



IBM Developer SKILLS NETWORK

Linear Regression 1D: Training One Parameter

Objective

- How to create cost or criterion function using MSE (Mean Square Error).

Table of Contents

In this lab, you will train a model with PyTorch by using data that you created. The model only has one parameter: the slope.

- [Make Some Data](#)
- [Create the Model and Cost Function \(Total Loss\)](#)
- [Train the Model](#)

Estimated Time Needed: **20 min**

Preparation

The following are the libraries we are going to use for this lab.

In [1]:

```
# These are the libraries will be used for this lab.  
  
import numpy as np  
import matplotlib.pyplot as plt
```

The class `plot_diagram` helps us to visualize the data space and the parameter space during training and has nothing to do with PyTorch.

In [2]:

```
# The class for plotting

class plot_diagram():

    # Constructor
    def __init__(self, X, Y, w, stop, go = False):
        start = w.data
        self.error = []
        self.parameter = []
        self.X = X.numpy()
        self.Y = Y.numpy()
        self.parameter_values = torch.arange(start, stop)
        self.Loss_function = [criterion(forward(X), Y) for w.data in self.parameter_values]
        w.data = start

    # Executor
    def __call__(self, Yhat, w, error, n):
        self.error.append(error)
        self.parameter.append(w.data)
        plt.subplot(212)
        plt.plot(self.X, Yhat.detach().numpy())
        plt.plot(self.X, self.Y, 'ro')
        plt.xlabel("A")
        plt.ylim(-20, 20)
        plt.subplot(211)
        plt.title("Data Space (top) Estimated Line (bottom) Iteration " + str(n))
        plt.plot(self.parameter_values.numpy(), self.Loss_function)
        plt.plot(self.parameter, self.error, 'ro')
        plt.xlabel("B")
        plt.figure()

    # Destructor
    def __del__(self):
        plt.close('all')
```

Make Some Data

Import PyTorch library:

In [3]:

```
# Import the library PyTorch

import torch
```

Generate values from -3 to 3 that create a line with a slope of -3. This is the line you will estimate.

In [4]:

```
# Create the f(X) with a slope of -3

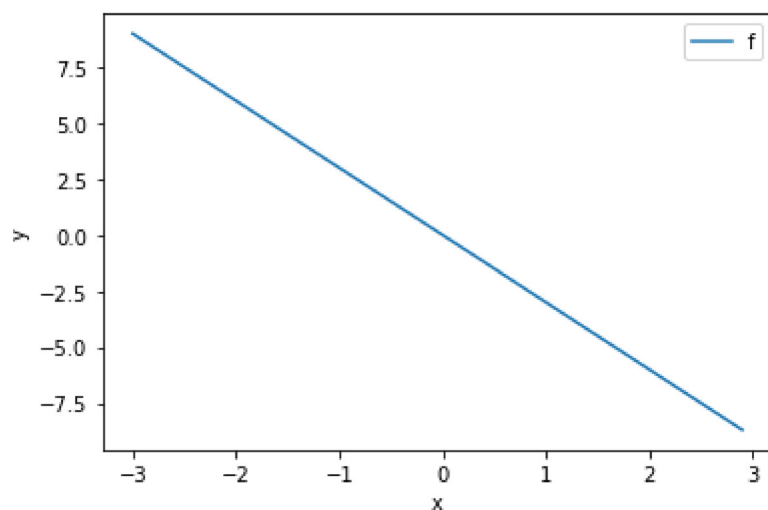
X = torch.arange(-3, 3, 0.1).view(-1, 1)
f = -3 * X
```

Let us plot the line.

In [5]:

```
# Plot the line with blue

plt.plot(X.numpy(), f.numpy(), label = 'f')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Let us add some noise to the data in order to simulate the real data. Use `torch.randn(X.size())` to generate Gaussian noise that is the same size as `X` and has a standard deviation of 0.1.

In [6]:

```
# Add some noise to f(X) and save it in Y

Y = f + 0.1 * torch.randn(X.size())
```

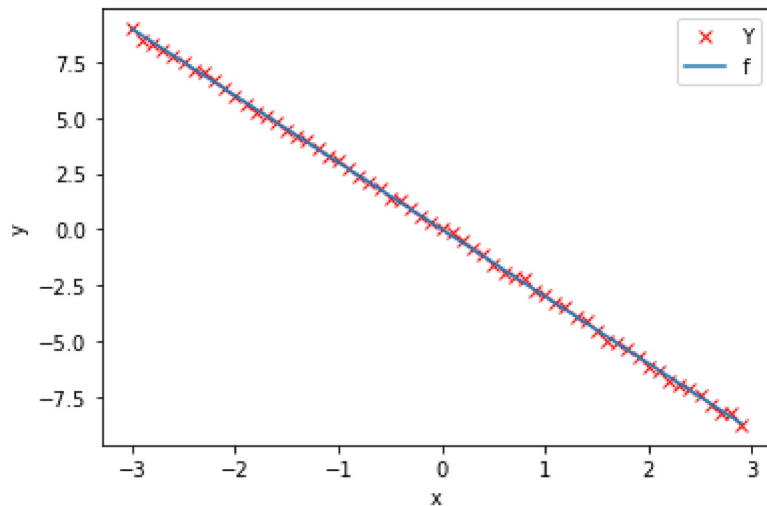
Plot the `Y` :

In [7]:

```
# Plot the data points

plt.plot(X.numpy(), Y.numpy(), 'rx', label = 'Y')

plt.plot(X.numpy(), f.numpy(), label = 'f')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Create the Model and Cost Function (Total Loss)

In this section, let us create the model and the cost function (total loss) we are going to use to train the model and evaluate the result.

First, define the forward function $y = w * x$. (We will add the bias in the next lab.)

In [8]:

```
# Create forward function for prediction

def forward(x):
    return w * x
```

Define the cost or criterion function using MSE (Mean Square Error):

In [9]:

```
# Create the MSE function for evaluate the result.

def criterion(yhat, y):
    return torch.mean((yhat - y) ** 2)
```

Define the learning rate `lr` and an empty list `LOSS` to record the loss for each iteration:

In [10]:

```
# Create Learning Rate and an empty list to record the loss for each iteration

lr = 0.1
LOSS = []
```

Now, we create a model parameter by setting the argument `requires_grad` to `True` because the system must learn it.

In [11]:

```
w = torch.tensor(-10.0, requires_grad = True)
```

Create a `plot_diagram` object to visualize the data space and the parameter space for each iteration during training:

In [12]:

```
gradient_plot = plot_diagram(X, Y, w, stop = 5)
```

Train the Model

Let us define a function for training the model. The steps will be described in the comments.

In [13]:

```
# Define a function for train the model

def train_model(iter):
    for epoch in range (iter):

        # make the prediction as we learned in the last lab
        Yhat = forward(X)

        # calculate the iteration
        loss = criterion(Yhat, Y)

        # plot the diagram for us to have a better idea
        gradient_plot(Yhat, w, loss.item(), epoch)

        # store the loss into list
        LOSS.append(loss.item())

        # backward pass: compute gradient of the loss with respect to all the learnable parameters
        loss.backward()

        # updata parameters
        w.data = w.data - lr * w.grad.data

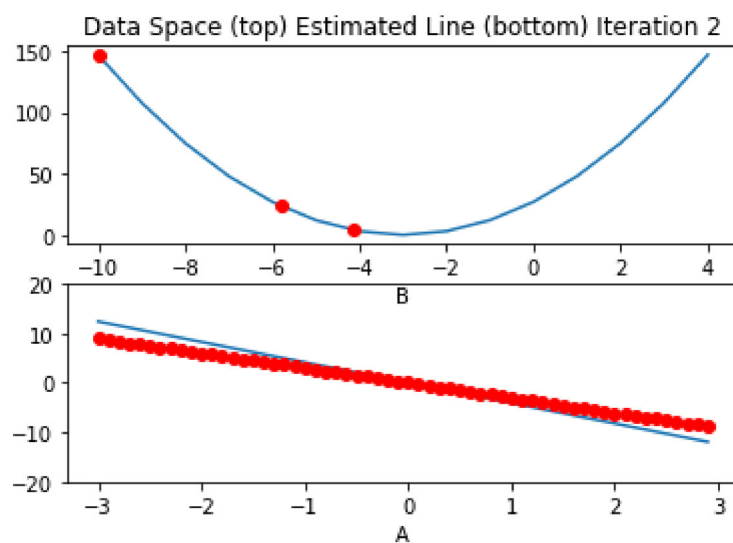
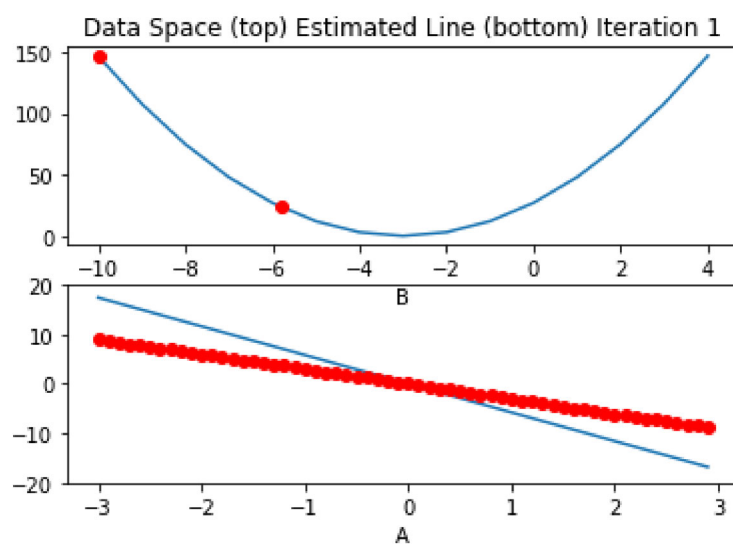
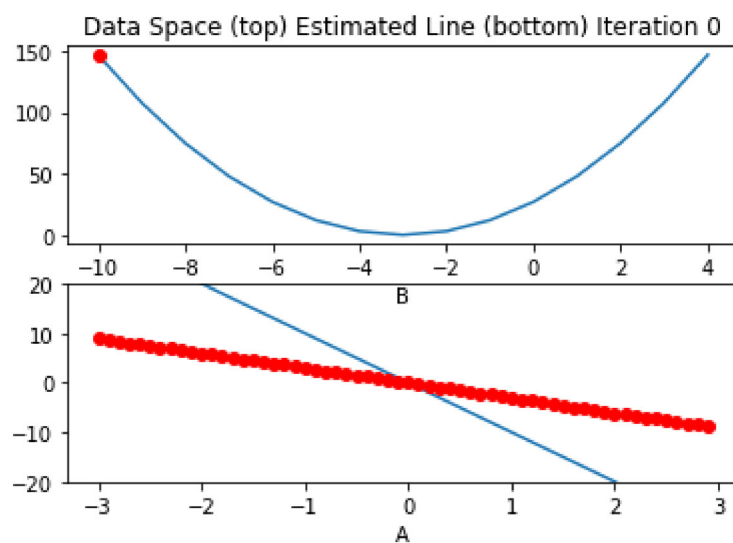
        # zero the gradients before running the backward pass
        w.grad.data.zero_()
```

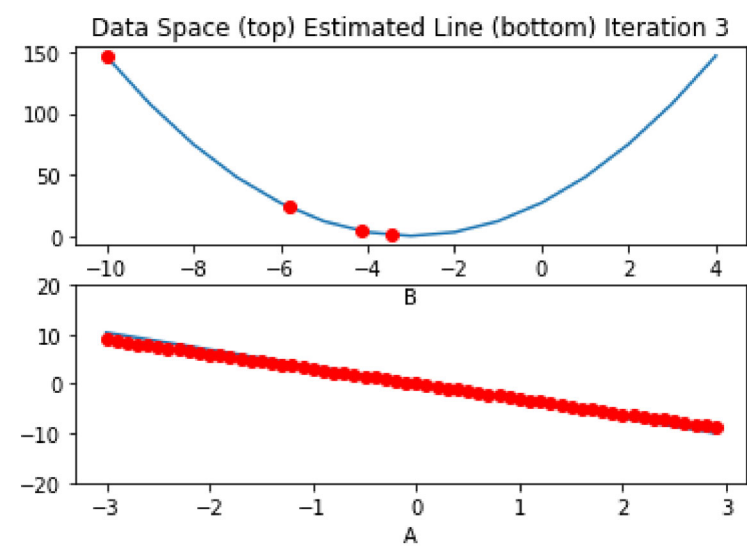
Let us try to run 4 iterations of gradient descent:

In [14]:

```
# Give 4 iterations for training the model here.
```

```
train_model(4)
```





<Figure size 432x288 with 0 Axes>

Plot the cost for each iteration:

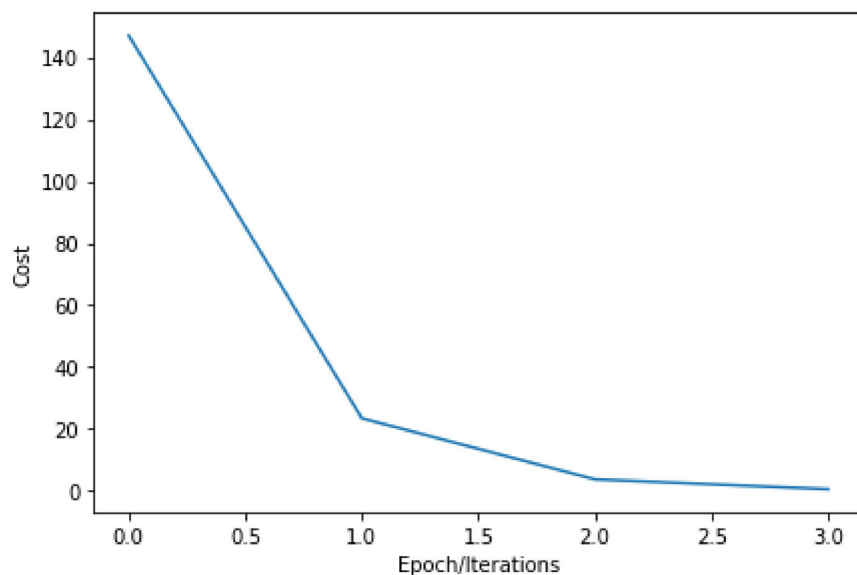
In [15]:

```
# Plot the loss for each iteration

plt.plot(LOSS)
plt.tight_layout()
plt.xlabel("Epoch/Iterations")
plt.ylabel("Cost")
```

Out[15]:

Text(23.875, 0.5, 'Cost')



Practice

Create a new learnable parameter w with an initial value of -15.0.

In []:

```
# Practice: Create w with the initial value of -15.0

# Type your code here
```

Double-click **here** for the solution.

Create an empty list `LOSS2` :

In []:

```
# Practice: Create LOSS2 list

# Type your code here
```

Double-click **here** for the solution.

Write your own `my_train_model` function with loss list `LOSS2` . And run it with 4 iterations.

In [17]:

```
# Practice: Create your own my_train_model

gradient_plot1 = plot_diagram(X, Y, w, stop = 15)
```

Double-click **here** for the solution.

<!--Empty Space for separating topics-->

Plot an overlay of the list `LOSS2` and `LOSS` .

In [18]:

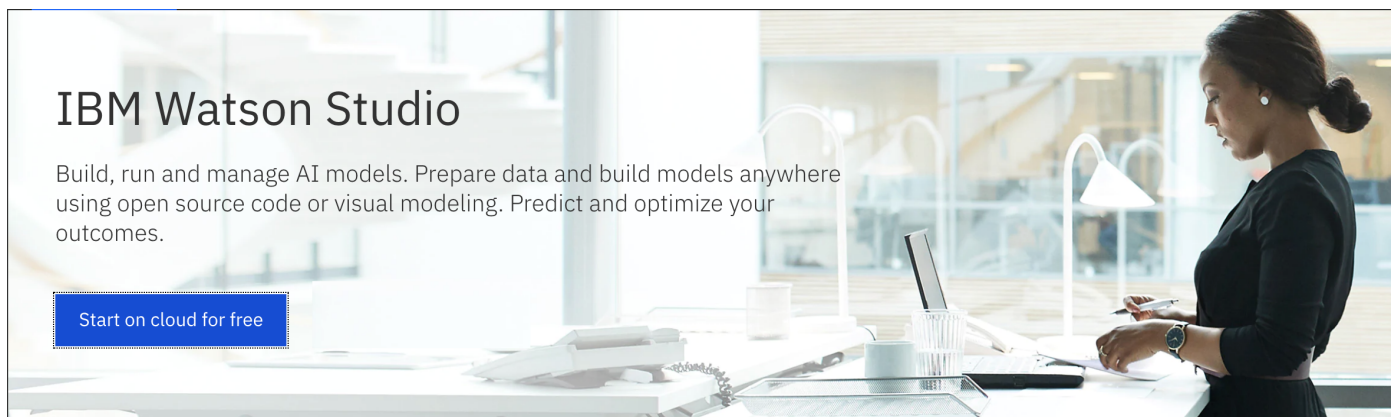
```
# Practice: Plot the list LOSS2 and LOSS

# Type your code here
```

Double-click **here** for the solution.

What does this tell you about the parameter value?

Double-click **here** for the solution.

An advertisement for IBM Watson Studio. The background is a blurred office scene with a woman in a black dress working at a desk with a laptop and a lamp. On the left, the text 'IBM Watson Studio' is displayed in a large, bold font. Below it, a paragraph describes the platform: 'Build, run and manage AI models. Prepare data and build models anywhere using open source code or visual modeling. Predict and optimize your outcomes.' At the bottom left, there is a blue button with the text 'Start on cloud for free'.

(https://dataplatform.cloud.ibm.com/registration/stepone?context=cpdaas&apps=data_science_experience,watson_machine_learning)

About the Authors:

[Joseph Santarcangelo](https://www.linkedin.com/in/joseph-s-50398b136/) (<https://www.linkedin.com/in/joseph-s-50398b136/>) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: [Michelle Carey](https://www.linkedin.com/in/michelleccarey/) (<https://www.linkedin.com/in/michelleccarey/>), [Mavis Zhou](https://www.linkedin.com/in/jiahui-mavis-zhou-a4537814a) (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-21	2.0	Shubham	Migrated Lab to Markdown and added to course repo in GitLab

© IBM Corporation 2020. All rights reserved.