exercise is meant to introduce you to neural network training using Pytorch. Open the (incomplete) Jupyter notebook provided as an attachment to this homework in Google Colab (or other cloud service of your choice) and complete the missing items. Save your finished notebook in PDF format and upload along with your answers to the above theory questions in a single PDF.

Link to the Notebook

Homework Assignment - 1

Name: Sagar Patel

NETID: sp5894

OK, thus far we have been talking about linear models. All these can be viewed as a single-layer neural net. The next step is to move on to multi-layer nets. Training these is a bit more involved, and implementing from scratch requires time and effort. Instead, we just use well-established libraries. I prefer PyTorch, which is based on an earlier library called Torch (designed for training neural nets via backprop).

```
import numpy as np
import torch
import torchvision
```

Torch handles data types a bit differently. Everything in torch is a *tensor*.

```
a = np.random.rand(2,3)
b = torch.from_numpy(a)

# Q4.1 Display the contents of a, b
print('a: ',a)
print('b: ',b)
```

```
a:  [[0.85717764 0.62909467 0.46403862]
 [0.60939195 0.57009568 0.0865028 ]]
b:  tensor([[0.8572, 0.6291, 0.4640],
        [0.6094, 0.5701, 0.0865]], dtype=torch.float64)
```

The idea in Torch is that tensors allow for easy forward (function evaluations) and backward (gradient) passes.

```
A = torch.rand(2,2)
b = torch.rand(2,1)
x = torch.rand(2,1, requires_grad=True)

y = torch.matmul(A,x) + b

print(y)
z = y.sum()
print(z)
```

```
z.backward()
print(x.grad)
print(x)
```

```
tensor([[0.9474],
        [1.1385]], grad_fn=<AddBackward0>)
tensor(2.0859, grad_fn=<SumBackward0>)
tensor([[1.8082],
        [1.0379]])
tensor([[0.2059],
        [0.5857]], requires_grad=True)
```

Notice how the backward pass computed the gradients using autograd. OK, enough background. Time to train some networks. Let us load the *Fashion MNIST* dataset, which is a database of grayscale images of clothing items.

```
trainingdata = torchvision.datasets.FashionMNIST('./FashionMNIST/',train=True,downloac
testdata = torchvision.datasets.FashionMNIST('./FashionMNIST/',train=False,download=Tı
```

Let us examine the size of the dataset.

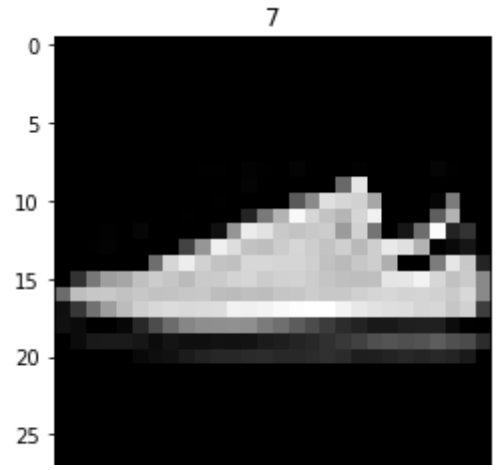
```
# Q4.2 How many training and testing data points are there in the dataset?
# What is the number of features in each data point?
print('Training Points: ',len(trainingdata))
print('Test Points: ',len(testdata))
print('Features in each Data Point: ',trainingdata[0][0].size())
```

```
Training Points:  60000
Test Points:  10000
Features in each Data Point:  torch.Size([1, 28, 28])
```

Let us try to visualize some of the images. Since each data point is a tensor (not an array) we need to postprocess to use matplotlib.

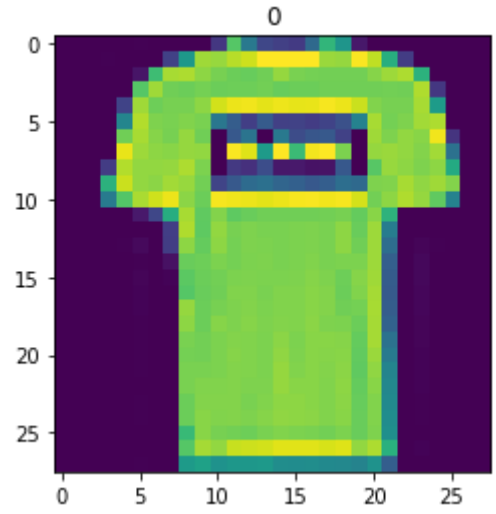
```
import matplotlib.pyplot as plt
%matplotlib inline
import random

image, label = trainingdata[0]
# Q4.3 Assuming each sample is an image of size 28x28, show it in matplotlib.
idx = random.randint(0,9)
plt.imshow(trainingdata[idx][0][0].numpy(), cmap='gray')
plt.title('%i'% trainingdata[idx][1])
```

```
Text(0.5, 1.0, '7')
```




```
import matplotlib.pyplot as plt
%matplotlib inline
import random

image, label = trainingdata[0]
# Q4.3 Assuming each sample is an image of size 28x28, show it in matplotlib.
idx = random.randint(0,9)
plt.imshow(trainingdata[idx][0][0].numpy())
plt.title('%i'% trainingdata[idx][1])
```

```
Text(0.5, 1.0, '0')
```



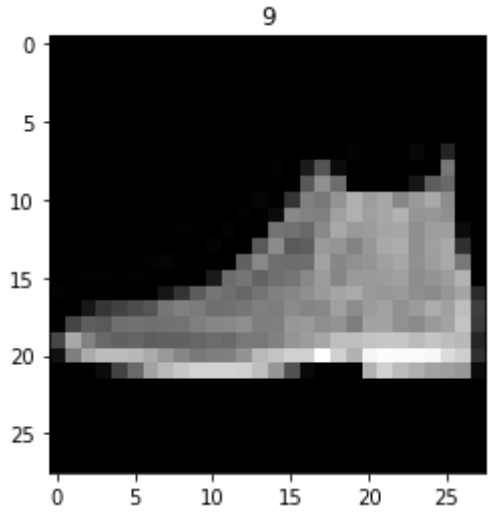
Let's try plotting several images. This is conveniently achieved in PyTorch using a *data loader*, which loads data in batches.

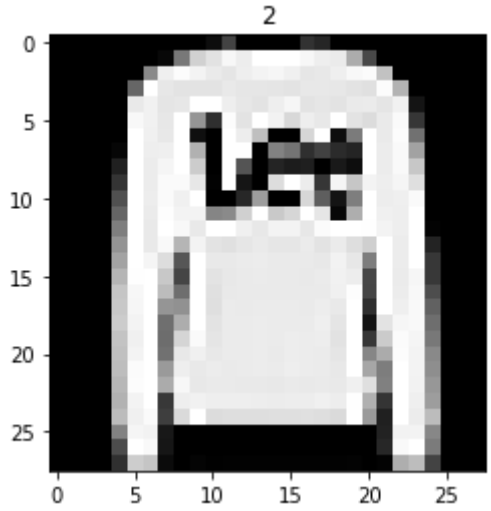
```
trainDataLoader = torch.utils.data.DataLoader(trainingdata, batch_size=64, shuffle=Tru
testDataLoader = torch.utils.data.DataLoader(testdata, batch_size=64, shuffle=False)
images, labels = iter(trainDataLoader).next()
print(images.size(), labels)
```

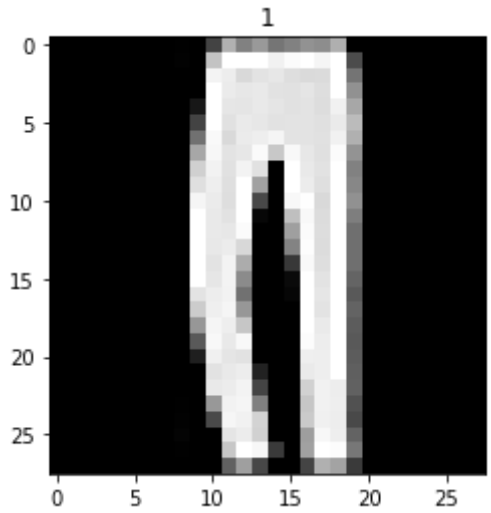
```
torch.Size([64, 1, 28, 28]) tensor([8, 5, 2, 3, 5, 4, 0, 8, 1, 0, 7, 6, 5, 0, 4,
        1, 6, 7, 5, 6, 9, 4, 3, 4, 2, 9, 5, 2, 8, 4, 0, 3, 3, 2, 0, 0, 8, 6, 5,
```

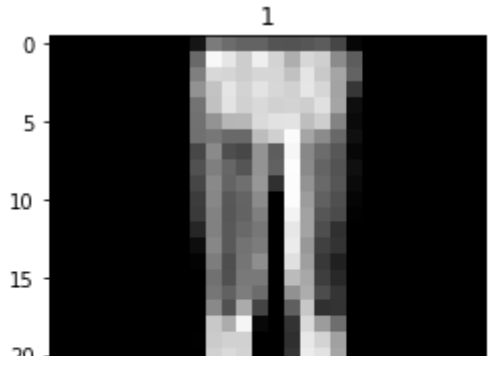
```
        6, 8, 2, 3, 2, 1, 4, 5, 6, 7, 0, 6, 6, 2, 2, 9])
```

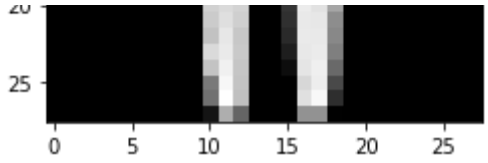
```
# Q4.4 Visualize the first 10 images of the first minibatch
# returned by testDataLoader.
images, labels = iter(testDataLoader).next()
for i in range(10):
  plt.imshow(images[i][0].numpy(), cmap='gray')
  plt.title('%i'% labels[i])
  plt.show()
```

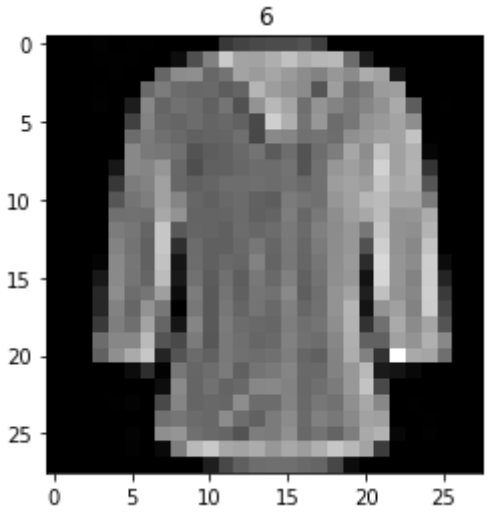

9

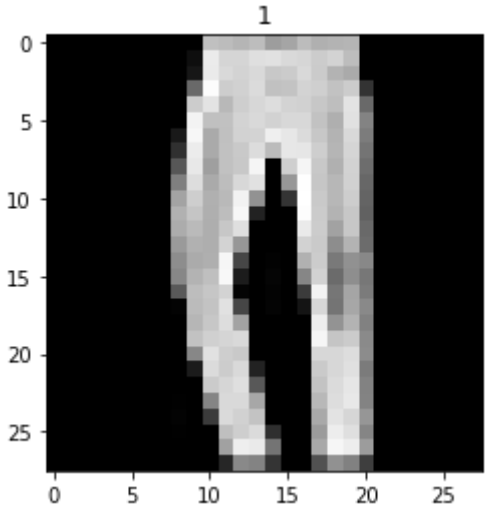

2

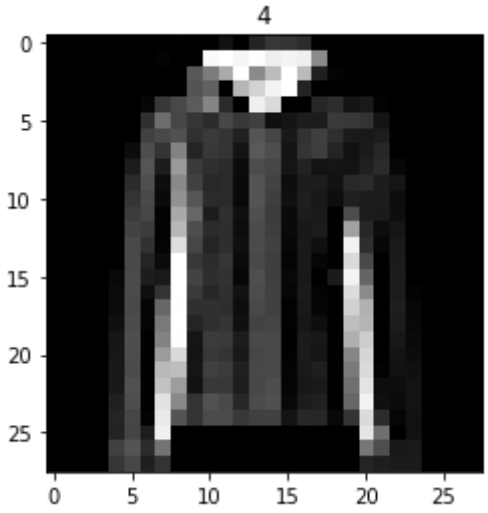

1


1



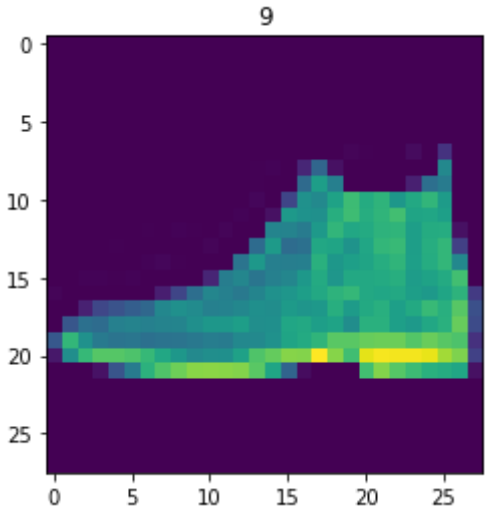


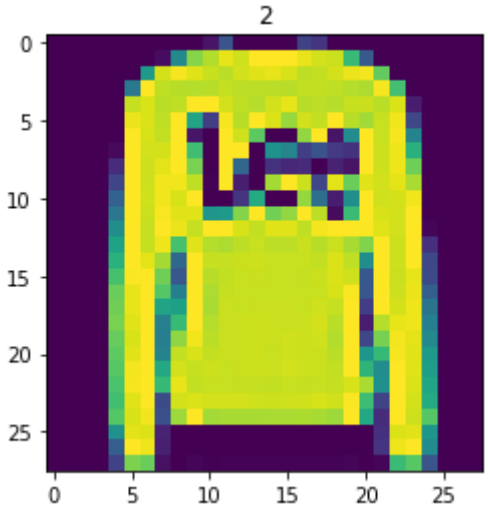


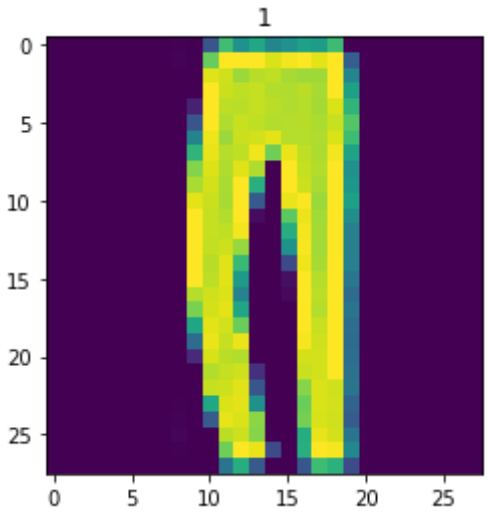


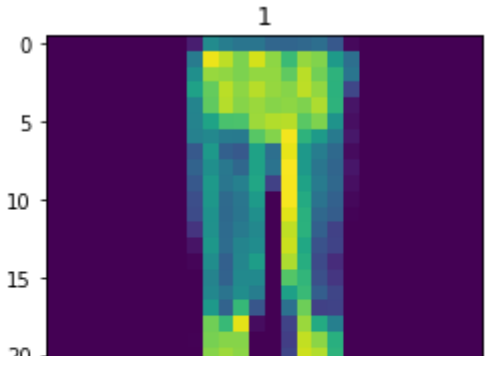

```
images, labels = iter(testDataLoader).next()
for i in range(10):
  plt.imshow(images[i][0].numpy())
```

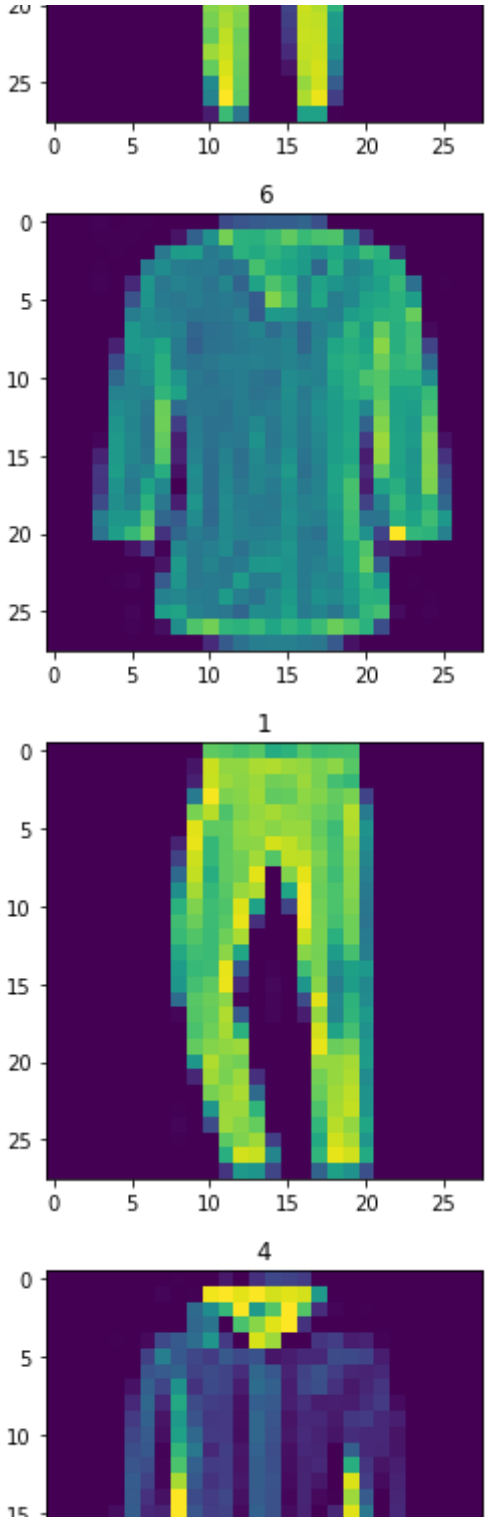
```
    plt.title('%i'% labels[i])
    plt.show()
```


9


2


1


1



Now we are ready to define our linear model. Here is some boilerplate PyTorch code that implements the forward model for a single layer network for logistic regression (similar to the one discussed in class notes).

```
class LinearReg(torch.nn.Module):
  def __init__(self):
    super(LinearReg, self).__init__()
    self.linear = torch.nn.Linear(28*28,10)

  def forward(self, x):
```

```
        x = x.view(-1,28*28)
        transformed_x = self.linear(x)
        return transformed_x

net = LinearReg()
Loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
```

Cool! Now we have set everything up. Let's try to train the network.

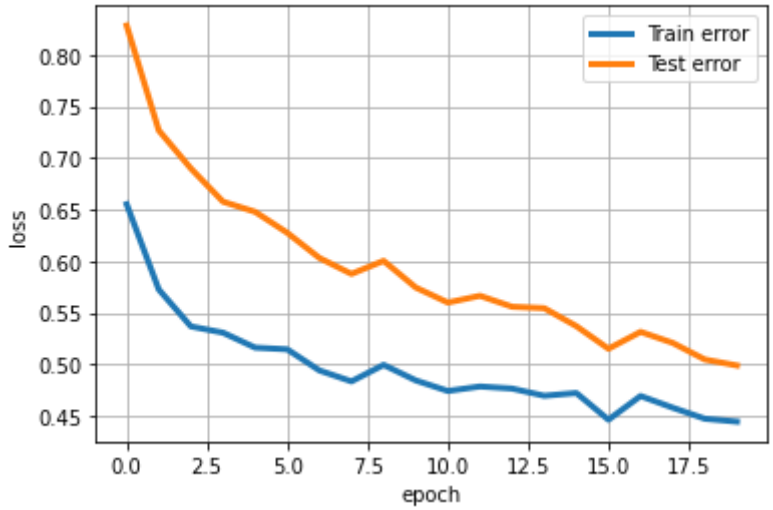
```
train_loss_history = []
test_loss_history = []

# Q4.5 Write down a for-loop that trains this network for 20 epochs,
# and print the train/test losses.
# Save them in the variables above. If done correctly, you should be able to
# execute the next code block.
x_train, y_train = iter(trainDataLoader).next()
x_test, y_test = iter(testDataLoader).next()

for epoch in range(20):
  for i,data in enumerate(trainDataLoader,0):
      X_train, Y_train = data
      optimizer.zero_grad()
      output = net(X_train)
      loss = Loss(output,Y_train)
      loss.backward()
      optimizer.step()

  train_loss = Loss(net(x_train),y_train)
  train_loss_history.append(train_loss)
  test_loss = Loss(net(x_test),y_test)
  test_loss_history.append(test_loss)
  print('Training Loss =',train_loss,'Testing Loss =',test_loss)
```

```
Training Loss = tensor(0.6553, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.5727, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.5369, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.5308, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.5163, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.5147, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4942, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4836, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4996, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4846, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4743, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4787, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4767, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4697, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4724, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4465, grad_fn=<NllLossBackward>) Testing Loss = tensor(
```

```
Training Loss = tensor(0.4695, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4581, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4474, grad_fn=<NllLossBackward>) Testing Loss = tensor(
Training Loss = tensor(0.4446, grad_fn=<NllLossBackward>) Testing Loss = tensor(
```

```
plt.plot(range(20),train_loss_history,'-',linewidth=3,label='Train error')
plt.plot(range(20),test_loss_history,'-',linewidth=3,label='Test error')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f26943bd198>
```



Neat! Now let's evaluate our model accuracy on the entire dataset. The predicted class label for a given input image can computed by looking at the output of the neural network model and computing the index corresponding to the maximum activation. Something like

predicted_output = net(images) _, predicted_labels = torch.max(predicted_output,1)

```
predicted_output = net(images)
print(torch.max(predicted_output, 1))
fit = Loss(predicted_output, labels)
print(labels)
```

```
torch.return_types.max(
values=tensor([ 6.3237,  9.2878, 12.3574, 11.2712,  4.0627,  8.7971,  5.8120,  5
         2.3723,  6.8020,  5.5309,  4.4548,  4.8034,  7.6182,  8.0388,  8.1988,
         4.7154,  5.1468,  5.6541,  9.9059,  4.4855,  5.8125,  9.8484,  6.5423,
        11.3322,  3.7887,  7.2196,  6.7671,  7.8740,  4.8908, 10.3487,  5.0584,
         5.1107,  6.9286,  6.7489,  4.9269,  7.7297,  4.1808,  8.5116, 10.3495,
         8.7489, 12.2093,  4.7805,  8.7050,  8.2103,  3.8250,  6.5184,  8.4097,
         4.1752,  6.6815,  6.7731,  3.5929,  2.6959,  6.4896,  4.5608,  7.0841,
         8.3861,  6.3503,  7.4553,  4.8277,  8.3544,  7.9292,  7.8718,  4.6105],
       grad_fn=<MaxBackward0>),
indices=tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 5, 3, 4, 1, 2, 6, 8, 0, 6, 7
```

```
            1, 2, 6, 0, 9, 3, 8, 8, 3, 3, 8, 0, 7, 5, 7, 9, 0, 1, 3, 9, 6, 7, 2, 1,
            2, 6, 6, 2, 5, 6, 2, 2, 8, 4, 8, 0, 7, 7, 8, 5]))
    tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 7, 3, 4, 1, 2, 4, 8, 0, 2, 5, 7, 9,
            1, 4, 6, 0, 9, 3, 8, 8, 3, 3, 8, 0, 7, 5, 7, 9, 6, 1, 3, 7, 6, 7, 2, 1,
            2, 2, 4, 4, 5, 8, 2, 2, 8, 4, 8, 0, 7, 7, 8, 5])
```

```
def evaluate(dataloader):
  # Q4.6 Implement a function here that evaluates training and testing accuracy.
  # Here, accuracy is measured by probability of successful classification.
  # ...
  # ...
  correct = 0
  total = 0

  with torch.no_grad():
        for data in dataloader:
            images, labels = data

            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

  return 100 * correct / total

print('Train Accuracy = %0.2f, Test Accuracy = %0.2f' % (evaluate(trainDataLoader), ev
```

```
    Train Accuracy = 84.92, Test Accuracy = 83.28
```