assignment-02

April 2, 2020

[198]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
import random

random.seed(100) #generate random seed
torch.manual_seed(1) #generate random seed

data_number = 50 #number of data

x_data = [] #space for x data
y_data = [] #space for y data
cost_list = [] #space for cost
W_list = []
b_list = []
for i in range(data_number): #repeat by data_number
    y_data.append(random.randint(1,1000)) #randomly extract y's from 1 to 999
    x_data.append(int(y_data[i]/2)+random.randint(1,10)) #y/2 is randomly added␣
 ↪from 1 to 9


x_train = torch.FloatTensor([sorted(x_data)]) #X_data input
y_train = torch.FloatTensor([sorted(y_data)]) #Y_data input
print(x_train) #print x_train
print(y_train) #print y_train
```

```
tensor([[ 49.,  75.,  76.,  83.,  84.,  86.,  87.,  87.,  93., 133., 133., 136.,
         162., 176., 182., 196., 214., 231., 236., 255., 275., 284., 285., 289.,
         316., 316., 341., 341., 355., 363., 368., 371., 377., 381., 391., 402.,
         406., 412., 416., 418., 419., 441., 461., 463., 468., 472., 472., 495.,
         497., 505.]])
tensor([[ 83., 130., 145., 150., 152., 165., 167., 172., 185., 259., 261., 270.,
         317., 344., 347., 379., 413., 444., 466., 496., 546., 556., 568., 569.,
         618., 624., 668., 675., 699., 707., 723., 732., 735., 750., 780., 786.,
         794., 819., 820., 821., 834., 868., 912., 915., 923., 924., 936., 984.,
         987., 994.]])
```

[199]:

```
W = torch.zeros(1, requires_grad=True) #initialize W to 0. requires_grad=True␣
 ↪means variable value that keep changing by learning
b = torch.zeros(1, requires_grad=True) #nitialize W to 0. requires_grad=True␣
 ↪means variable value that keep changing by learning
```

[200]:

```
optimizer = optim.SGD([W, b], lr=1e-7) #pytorch optimzer library call

nb_epochs = 1000 # repeat gradient descent as many as we want.
for epoch in range(nb_epochs + 1): #repeat nb_epochs times

    # H(x)
    hypothesis = x_train * W + b

    # cost  = objective function
    cost = torch.mean((hypothesis - y_train) ** 2)/2
    cost_list.append(cost.item())
    # Gradient Descent algorithm
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    W_list.append(W.item())
    b_list.append(b.item())

    # print functions every hundred times
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, W.item(), b.item(), cost.item()
        ))
```

```
Epoch    0/1000 W: 0.021, b: 0.000 Cost: 203117.515625
Epoch  100/1000 W: 1.289, b: 0.004 Cost: 24733.380859
Epoch  200/1000 W: 1.732, b: 0.005 Cost: 3028.404053
Epoch  300/1000 W: 1.886, b: 0.005 Cost: 387.437347
Epoch  400/1000 W: 1.940, b: 0.005 Cost: 66.095879
Epoch  500/1000 W: 1.959, b: 0.005 Cost: 26.996191
Epoch  600/1000 W: 1.966, b: 0.005 Cost: 22.239185
Epoch  700/1000 W: 1.968, b: 0.005 Cost: 21.660322
Epoch  800/1000 W: 1.969, b: 0.005 Cost: 21.589867
Epoch  900/1000 W: 1.969, b: 0.005 Cost: 21.581261
Epoch 1000/1000 W: 1.969, b: 0.005 Cost: 21.580193
```

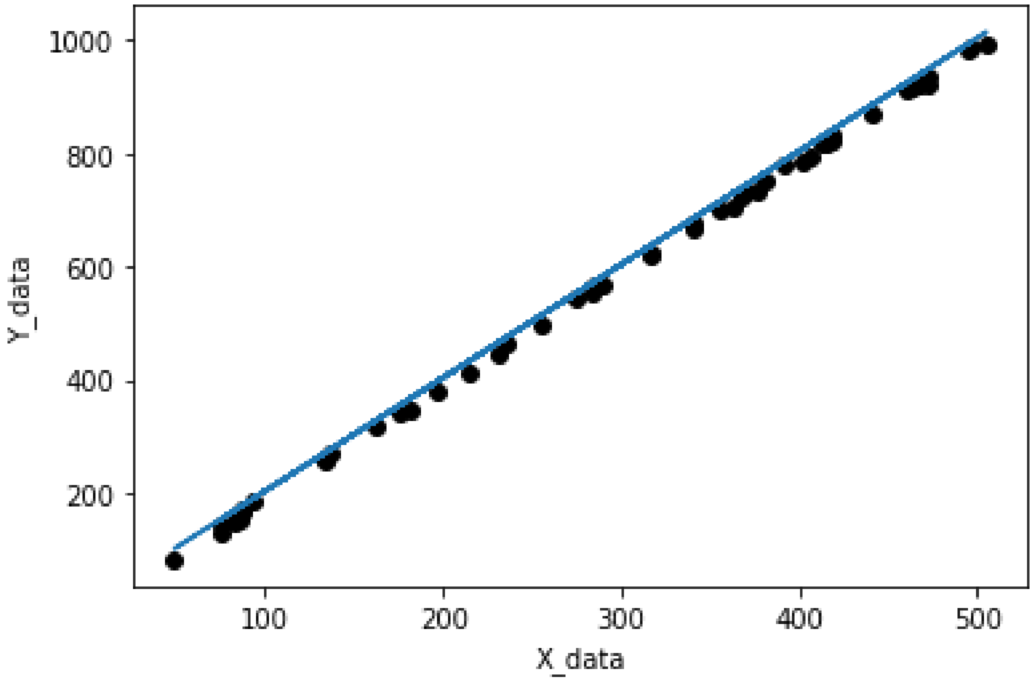
[201]:

```
Hy = [] #value of y that predicted by x_data
for i in x_data:
    Hy.append(int(W.item()*i+b.item())) #put value in Hy (value of y that␣
 ↪predicted by x_data)
print(x_data,end="\n\n")
print(y_data,end="\n\n")
print(Hy)
```

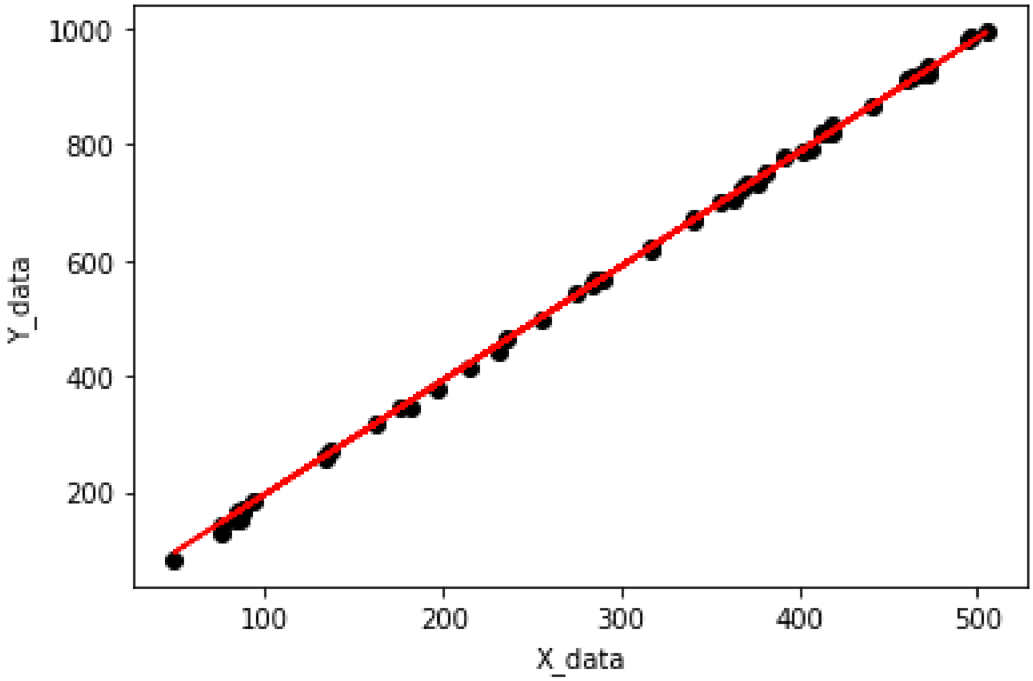
```
[83, 236, 368, 381, 231, 412, 275, 49, 136, 341, 176, 162, 495, 76, 463, 196,
497, 214, 289, 468, 84, 93, 316, 86, 87, 87, 472, 505, 416, 75, 285, 316, 377,
406, 371, 363, 182, 355, 284, 255, 391, 419, 461, 341, 133, 441, 402, 418, 472,
133]

[150, 466, 723, 750, 444, 820, 546, 83, 270, 675, 344, 317, 984, 145, 915, 379,
987, 413, 569, 923, 165, 185, 618, 152, 167, 172, 936, 994, 819, 130, 568, 624,
735, 794, 732, 707, 347, 699, 556, 496, 780, 834, 912, 668, 259, 868, 786, 821,
924, 261]

[163, 464, 724, 750, 454, 811, 541, 96, 267, 671, 346, 318, 974, 149, 911, 385,
978, 421, 569, 921, 165, 183, 622, 169, 171, 171, 929, 994, 819, 147, 561, 622,
742, 799, 730, 714, 358, 698, 559, 502, 769, 825, 907, 671, 261, 868, 791, 823,
929, 261]
```

[202]:

```
#1. Input data [2pt]
x=[]
for i in x_data:
    x.append(i*2+4.5) #fictitous Linear Regression function

plt.scatter(x_data,y_data,color='black') #print x_data and y_data to black
plt.plot(x_data,x, label="linear function") #print predicted fuction to graph

plt.xlabel("X_data") #name of label
plt.ylabel("Y_data") #name of label

plt.show() #show
```

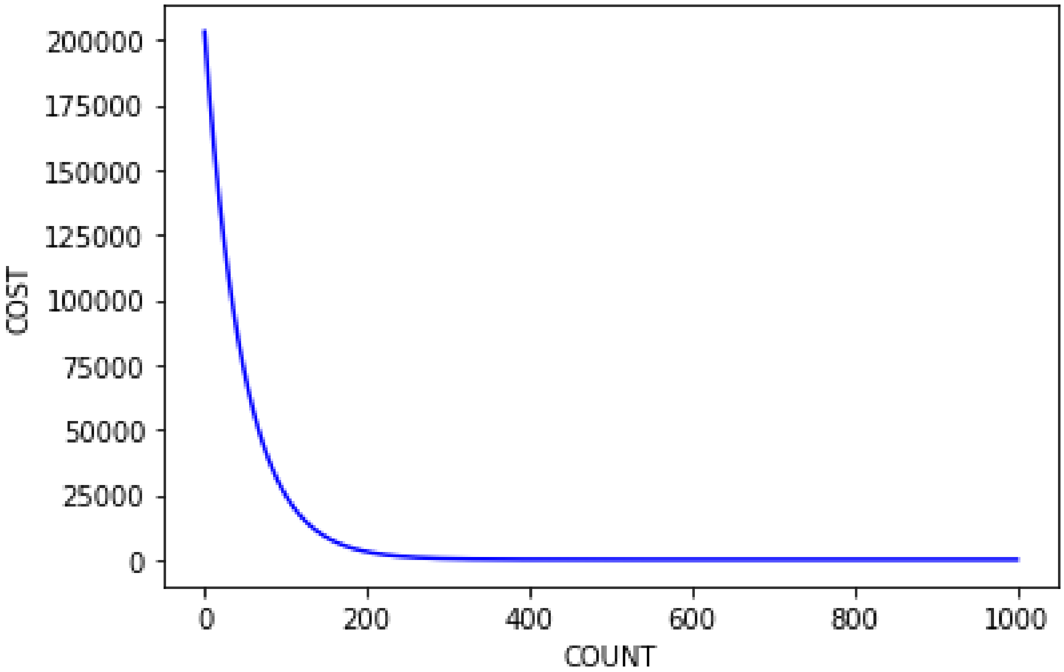


[203]:
```
#2.Plotting the output results [1pt]
plt.scatter(x_data,y_data,color='black') #print x_data and y_data to black␣
 ↪points
plt.plot(x_data,Hy, label="linear function",color='red') #print graph made from␣
 ↪learning data

plt.xlabel("X_data") #name of label
plt.ylabel("Y_data") #name of label
plt.show() #show
```



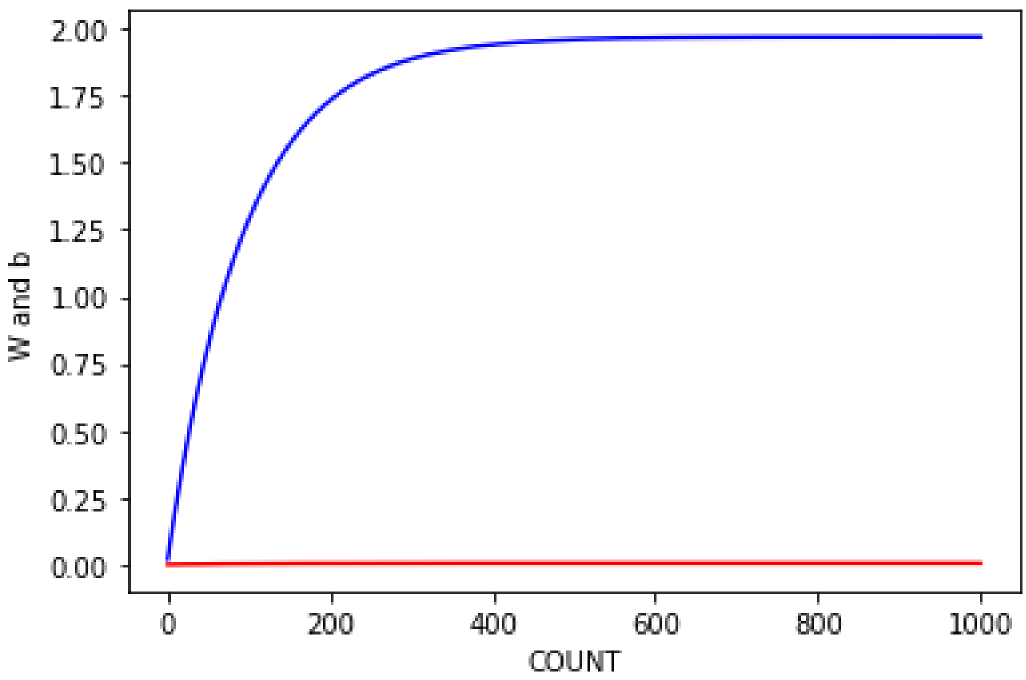
[204]:

```
plt.plot([i for i in range(len(cost_list))],cost_list,color='b') #After making␣
↪list by List Comprehension by number of cost_list,
#make a graph out of a cost_list and List Comprehension made eariler

plt.xlabel("COUNT") #name of label
plt.ylabel("COST") #name of label
plt.show() #show
```



[206]:

```
plt.plot([i for i in range(len(W_list))],W_list,color='b') #After making list␣
 ↪by List Comprehension by number of W_list,
#make a graph out of a W_list and List Comprehension made eariler
plt.plot([i for i in range(len(b_list))],b_list,color='red') #After making list␣
 ↪by List Comprehension by number of b_list,
#make a graph out of a b_list and List Comprehension made eariler

plt.xlabel("COUNT") #name of label
plt.ylabel("W and b") #name of label
plt.show() #show
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
W and b
0
200
400
600
800
1000
COUNT