EE6310

Problem Set 6

This problem set will guide you to write the linear regression algorithm in python and train a linear model to predict the heat of formation of linear alkane, $CH_3(CH_2)_xCH3$. Below are the data points.

Number of CH_2	Heat of formation [eV]
2	-0.87024
3	-1.08469
4	-1.30122
5	-1.52085
6	-1.73116
7	-1.94561
8	-2.16213
9	-2.36519
10	-2.58689
11	-2.80031
12	-3.01372
13	-3.22714
14	-3.44056
15	-3.67573
16	-3.88396
17	-4.0808
18	-4.29526
19	-4.50764
20	-4.72209
32	-7.21678

In python code, the following code can be copied and pasted to make a numpy array.

```
import numpy as np
data = np.array([[2,-0.87024],[3,-1.08469],[4,-1.30122],[5,-
1.52085],[6,-1.73116],[7,-1.94561],[8,-2.16213],[9,-2.36519],[10,-
2.58689],[11,-2.80031],[12,-3.01372],[13,-3.22714],[14,-
3.44056],[15,-3.67573],[16,-3.88396],[17,-4.0808],[18,-
4.29526],[19,-4.50764],[20,-4.72209],[32,-7.21678]])
```

Follow the steps below to make the linear regression algorithm. Let's get started!

(a) Let's start with a simple numpy coding. Using the data array above, use the slicing to separate the data into *x* (feature) and *y* (target). In the data array, the first column contains the *x*, and the second column contains the *y*. The answer should have a form of:

```
x = …
y = …
```

(b) Define the function that predicts the $\hat{y}$ given *w*, *b*, and *x*. Remember the equation:

$$f_{w,b}(x) = \hat{y} = wx + b$$

The input to this function are *w*, *b*, and *x*. Thus, the function should have a form below

```
def linear(w,b,x):
    # your code
    return y_hat
```

You can use the for loop, but you will find that utilizing the broadcasting will significantly boost the computation speed. If the code is correct, you should get the result in the comment (written after hashtag #).

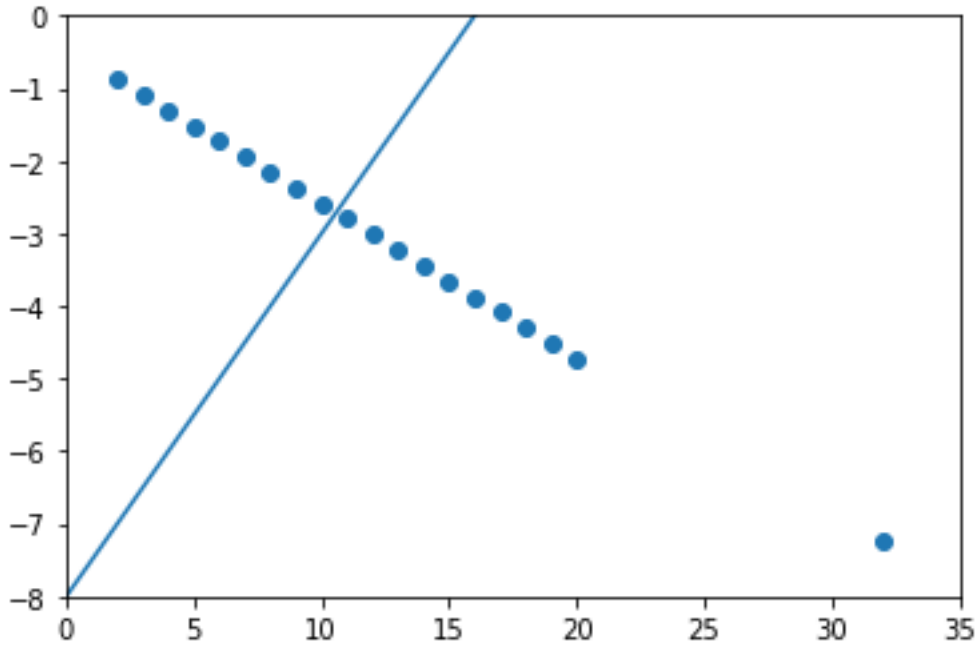
```
yhat = function(0.5,-8,x)
print(yhat.sum()) # You should get -39.5
```

(c) Let's see how the w and b value we selected perform. Write a code that uses matplotlib.pyplot.scatter (plt.scatter if you imported matplotlib.pyplot as plt) to plot the data points (x,y). Use the matplotlib.pyplot.xlim (plt.xlim) and matplotlib.pyplot.ylim (plt.ylim) to limit the plot range to [0,35] for x, and [-8,0] for y. Look up Google for the function syntax! For plotting the function, calculate the $\hat{y}$ at $x = 0$ and $x = 35$ using the linear function above, and use matplotlib.pyplot.plot (plt.plot) to draw the line. The input should be *w*, *b*, *x*, and *y*.

```
import matplotlib.pyplot as plt
def draw(w,b,x,y):
    # your code
```

The following code should draw a plot below:

```
draw(0.5,-8,x,y)
```



If you followed up to here, brilliant! If you are struggling, work with your colleagues together. I encourage working together. Don't just copy other's work. Be ready to explain the code. ChatGPT is also your friend here.

(d) Now the figure shows that the model isn't exactly the best figure here. We need to implement the gradient descent to improve the model. First, let's make the function that calculates the cost function, so we can keep track of the goodness of the model:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

Your function should take *w*, *b*, *x* and *y*. You will realize that you can use the linear function to calculate the $\hat{y}$. It may be easier to write the code with for loop first. Try using the broadcasting for faster speed. *m* can be calculated using the shape of the numpy array. If you feel fancy, you can also utilize mean() function of the numpy array.

```
def cost_function(w,b,x,y):
    # your code here
    return J
```

If the code is correct, you should get the 13.098 for the following code.:

```
print(cost_function(0.5,-8,x,y))
# output should be 13.098
```

(e) Time to implement the gradient descent. Remember the algorithm was:

Repeat until convergence

$$w_{\text{tmp}} = w - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

$$b_{\text{tmp}} = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)$$

$$w = w_{\text{tmp}}{}'$$

$$b = b_{\text{tmp}}{}'$$

Implement this algorithm by writing a function that takes in the *w*, *b*, *x*, *y*, and alpha, and returns the new *w* and *b*.

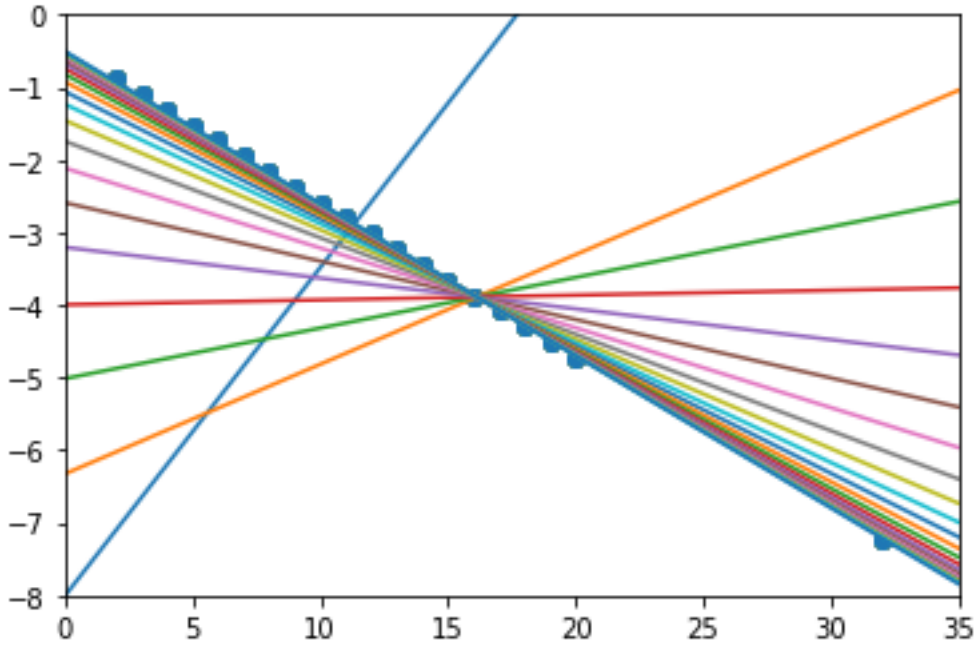
```
def gradient_descent(w,b,x,y,alpha):
    # your code
    return w_new, b_new
```

If your code works, you should get 12.54 for the following code.

```
w,b = gradient_descent(0.5,-8,x,y,0.01)
print(w,b) # 0.021578610000000054 -8.010465985
print(cost_function(w,b,x,y)) # 12.535610723807585
```

(f) Try using different alpha values here. At which alpha value does the cost_function increases for the first gradient descent step? What is an alpha value that decreases the cost function most for the first gradient descent step? You don't have to get too accurate in finding the exact value. Test the value between 0 and 1. To initially test the alpha value, try following values: 0.1, 0.03, 0.01, 0.003, 0.001, … which represents roughly, 10^{-1}, $10^{-1.5}$, 10^{-2}, $10^{-2.5}$, …. When you locate a good value from this testing, you can fine tune the alpha value.

(g) Great! Now, let's perform multiple gradient descent steps to optimize your model. Use the best alpha value you found in (f), and use the for loop to repeatedly update the w and b value from the initial w and b values of 0.5, and -8. If you want to see the progress, draw every 1000 gradient steps. The model should converge within about 20,000 steps. If you draw every 1000 gradient steps or so, you should get the following graph:



If you followed down to here, congratulation! you trained your first ML model.