ECGR 4106 Real-time Machine Learning
Homework 1
Jonathon Nguyen
ID: 801093003
Github link: https://github.com/nguyjd/ecgr-4106-homework-1

Problem 1
The goal of the problem is take pictues of the different items that are red, green, and blue. Once that is done, the image is loaded into the program and converted into a tensor. The mean will be found for each picture and each of the color channel for each picture.

For problem 1 part a, I took five picture. One of a mask box, then my blue slides, A green food lion bag, old green textbook and a red cough drop bag.

For problem 1 part b, The code that converts the images to a tensor is in IN[2]. I had to use np.ascontiguousarray() because when it was reading it in, it was using a negitive stride and that is not allowed in the torch api.

For problem 1 part c: In IN[3], The tensor data type is converted to a float so I can take the mean of the image. Also in IN[3], I take the mean of each pictures.

Problem 1 part d: The code that finds the mean of each color channel for all the images are in IN[4], IN[5], IN[6], IN[7], IN[8]. I used indexing to get the channels and printed all the data out.

If we only consider the highest color channel mean for each image, I say that we can identify red, green and blue items just off the channel average.

Overall mean: 116.16

Red channel mean: 97.59

Green channel mean: 131.46

Blue channel mean: 119.61

This is my blue slide.

Overall mean: 120.75

Red channel mean: 116.03

Green channel mean: 117.91

Blue channel mean: 128.32

This is my green food lion bag.

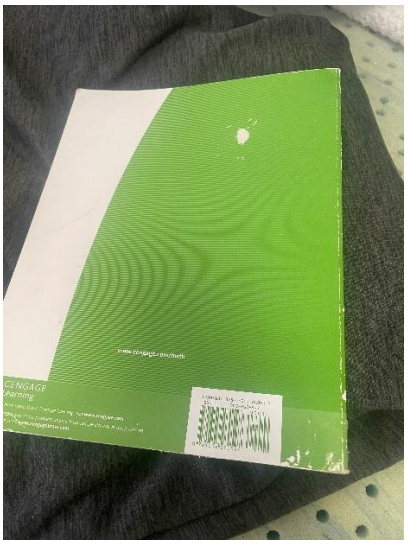
Overall mean: 107.55

Red channel mean: 108.63

Green channel mean: 120.59

Blue channel mean: 93.41

This is my old green textbook



Overall mean: 117.69

Red channel mean: 121.312

Green channel mean: 137.38

Blue channel mean: 94.36

This is a red cough drop bag.

Overall mean: 128.55

Red channel mean: 175.66

Green channel mean: 116.61

Blue channel mean: 93.39

Problem 2

The goal of the problem is convert the temperature prediction example from a linear model to a non-linear model. The training loop will have to be modify as well as the amount of parameters in the model.

For problem 2 a, the training loop was modified in IN[10] by unpacking an additional parameter. `w2, w1, b = params` Then the model function was change to take in the parameter IN[9]. The autograd feature was also used to help with calculating the gradent.
Also, a if statement was used to print out the loss every 500 epoch.

For problem 2 B, Four different training loops was created in IN[12], IN[13] IN[14] IN[15] with learning rates of 0.1, 0.01, 0.001, 0.0001 respectively.
The training loops with learning rates of 0.1, 0.01, 0.001 had to large of a learning rate that is exploded in values and loss. However the training loop with a learning rate of 0.0001 was able to produce something and it ended a loss of 3.861745

```
# Call the training loop with a learning ra
LEARNING_RATE = 0.1
params = torch.tensor([1.0, 1.0, 0.0], requ
training_loop(NUM_EPOCHS, LEARNING_RATE, pa
```

```
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

tensor([nan, nan, nan], requires_grad=True)
```

Learning rate of 0.1

```
# Call the training loop with a learning rat
LEARNING_RATE = 0.01
params = torch.tensor([1.0, 1.0, 0.0], requi
training_loop(NUM_EPOCHS, LEARNING_RATE, par
```

```
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

tensor([nan, nan, nan], requires_grad=True)
```

Learning rate of 0.01

```
# Call the training loop with a learning ra
LEARNING_RATE = 0.001
params = torch.tensor([1.0, 1.0, 0.0], requ
training_loop(NUM_EPOCHS, LEARNING_RATE, pa
```

```
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

tensor([nan, nan, nan], requires_grad=True)
```

Learning rate of 0.001

```
# Call the training loop with a learning
LEARNING_RATE = 0.0001
params = torch.tensor([1.0, 1.0, 0.0], r
params_new = training_loop(NUM_EPOCHS, L
params_new
```

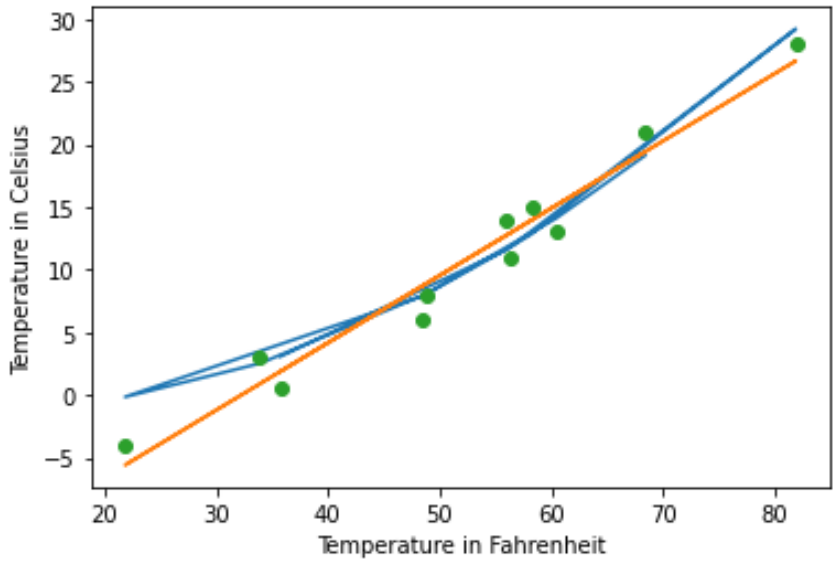
```
Epoch 500, Loss 10.708596
Epoch 1000, Loss 8.642084
Epoch 1500, Loss 7.171004
Epoch 2000, Loss 6.123477
Epoch 2500, Loss 5.377228
Epoch 3000, Loss 4.845287
Epoch 3500, Loss 4.465787
Epoch 4000, Loss 4.194724
Epoch 4500, Loss 4.000801
Epoch 5000, Loss 3.861745

tensor([ 0.5570, -0.8881, -0.8753], requ
```

Learning rate of 0.0001

The training loop with a learning rate of 0.0001 produce the tensor with “0.5570, -0.8881, -0.8753” and this is the best params for the non linear model and it is the best model.

This is the visualizion of the non-linear model against the linear model over the input values.



The blue line is the non-linear model and the orange line is the linear model. The green dots are the real values.

Looking at this graph, the non-linear model is **worse** then the linear model. The reason I say this is because the ends of the blue lines are not fitting to the data while the orange line fits better. The performance in the middle is about the same between the two lines.

Problem 3

The goal of problem 3 is to make a linear regression model that predicts the housing prices based on area, bedroom, bathrooms, stories and parking. The model is coded in IN[17]

Problem 3 A: The loss function is the squared difference error. The training loop takes from problem 2 with the autograd method. The only thing that really changed from problem 2 is the variable names and the model input parameters.

The preprocessing is much different from problem 2. All the values are single digit except the area. The area is normalized so the gradient does not get crazy.

Problem 3 B: Four different training loops was created in IN[20], IN[21] IN[22] IN[23] with learning rates of 0.1, 0.01, 0.001, 0.0001 respectively. The loss was reported every 500 epochs. The loss was great in all case.

```
# Call the training loop with a learning
LEARNING_RATE = 0.1
params = torch.tensor([1.0, 1.0, 1.0, 1.
training_loop(NUM_EPOCHS, LEARNING_RATE,
```

```
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan

tensor([nan, nan, nan, nan, nan, nan], r
```

Learning rate of 0.1

```
# Call the training loop with a learning
LEARNING_RATE = 0.01
params = torch.tensor([1.0, 1.0, 1.0, 1.
params_new = training_loop(NUM_EPOCHS, L
```

```
Epoch 500, Loss 1567288983552.000000
Epoch 1000, Loss 1543507673088.000000
Epoch 1500, Loss 1535782420480.000000
Epoch 2000, Loss 1532843261952.000000
Epoch 2500, Loss 1531714076672.000000
Epoch 3000, Loss 1531279835136.000000
Epoch 3500, Loss 1531112849408.000000
Epoch 4000, Loss 1531048755200.000000
Epoch 4500, Loss 1531023982592.000000
Epoch 5000, Loss 1531014414336.000000
```

Learning rate of 0.01

```
# Call the training loop with a learning rate o
LEARNING_RATE = 0.001
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0,
training_loop(NUM_EPOCHS, LEARNING_RATE, params
```

```
Epoch 500, Loss 1792176291840.000000
Epoch 1000, Loss 1691893235712.000000
Epoch 1500, Loss 1652624588800.000000
Epoch 2000, Loss 1627743453184.000000
Epoch 2500, Loss 1609898131456.000000
Epoch 3000, Loss 1596611493888.000000
Epoch 3500, Loss 1586484871168.000000
Epoch 4000, Loss 1578599841792.000000
Epoch 4500, Loss 1572331061248.000000
Epoch 5000, Loss 1567245336576.000000

tensor([ 682388.0000,  415422.4688, 1112289.250
         764936.3125], requires_grad=True)
```

Learning rate of 0.001

```
# Call the training loop with a learning ra
LEARNING_RATE = 0.0001
params = torch.tensor([1.0, 1.0, 1.0, 1.0,
training_loop(NUM_EPOCHS, LEARNING_RATE, pa
```

```
Epoch 500, Loss 3214041415680.000000
Epoch 1000, Loss 2189241876480.000000
Epoch 1500, Loss 2069992964096.000000
Epoch 2000, Loss 2001922162688.000000
Epoch 2500, Loss 1947758493696.000000
Epoch 3000, Loss 1903602434048.000000
Epoch 3500, Loss 1867387764736.000000
Epoch 4000, Loss 1837514752000.000000
Epoch 4500, Loss 1812715143168.000000
Epoch 5000, Loss 1791981256704.000000

tensor([455855.6875, 793864.2500, 559174.25
        331979.4688], requires_grad=True)
```

Learning rate of 0.0001

Problem 3 C: The best linear model was from the training loop with a learning rate of 0.01. This is the visuaization of the model verus the actual values. There is a lot of peaks and doesn't really follow the data too. This could be due to the way I normalized the data.