Task-01

August 4, 2022

0.1 Task: 01

Let us consider the task of developing a linear model as follows:

Linear model: $y = w * x$

- In this task, you will create a linear model for given `x_data` and `y_data` (Supervised Learning).
- The main goal of this task is to make you familiar with python and provde a hands-on expereince.
- You have to create functions definitions for the forward-pass and loss-function and write a code snippet to find the optimal value of w
- Eventually we also plot the value of `w` against the difference in the prediction and actual value.

[1]:
```
#First let us import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
```

Let us make use of a randomly-created sample dataset

[2]:
```
#sample-dataset
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
```

0.2 Task: 01 - a

Implement the forward and loss functions

[5]:
```
#forward function to calculate y_pred for a given x according to the linear␣
↪model defined above
def forward(x):
    #implement the forward model to compute y_pred as w*x
    ## YOUR CODE STARTS HERE
    return (w*x)

    ## YOUR CODE ENDS HERE

#loss-function to compute the mean-squared error between y_pred and y_actual
def loss(y_pred, y_actual):
    #calculate the mean-squared-error between y_pred and y_actual
```

```
    ## YOUR CODE STARTS HERE

    return ((y_pred - y_actual)**2)
    ## YOUR CODE ENDS HERE
```

0.3 Task: 01 - b

Compute the loss for different values of w and identify the w with minimum loss value

[7]:
```
#initialize wieght and loss lists to monitor
weight_list=[]
loss_list=[]

for w in np.arange(0.0,4.1,0.1): # w can vary between 0 and 4 (both included)␣
 ↪with a step-size of 0.1
    print("\nWeight : ", w)
    total_loss=0
    count = 0
    for x, y in zip (x_data, y_data):
        #call the forward and loss functions to compute the loss value for the␣
 ↪given data-point pair
        ## YOUR CODE STARTS HERE
        yPred = forward(x)
        current_loss = loss(yPred , y)
        ## YOUR CODE ENDS HERE
        total_loss += current_loss
        count += 1

    #calculate the average mse-loss across three samples in our dataset
    ## YOUR CODE STARTS HERE
    avg_mse = 1/count * (total_loss)

    ## YOUR CODE ENDS HERE

    print("Average Mean Squared Error : ", avg_mse)
    weight_list.append(w)
    loss_list.append(avg_mse)
```

```
Weight :  0.0
Average Mean Squared Error :  18.666666666666664

Weight :  0.1
Average Mean Squared Error :  16.846666666666668

Weight :  0.2
Average Mean Squared Error :  15.120000000000001
```

```
Weight :  0.30000000000000004
Average Mean Squared Error :  13.486666666666665

Weight :  0.4
Average Mean Squared Error :  11.946666666666667

Weight :  0.5
Average Mean Squared Error :  10.5

Weight :  0.6000000000000001
Average Mean Squared Error :  9.146666666666663

Weight :  0.7000000000000001
Average Mean Squared Error :  7.886666666666665

Weight :  0.8
Average Mean Squared Error :  6.719999999999999

Weight :  0.9
Average Mean Squared Error :  5.646666666666666

Weight :  1.0
Average Mean Squared Error :  4.666666666666666

Weight :  1.1
Average Mean Squared Error :  3.7799999999999985

Weight :  1.2000000000000002
Average Mean Squared Error :  2.986666666666665

Weight :  1.3
Average Mean Squared Error :  2.2866666666666657

Weight :  1.4000000000000001
Average Mean Squared Error :  1.6799999999999993

Weight :  1.5
Average Mean Squared Error :  1.1666666666666665

Weight :  1.6
Average Mean Squared Error :  0.7466666666666659

Weight :  1.7000000000000002
Average Mean Squared Error :  0.4199999999999995

Weight :  1.8
Average Mean Squared Error :  0.18666666666666648
```

```
Weight :  1.9000000000000001
Average Mean Squared Error :  0.046666666666666586

Weight :  2.0
Average Mean Squared Error :  0.0

Weight :  2.1
Average Mean Squared Error :  0.046666666666666835

Weight :  2.2
Average Mean Squared Error :  0.18666666666666698

Weight :  2.3000000000000003
Average Mean Squared Error :  0.4200000000000005

Weight :  2.4000000000000004
Average Mean Squared Error :  0.7466666666666679

Weight :  2.5
Average Mean Squared Error :  1.1666666666666665

Weight :  2.6
Average Mean Squared Error :  1.6800000000000008

Weight :  2.7
Average Mean Squared Error :  2.2866666666666693

Weight :  2.8000000000000003
Average Mean Squared Error :  2.986666666666668

Weight :  2.9000000000000004
Average Mean Squared Error :  3.780000000000003

Weight :  3.0
Average Mean Squared Error :  4.666666666666666

Weight :  3.1
Average Mean Squared Error :  5.646666666666668

Weight :  3.2
Average Mean Squared Error :  6.720000000000003

Weight :  3.3000000000000003
Average Mean Squared Error :  7.886666666666668

Weight :  3.4000000000000004
Average Mean Squared Error :  9.14666666666667
```

```
Weight :  3.5
Average Mean Squared Error :  10.5

Weight :  3.6
Average Mean Squared Error :  11.946666666666669

Weight :  3.7
Average Mean Squared Error :  13.486666666666673

Weight :  3.8000000000000003
Average Mean Squared Error :  15.120000000000005

Weight :  3.9000000000000004
Average Mean Squared Error :  16.84666666666667

Weight :  4.0
Average Mean Squared Error :  18.666666666666664
```

0.4 Visualize the logs

[8]:

```python
plt.plot(weight_list, loss_list)
plt.ylabel('Loss')
plt.xlabel('Weight (w)')
plt.show()
```