2.2_linear_regression_one_parameter_v3

March 23, 2022

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.

[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.

[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:

[3]:
```
# Import the library PyTorch

import torch
```

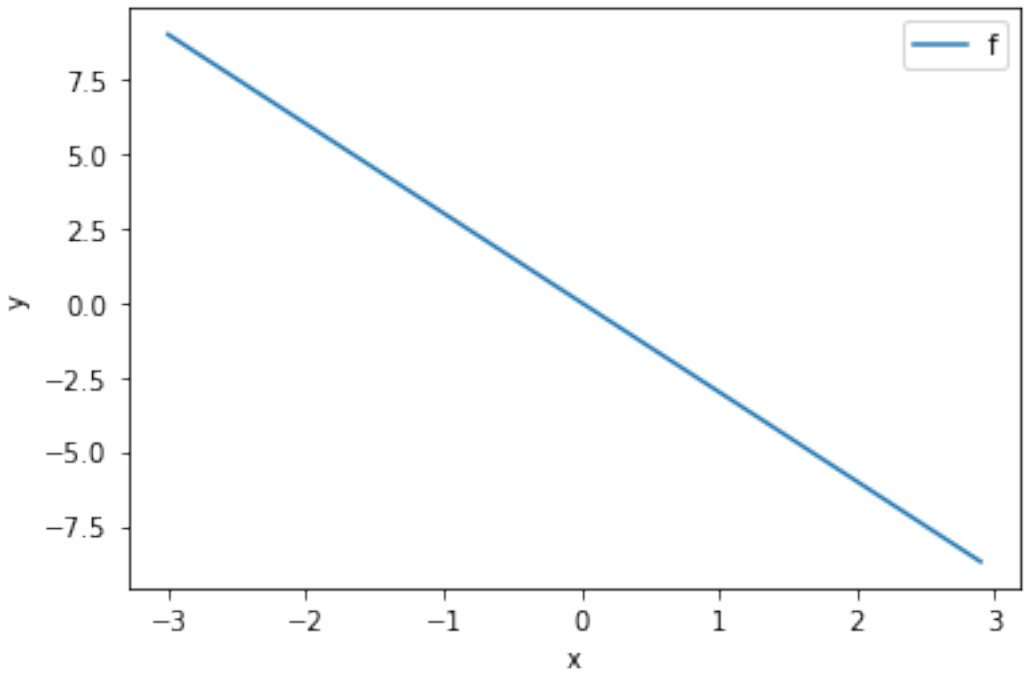
Generate values from -3 to 3 that create a line with a slope of -3. This is the line you will estimate.

[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.

[5]:
```
# Plot the line with blue

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

```
plt.show()
```

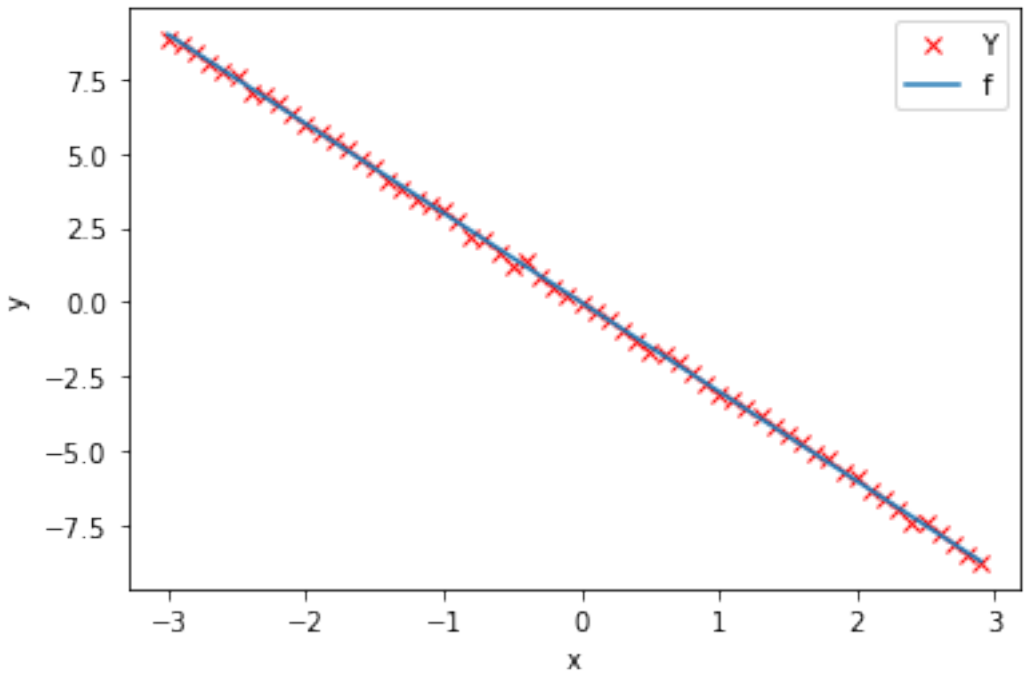


Let us add some noise to the data in order to simulate the real data. Use torch.randn(X.size()) to generate Gaussian noise that is the same size as X and has a standard deviation opf 0.1.

[6]:
```
# Add some noise to f(X) and save it in Y

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

Plot the Y:

[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 $y = wx$. (We will add the bias in the next lab.)

[8]:
```
# Create forward function for prediction

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

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

[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:

[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.

[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:

[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.

[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_()
```

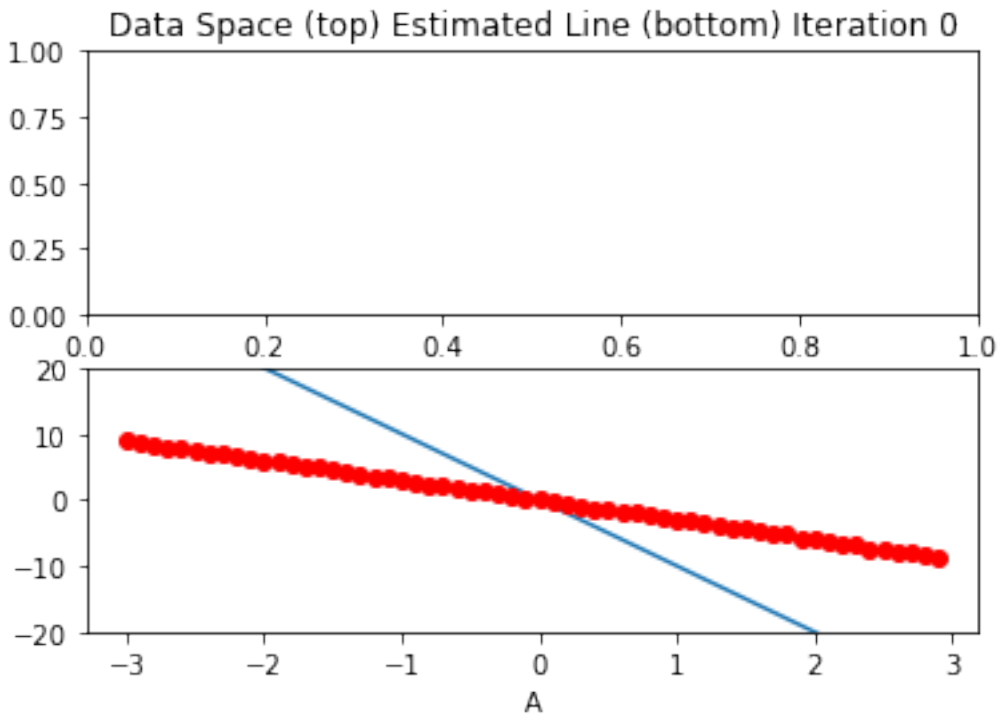
Let us try to run 4 iterations of gradient descent:

[14]:
```
# Give 4 iterations for training the model here.

train_model(4)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
```

```
/tmp/ipykernel_65/1280318540.py in <module>
      1 # Give 4 iterations for training the model here.
      2
----> 3 train_model(4)

/tmp/ipykernel_65/2558599578.py in train_model(iter)
     11
     12         # plot the diagram for us to have a better idea
---> 13         gradient_plot(Yhat, w, loss.item(), epoch)
     14
     15         # store the loss into list

/tmp/ipykernel_65/2928310021.py in __call__(self, Yhat, w, error, n)
     25         plt.subplot(211)
     26         plt.title("Data Space (top) Estimated Line (bottom) Iteration "
 ↪+ str(n))
---> 27         plt.plot(self.parameter_values.numpy(), self.Loss_function)
     28         plt.plot(self.parameter, self.error, 'ro')
     29         plt.xlabel("B")

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
 ↪plot(scalex, scaley, data, *args, **kwargs)
   2757     return gca().plot(
   2758         *args, scalex=scalex, scaley=scaley,
-> 2759         **({"data": data} if data is not None else {}), **kwargs)
   2760
   2761

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_axes.py in␣
 ↪plot(self, scalex, scaley, data, *args, **kwargs)
   1630         """
   1631         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1632         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1633         for line in lines:
   1634             self.add_line(line)

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_base.py in␣
 ↪__call__(self, data, *args, **kwargs)
    310                 this += args[0],
    311                 args = args[1:]
--> 312             yield from self._plot_args(this, kwargs)
    313
    314     def get_next_color(self):

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_base.py in␣
 ↪_plot_args(self, tup, kwargs, return_kwargs)
    486         if len(xy) == 2:
```

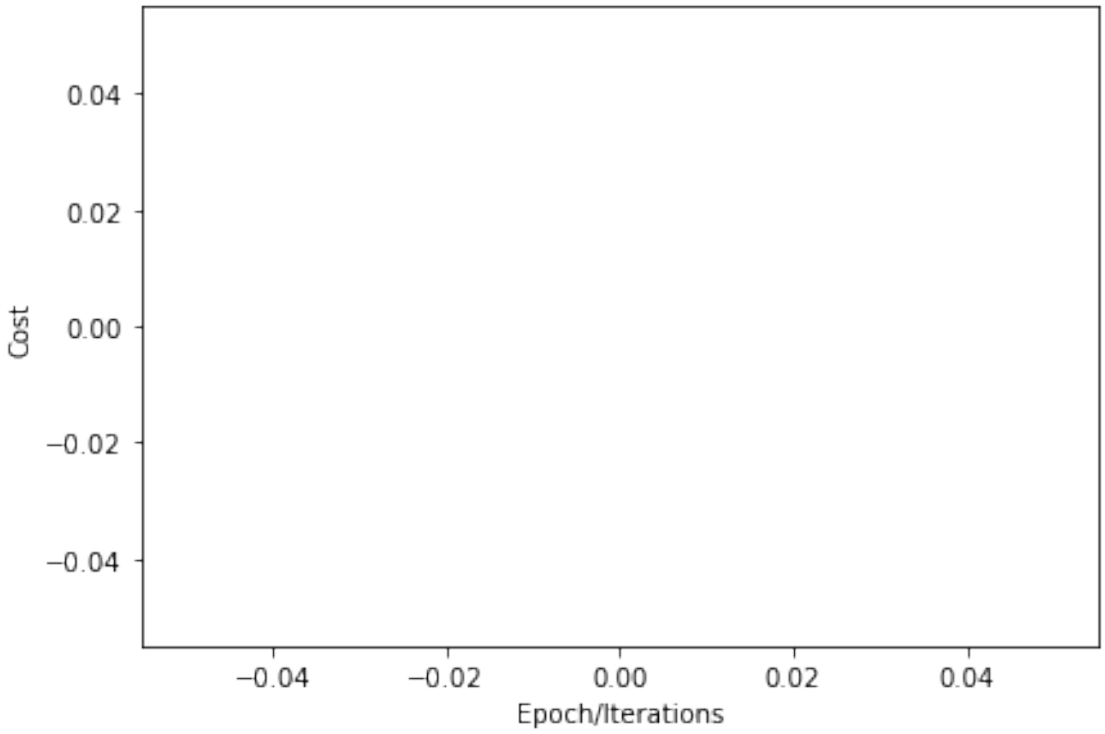
```
    487             x = _check_1d(xy[0])
--> 488             y = _check_1d(xy[1])
    489         else:
    490             x, y = index_of(xy[-1])

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/cbook/__init__.py in
 ↪_check_1d(x)
   1302     """Convert scalars to 1D arrays; pass-through arrays as is."""
   1303     if not hasattr(x, 'shape') or len(x.shape) < 1:
-> 1304         return np.atleast_1d(x)
   1305     else:
   1306         try:

<__array_function__ internals> in atleast_1d(*args, **kwargs)

~/conda/envs/python/lib/python3.7/site-packages/numpy/core/shape_base.py in␣
 ↪atleast_1d(*arys)
     63     res = []
     64     for ary in arys:
---> 65         ary = asanyarray(ary)
     66         if ary.ndim == 0:
     67             result = ary.reshape(1)

~/conda/envs/python/lib/python3.7/site-packages/torch/tensor.py in␣
 ↪__array__(self, dtype)
    490     def __array__(self, dtype=None):
    491         if dtype is None:
--> 492             return self.numpy()
    493         else:
    494             return self.numpy().astype(dtype, copy=False)

RuntimeError: Can't call numpy() on Variable that requires grad. Use var.
 ↪detach().numpy() instead.
```



Plot the cost for each iteration:

[15]:
```
# Plot the loss for each iteration

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

[15]:
```
Text(12.25, 0.5, 'Cost')
```



Practice

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

[18]:
```
# Practice: Create w with the inital value of -15.0

# Type your code here
w=torch.tensor(-15.0,requires_grad=True)
print(f'w: {w}')
```

```
w: -15.0
```

Double-click here for the solution.

Create an empty list LOSS2:

[19]:
```
# Practice: Create LOSS2 list

# Type your code here
LOSS2=[]
```

Double-click here for the solution.

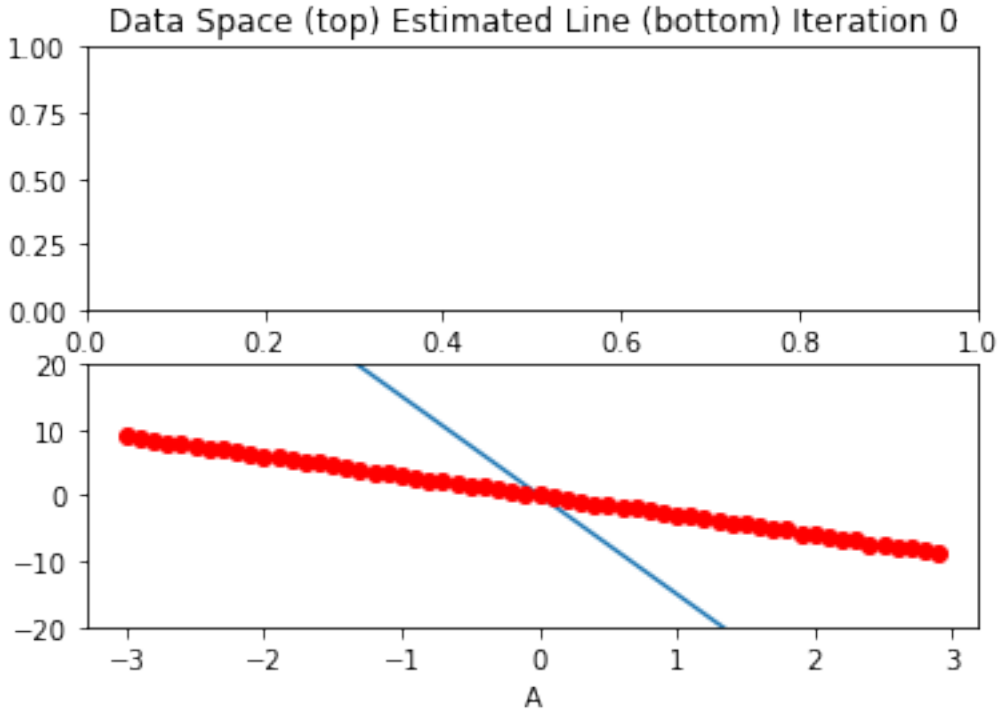
Write your own my_train_model function with loss list LOSS2. And run it with 4 iterations.

[21]:
```
# Practice: Create your own my_train_model

gradient_plot1 = plot_diagram(X, Y, w, stop = 15)
def my_train_model(iter):
    for epoch in range (iter):
        Yhat = forward(X)
        loss = criterion(Yhat,Y)
        gradient_plot1(Yhat, w, loss.item(), epoch)
        LOSS2.append(loss)
        loss.backward()
        w.data = w.data - lr * w.grad.data
        w.grad.data.zero_()


my_train_model(4)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_65/252167769.py in <module>
     11         w.data = w.data - lr * w.grad.data
     12         w.grad.data.zero_()
---> 13 my_train_model(4)

/tmp/ipykernel_65/252167769.py in my_train_model(iter)
      6         Yhat = forward(X)
      7         loss = criterion(Yhat,Y)
----> 8         gradient_plot1(Yhat, w, loss.item(), epoch)
      9         LOSS2.append(loss)
     10         loss.backward()

/tmp/ipykernel_65/2928310021.py in __call__(self, Yhat, w, error, n)
     25         plt.subplot(211)
     26         plt.title("Data Space (top) Estimated Line (bottom) Iteration "
 ↪+ str(n))
---> 27         plt.plot(self.parameter_values.numpy(), self.Loss_function)
     28         plt.plot(self.parameter, self.error, 'ro')
     29         plt.xlabel("B")

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
 ↪plot(scalex, scaley, data, *args, **kwargs)
   2757     return gca().plot(
   2758         *args, scalex=scalex, scaley=scaley,
-> 2759         **({"data": data} if data is not None else {}), **kwargs)
   2760
   2761
```

```
~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_axes.py in␣
 ↪plot(self, scalex, scaley, data, *args, **kwargs)
   1630         """
   1631         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1632         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1633         for line in lines:
   1634             self.add_line(line)

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_base.py in␣
 ↪__call__(self, data, *args, **kwargs)
    310                 this += args[0],
    311                 args = args[1:]
--> 312             yield from self._plot_args(this, kwargs)
    313
    314     def get_next_color(self):

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/axes/_base.py in␣
 ↪_plot_args(self, tup, kwargs, return_kwargs)
    486         if len(xy) == 2:
    487             x = _check_1d(xy[0])
--> 488             y = _check_1d(xy[1])
    489         else:
    490             x, y = index_of(xy[-1])

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/cbook/__init__.py in␣
 ↪_check_1d(x)
   1302     """Convert scalars to 1D arrays; pass-through arrays as is."""
   1303     if not hasattr(x, 'shape') or len(x.shape) < 1:
-> 1304         return np.atleast_1d(x)
   1305     else:
   1306         try:

<__array_function__ internals> in atleast_1d(*args, **kwargs)

~/conda/envs/python/lib/python3.7/site-packages/numpy/core/shape_base.py in␣
 ↪atleast_1d(*arys)
     63     res = []
     64     for ary in arys:
---> 65         ary = asanyarray(ary)
     66         if ary.ndim == 0:
     67             result = ary.reshape(1)

~/conda/envs/python/lib/python3.7/site-packages/torch/tensor.py in␣
 ↪__array__(self, dtype)
    490     def __array__(self, dtype=None):
    491         if dtype is None:
--> 492             return self.numpy()
    493         else:
```

```
    494             return self.numpy().astype(dtype, copy=False)

RuntimeError: Can't call numpy() on Variable that requires grad. Use var.
  ↪detach().numpy() instead.
```

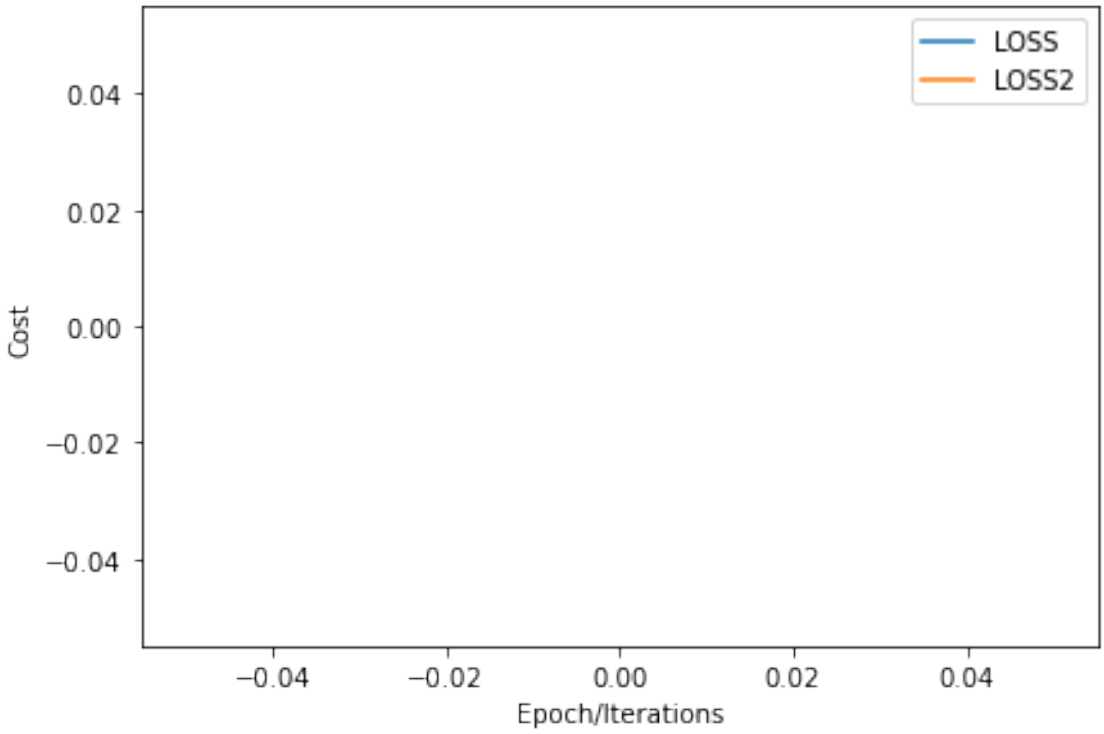


Double-click here for the solution.

Plot an overlay of the list LOSS2 and LOSS.

[22]:
```
# Practice: Plot the list LOSS2 and LOSS

# Type your code here
plt.plot(LOSS, label = "LOSS")
plt.plot(LOSS2, label = "LOSS2")
plt.tight_layout()
plt.xlabel("Epoch/Iterations")
plt.ylabel("Cost")
plt.legend()
```

[22]:
```
<matplotlib.legend.Legend at 0x7f73a0307c10>
```



Double-click here for the solution.

What does this tell you about the parameter value?

Double-click here for the solution.

About the Authors:

Joseph Santarcangelo 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, Mavis Zhou

0.1 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

##