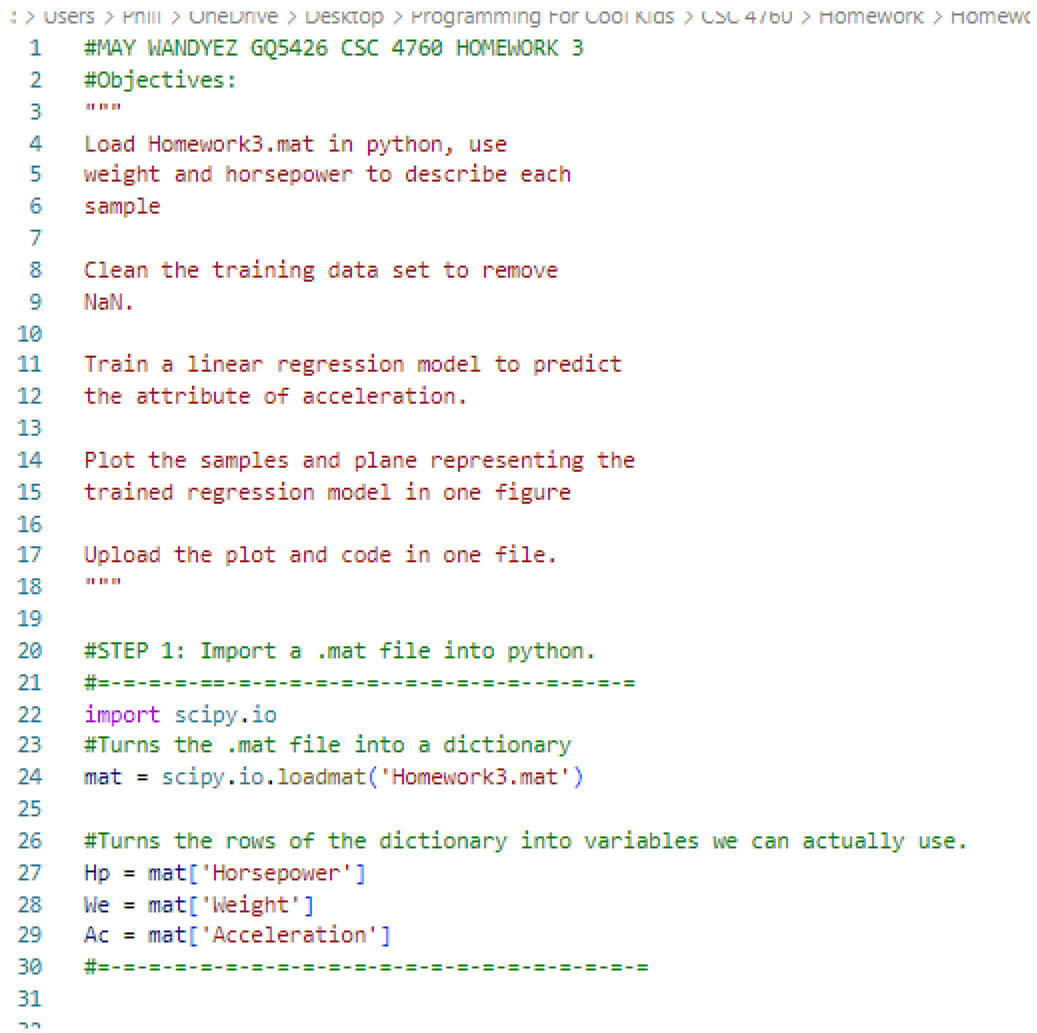
CSC 4760 – Homework 3
May Wandyez
Gq5426
2/14/2024

CSC 4760 Homework 3

Instructions:

1.) Train a linear regression model to predict the attribute of acceleration based on weight and horsepower from the provided data set in 'Homework3.mat'
2.) Plot the samples and the plane representing the trained regression model together in one figure.
3.) Upload the plot and your code in one file.

Results:

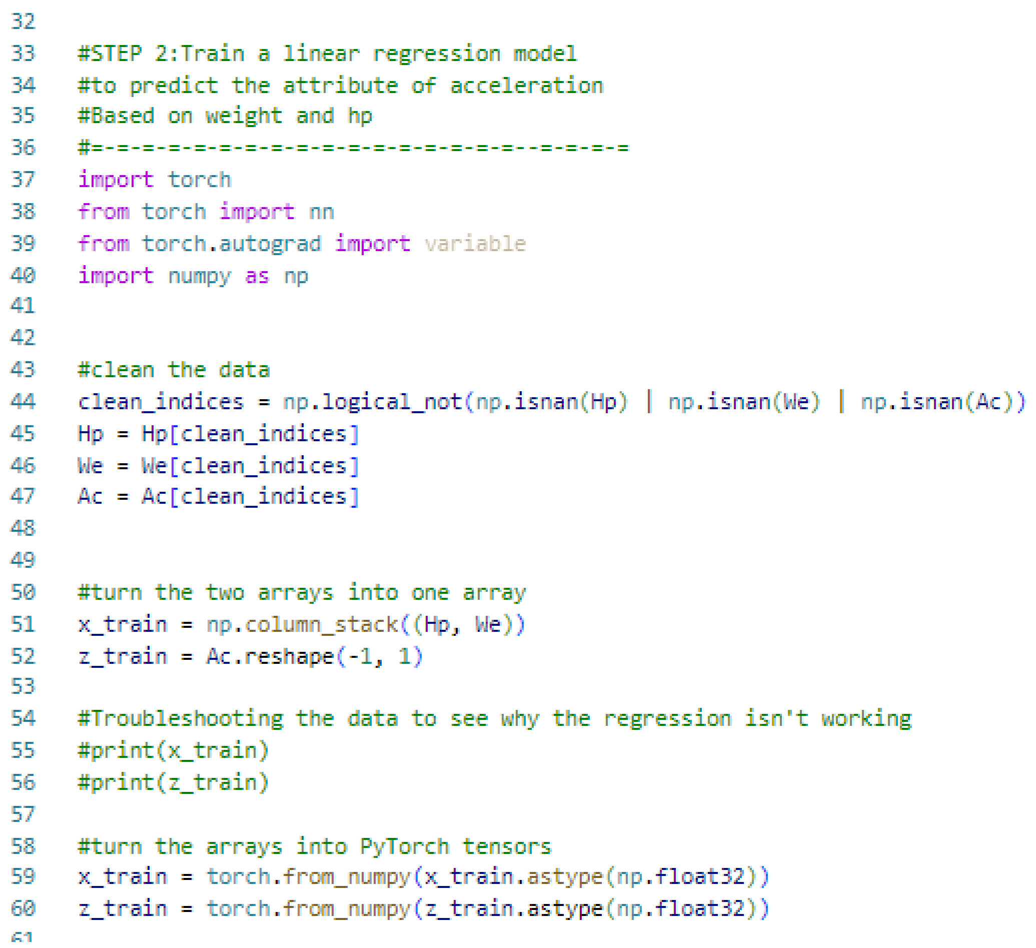


: > Users > Phill > OneDrive > Desktop > Programming For Cool Kids > CSC 4760 > Homework > Homewc

```
1   #MAY WANDYEZ GQ5426 CSC 4760 HOMEWORK 3
2   #Objectives:
3   """
4   Load Homework3.mat in python, use
5   weight and horsepower to describe each
6   sample
7
8   Clean the training data set to remove
9   NaN.
10
11  Train a linear regression model to predict
12  the attribute of acceleration.
13
14  Plot the samples and plane representing the
15  trained regression model in one figure
16
17  Upload the plot and code in one file.
18  """
19
20  #STEP 1: Import a .mat file into python.
21  #=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
22  import scipy.io
23  #Turns the .mat file into a dictionary
24  mat = scipy.io.loadmat('Homework3.mat')
25
26  #Turns the rows of the dictionary into variables we can actually use.
27  Hp = mat['Horsepower']
28  We = mat['Weight']
29  Ac = mat['Acceleration']
30  #=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
31
```

Figure 1 – Lines 1-30:

The section of code depicted in figure 1 describes the initial objectives for the lab, and also contains the code for importing the .mat file into python. Python does not natively support .mat files, which means you need to install scipy or some other converter. Scipy converts the .mat file into a dictionary, which is why the data values for Horsepower, Weight, and Acceleration are using the name for that column to create the data set.

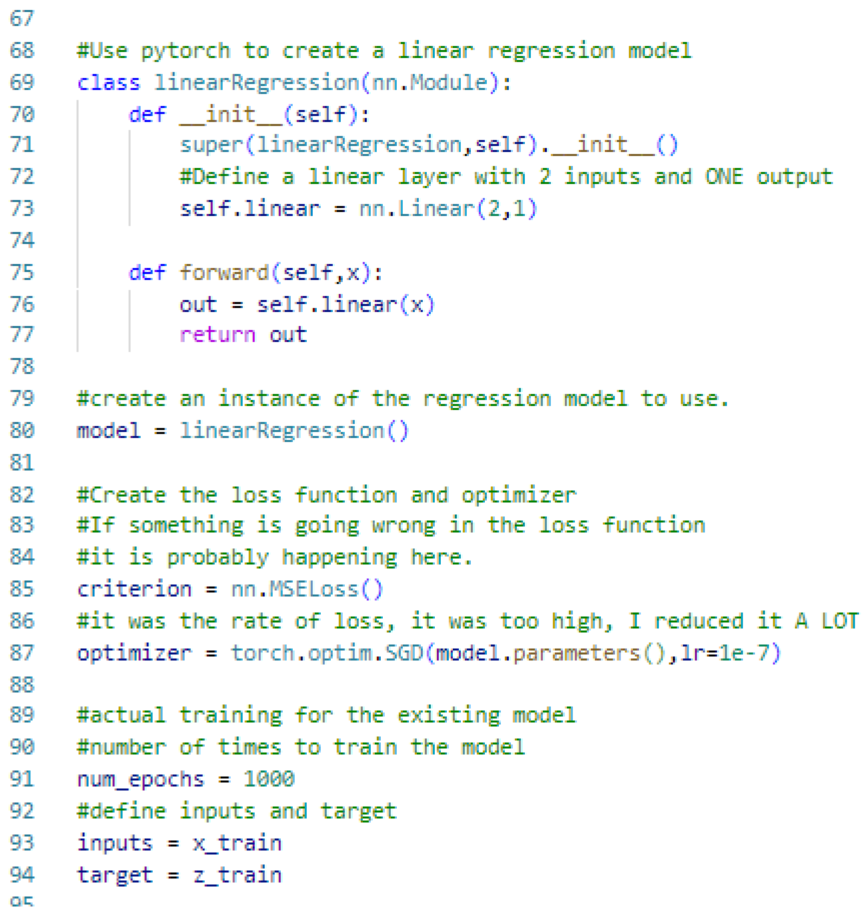


```
32
33  #STEP 2:Train a linear regression model
34  #to predict the attribute of acceleration
35  #Based on weight and hp
36  #=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
37  import torch
38  from torch import nn
39  from torch.autograd import variable
40  import numpy as np
41
42
43  #clean the data
44  clean_indices = np.logical_not(np.isnan(Hp) | np.isnan(We) | np.isnan(Ac))
45  Hp = Hp[clean_indices]
46  We = We[clean_indices]
47  Ac = Ac[clean_indices]
48
49
50  #turn the two arrays into one array
51  x_train = np.column_stack((Hp, We))
52  z_train = Ac.reshape(-1, 1)
53
54  #Troubleshooting the data to see why the regression isn't working
55  #print(x_train)
56  #print(z_train)
57
58  #turn the arrays into PyTorch tensors
59  x_train = torch.from_numpy(x_train.astype(np.float32))
60  z_train = torch.from_numpy(z_train.astype(np.float32))
61
```

Figure 2 – Code lines 33-60

The section of code depicted in figure 2 cleans the data and converts it into pytorch tensors. The data is cleaned to get rid of any not a number result. The model that was presented as an example in class could handle multiple inputs, but only if they were combined as one array as a column_stack so that it could be later passed to the training model.

CSC 4760 – Homework 3
May Wandyez
Gq5426
2/14/2024

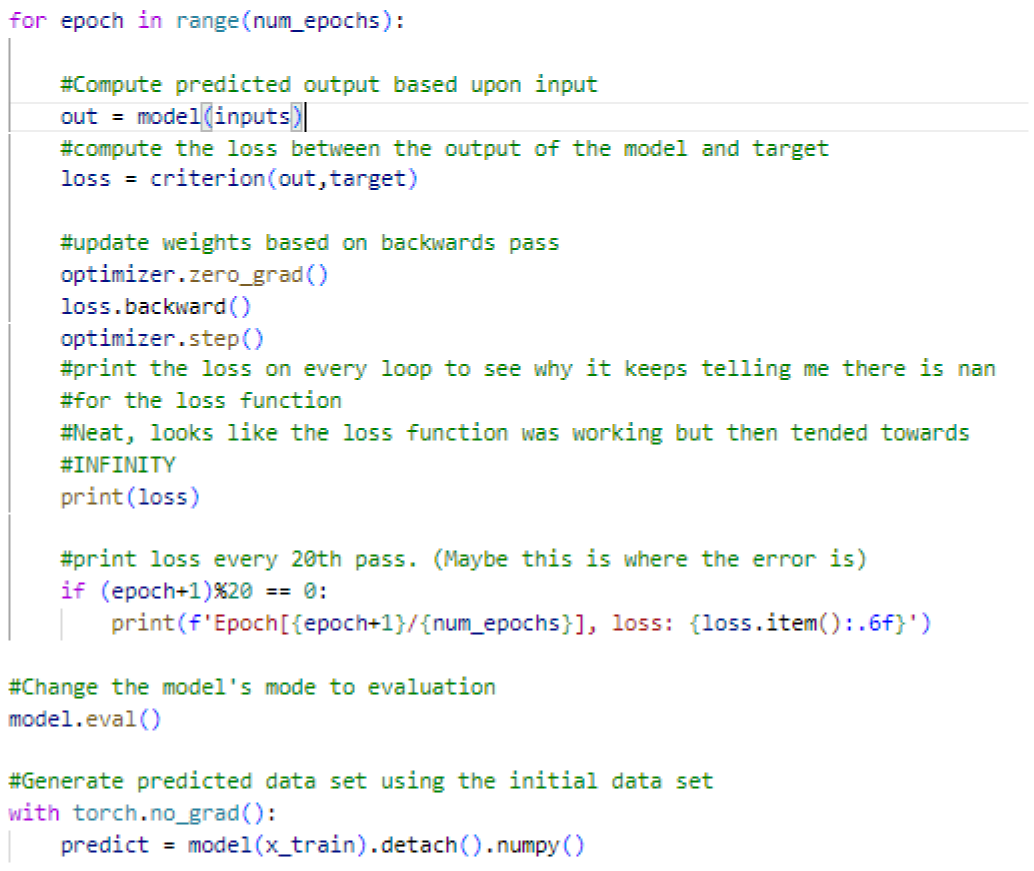


```
67
68  #Use pytorch to create a linear regression model
69  class linearRegression(nn.Module):
70      def __init__(self):
71          super(linearRegression,self).__init__()
72          #Define a linear layer with 2 inputs and ONE output
73          self.linear = nn.Linear(2,1)
74
75      def forward(self,x):
76          out = self.linear(x)
77          return out
78
79  #create an instance of the regression model to use.
80  model = linearRegression()
81
82  #Create the loss function and optimizer
83  #If something is going wrong in the loss function
84  #it is probably happening here.
85  criterion = nn.MSELoss()
86  #it was the rate of loss, it was too high, I reduced it A LOT
87  optimizer = torch.optim.SGD(model.parameters(),lr=1e-7)
88
89  #actual training for the existing model
90  #number of times to train the model
91  num_epochs = 1000
92  #define inputs and target
93  inputs = x_train
94  target = z_train
```

Figure 3 – Code lines 68-94

The code depicted in figure 3 defines the class for the linear regression model, this involves initializing the class and making sure that it can use torch for the linear regression. The model is then created as an instance of the class, and so is the type of loss function, and the rate of loss. It should be noted that the original loss rate tended to overcorrect and lead to infinity loss, as a result the loss rate was adjusted to a much lower number. The training inputs were then defined for the actual training of the model.

CSC 4760 – Homework 3
May Wandyez
Gq5426
2/14/2024

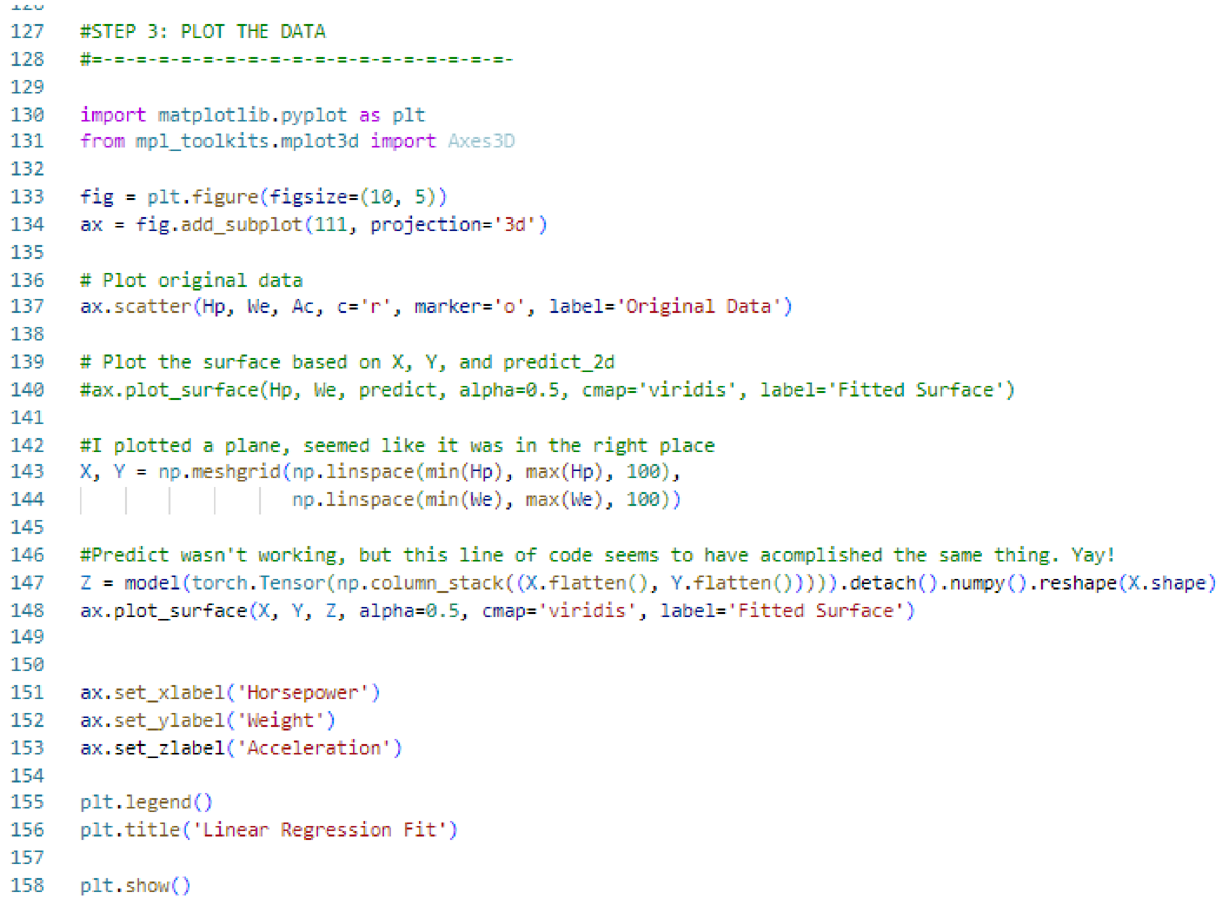


```
for epoch in range(num_epochs):

    #Compute predicted output based upon input
    out = model(inputs)
    #compute the loss between the output of the model and target
    loss = criterion(out,target)

    #update weights based on backwards pass
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    #print the loss on every loop to see why it keeps telling me there is nan
    #for the loss function
    #Neat, looks like the loss function was working but then tended towards
    #INFINITY
    print(loss)

    #print loss every 20th pass. (Maybe this is where the error is)
    if (epoch+1)%20 == 0:
        print(f'Epoch[{epoch+1}/{num_epochs}], loss: {loss.item():.6f}')

#Change the model's mode to evaluation
model.eval()

#Generate predicted data set using the initial data set
with torch.no_grad():
    predict = model(x_train).detach().numpy()
```

Figure 4 – Code lines 96-122

Figure 4 depicts the actual training of the model. The model class is used to generate inputs based on the training data, the loss is then calculated based on the different between the output from the model and the training data. The weights are then adjusted. For utility in the comments I printed the loss each time – as for some loss rates it was trending towards infinity. The model was then used to generate training data – which was then not used because it caused issues in plotting, so instead the model was used directly for plotting – and it tended to work!

CSC 4760 – Homework 3
May Wandyez
Gq5426
2/14/2024



```
127  #STEP 3: PLOT THE DATA
128  #=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
129
130  import matplotlib.pyplot as plt
131  from mpl_toolkits.mplot3d import Axes3D
132
133  fig = plt.figure(figsize=(10, 5))
134  ax = fig.add_subplot(111, projection='3d')
135
136  # Plot original data
137  ax.scatter(Hp, We, Ac, c='r', marker='o', label='Original Data')
138
139  # Plot the surface based on X, Y, and predict_2d
140  #ax.plot_surface(Hp, We, predict, alpha=0.5, cmap='viridis', label='Fitted Surface')
141
142  #I plotted a plane, seemed like it was in the right place
143  X, Y = np.meshgrid(np.linspace(min(Hp), max(Hp), 100),
144                     np.linspace(min(We), max(We), 100))
145
146  #Predict wasn't working, but this line of code seems to have acomplished the same thing. Yay!
147  Z = model(torch.Tensor(np.column_stack((X.flatten(), Y.flatten())))).detach().numpy().reshape(X.shape)
148  ax.plot_surface(X, Y, Z, alpha=0.5, cmap='viridis', label='Fitted Surface')
149
150
151  ax.set_xlabel('Horsepower')
152  ax.set_ylabel('Weight')
153  ax.set_zlabel('Acceleration')
154
155  plt.legend()
156  plt.title('Linear Regression Fit')
157
158  plt.show()
```

Figure 5 – Code lines 127-158

Figure 5 depicts the code that actually plots the data. This is pretty simple, a 3d data plane is generated and the original data is populated into the data plane. However due to weird formatting, the predict function was not outputting correctly – instead the model was used to generate new data points based on flattened version of horsepower and acceleration – and then a plane was generated. This did not cause any issues, as the dictionary generated initial versions of those arrays was a list of lists where the lowest level of list only contained one value.

CSC 4760 – Homework 3
May Wandyez
Gq5426
2/14/2024

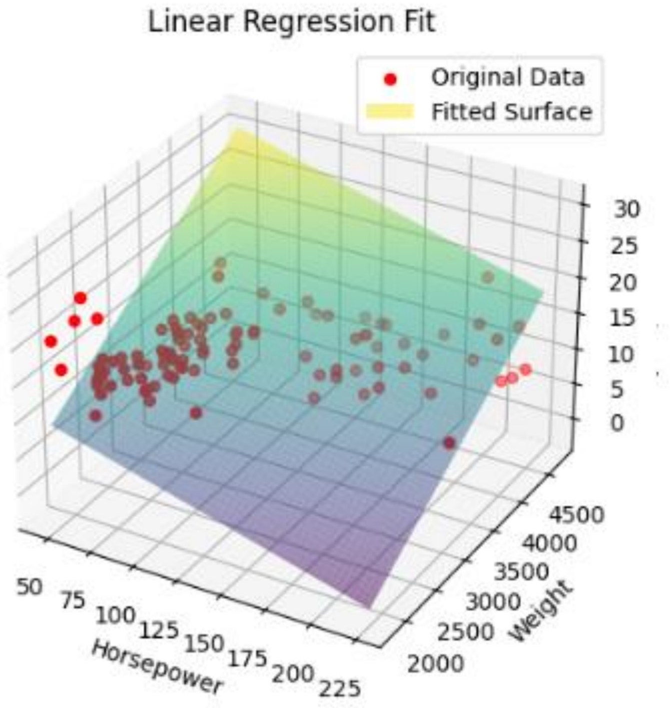


Figure 6 – The Generated Plot