2.3_training_slope_and_bias_v3

March 23, 2022

Linear regression 1D: Training Two Parameter

Objective

How to train the model and visualize the loss results.

Table of Contents

In this lab, you will train a model with PyTorch by using the data that we created. The model will have the slope and bias. And we will review how to make a prediction in several different ways by using PyTorch.

Make Some Data

Create the Model and Cost Function (Total Loss)

Train the Model

Estimated Time Needed: 20 min

Preparation

We'll need the following libraries:

[1]:

```python
# These are the libraries we are going to use in the lab.

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
```

The class plot_error_surfaces is just to help you visualize the data space and the parameter space during training and has nothing to do with PyTorch.

[2]:

```python
# The class for plot the diagram

class plot_error_surfaces(object):

    # Constructor
    def __init__(self, w_range, b_range, X, Y, n_samples = 30, go = True):
        W = np.linspace(-w_range, w_range, n_samples)
        B = np.linspace(-b_range, b_range, n_samples)
        w, b = np.meshgrid(W, B)
        Z = np.zeros((30,30))
        count1 = 0
        self.y = Y.numpy()
        self.x = X.numpy()
        for w1, b1 in zip(w, b):
            count2 = 0
            for w2, b2 in zip(w1, b1):
                Z[count1, count2] = np.mean((self.y - w2 * self.x + b2) ** 2)
                count2 += 1
            count1 += 1
        self.Z = Z
        self.w = w
        self.b = b
        self.W = []
        self.B = []
        self.LOSS = []
        self.n = 0
        if go == True:
            plt.figure()
            plt.figure(figsize = (7.5, 5))
            plt.axes(projection='3d').plot_surface(self.w, self.b, self.Z,␣
↪rstride = 1, cstride = 1,cmap = 'viridis', edgecolor = 'none')
            plt.title('Cost/Total Loss Surface')
            plt.xlabel('w')
            plt.ylabel('b')
            plt.show()
            plt.figure()
            plt.title('Cost/Total Loss Surface Contour')
            plt.xlabel('w')
            plt.ylabel('b')
            plt.contour(self.w, self.b, self.Z)
            plt.show()

    # Setter
    def set_para_loss(self, W, B, loss):
        self.n = self.n + 1
        self.W.append(W)
        self.B.append(B)
        self.LOSS.append(loss)

    # Plot diagram
    def final_plot(self):
        ax = plt.axes(projection = '3d')
        ax.plot_wireframe(self.w, self.b, self.Z)
        ax.scatter(self.W,self.B, self.LOSS, c = 'r', marker = 'x', s = 200,␣
↪alpha = 1)
        plt.figure()
        plt.contour(self.w,self.b, self.Z)
        plt.scatter(self.W, self.B, c = 'r', marker = 'x')
        plt.xlabel('w')
        plt.ylabel('b')
        plt.show()

    # Plot diagram
    def plot_ps(self):
        plt.subplot(121)
        plt.ylim
        plt.plot(self.x, self.y, 'ro', label="training points")
        plt.plot(self.x, self.W[-1] * self.x + self.B[-1], label = "estimated␣
 ↪line")
        plt.xlabel('x')
        plt.ylabel('y')
        plt.ylim((-10, 15))
        plt.title('Data Space Iteration: ' + str(self.n))

        plt.subplot(122)
        plt.contour(self.w, self.b, self.Z)
        plt.scatter(self.W, self.B, c = 'r', marker = 'x')
        plt.title('Total Loss Surface Contour Iteration' + str(self.n))
        plt.xlabel('w')
        plt.ylabel('b')
        plt.show()
```

Make Some Data

Import PyTorch:

[3]:

```
# Import PyTorch library

import torch
```

Start with generating values from -3 to 3 that create a line with a slope of 1 and a bias of -1. This is the line that you need to estimate.

[4]:

```
# Create f(X) with a slope of 1 and a bias of -1

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

Now, add some noise to the data:

[5]:

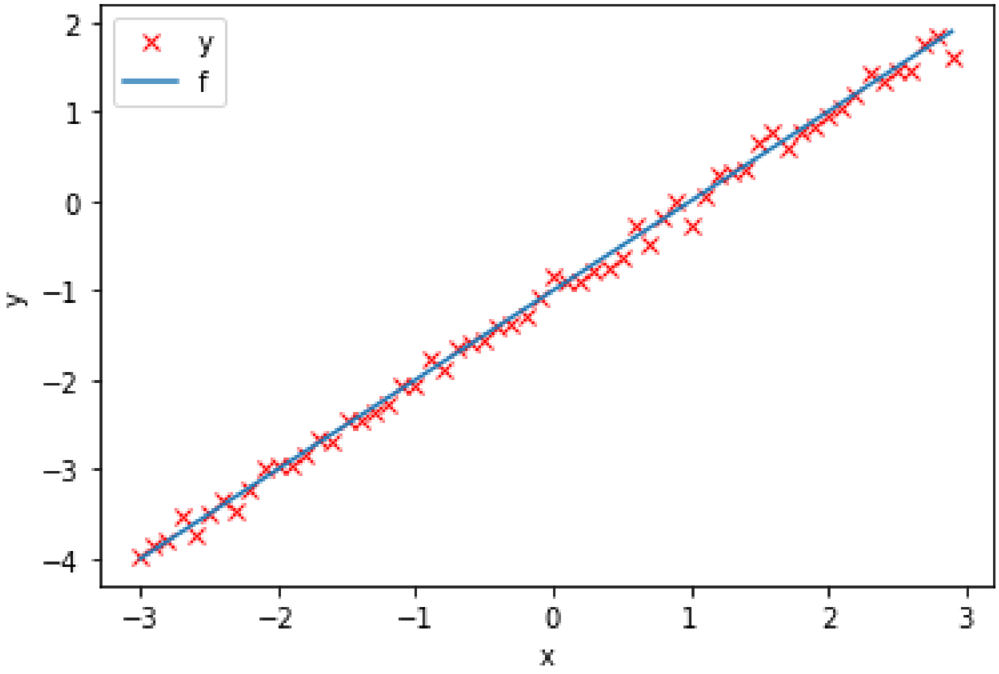
```
# Add noise

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

Plot the line and Y with noise:

[6]:

```
# Plot out the line and the points with noise

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()
```

```
[6]: <matplotlib.legend.Legend at 0x7fb18a12ddd0>
```



Create the Model and Cost Function (Total Loss)

Define the forward function:

[7]:

```
# Define the forward function

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

Define the cost or criterion function (MSE):

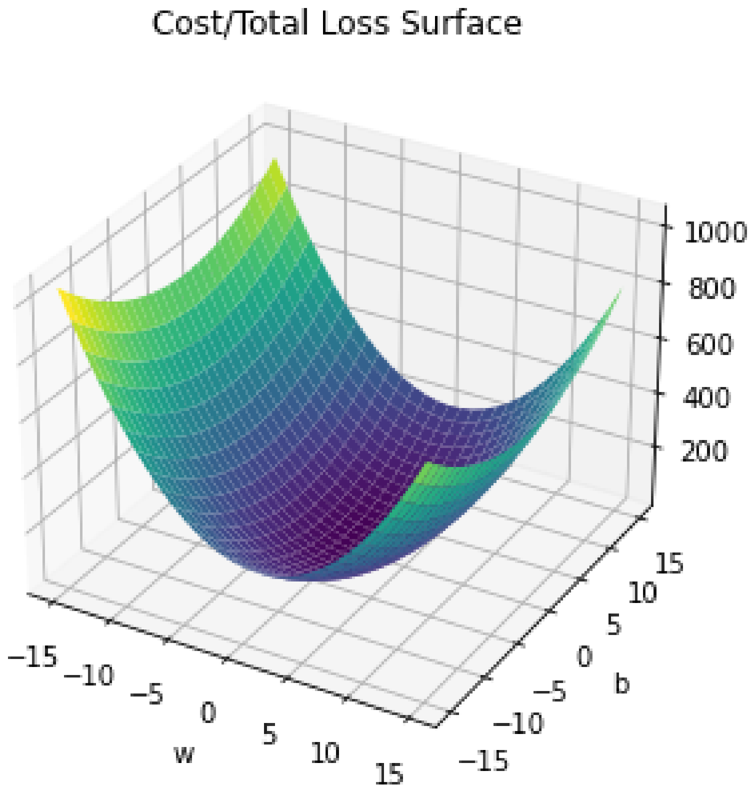
[8]:

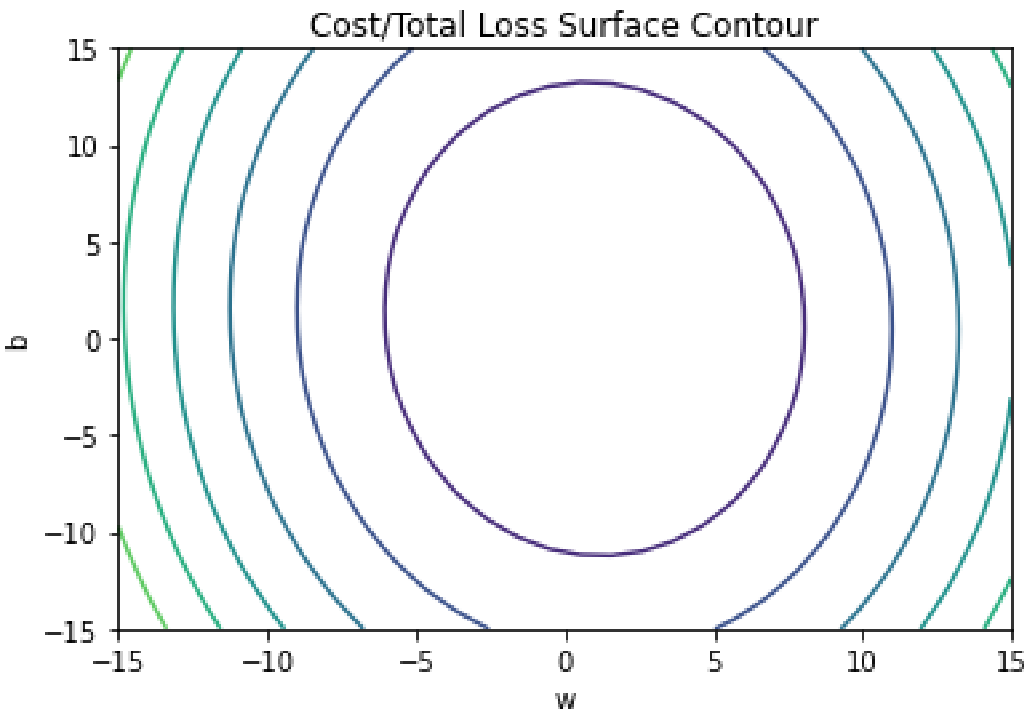
```
# Define the MSE Loss function

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

Create a plot__error__surfaces object to visualize the data space and the parameter space during training:

[9]:
```
# Create plot_error_surfaces for viewing the data

get_surface = plot_error_surfaces(15, 15, X, Y, 30)
```

```
<Figure size 432x288 with 0 Axes>
```





Train the Model

Create model parameters w, b by setting the argument requires_grad to True because we must learn it using the data.

[10]:

```
# Define the parameters w, b for y = wx + b

w = torch.tensor(-15.0, requires_grad = True)
b = torch.tensor(-10.0, requires_grad = True)
```

Set the learning rate to 0.1 and create an empty list LOSS for storing the loss for each iteration.

[11]:

```
# Define learning rate and create an empty list for containing the loss for each iteration.

lr = 0.1
LOSS = []
```

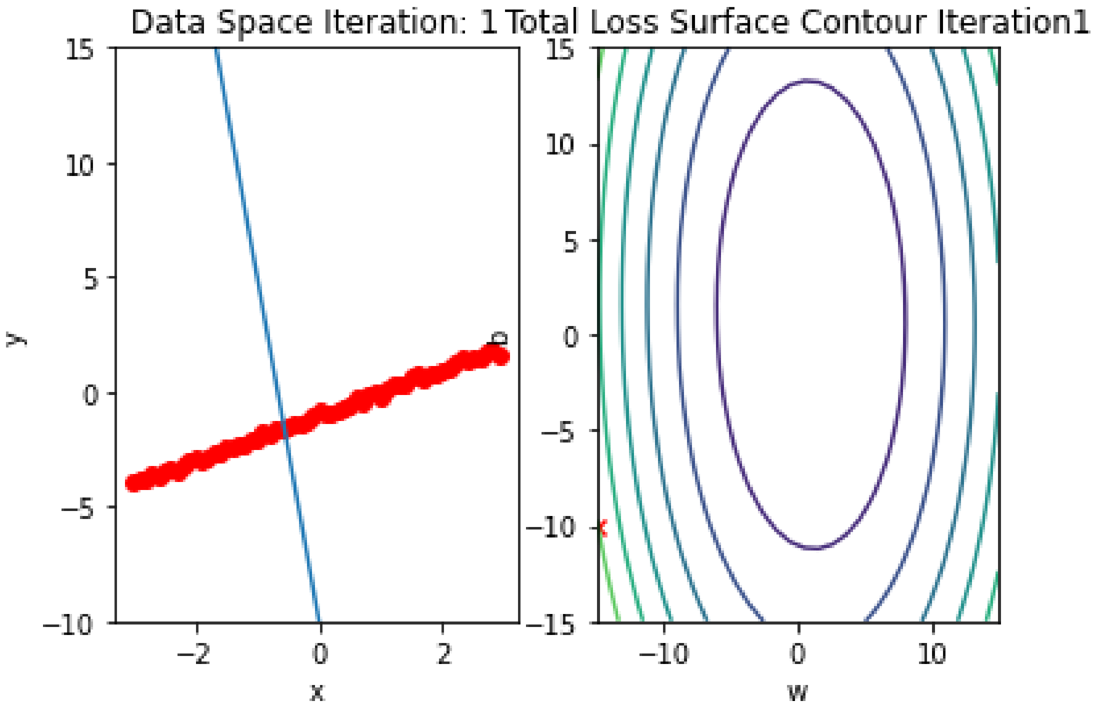
Define train_model function for train the model.

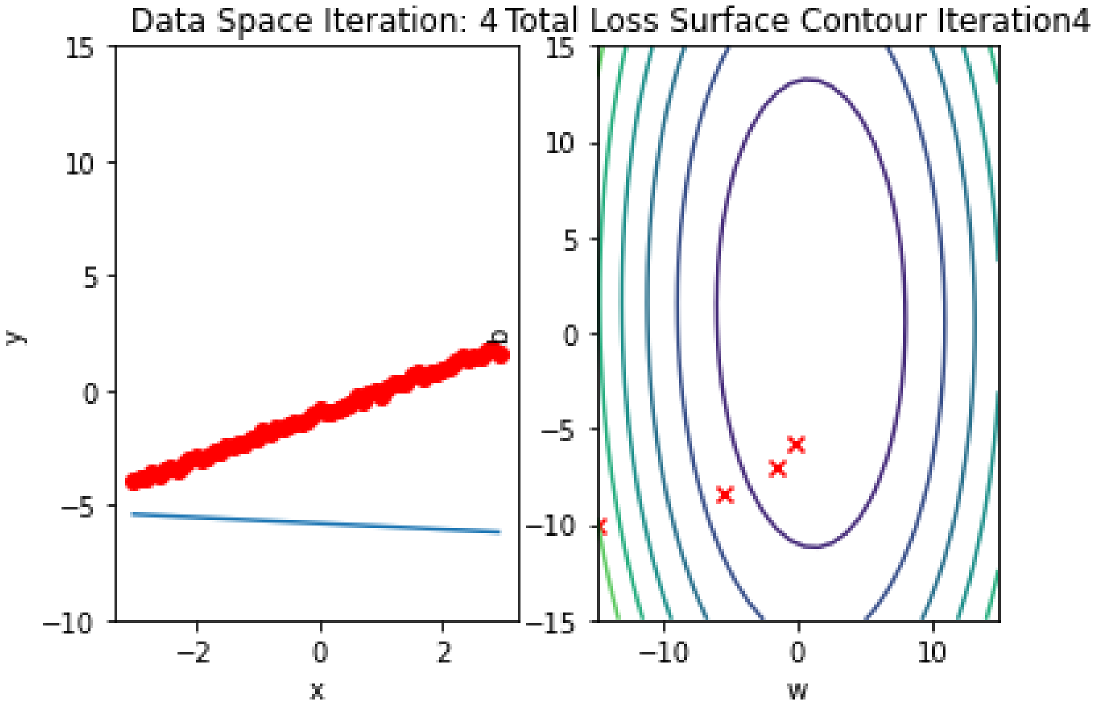
[12]:

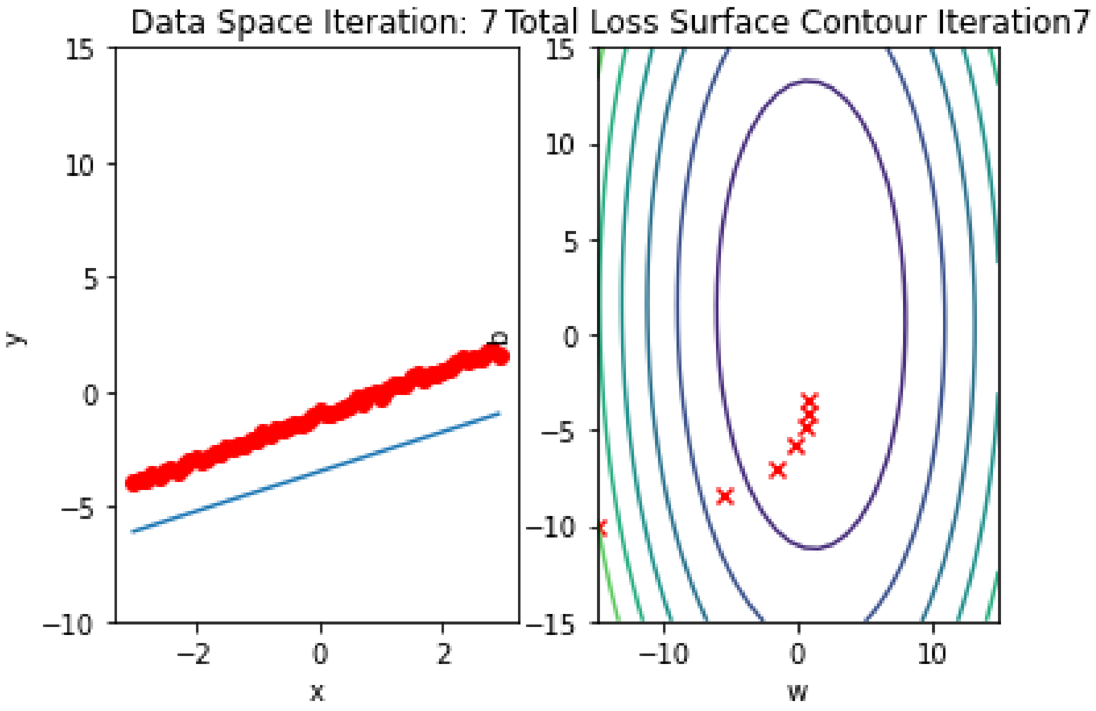
```
# The function for training the model

def train_model(iter):

    # Loop
    for epoch in range(iter):

        # make a prediction
        Yhat = forward(X)

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

        # Section for plotting
        get_surface.set_para_loss(w.data.tolist(), b.data.tolist(), loss.
↪tolist())
        if epoch % 3 == 0:
            get_surface.plot_ps()

        # store the loss in the list LOSS
        LOSS.append(loss)

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

        # update parameters slope and bias
        w.data = w.data - lr * w.grad.data
        b.data = b.data - lr * b.grad.data

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

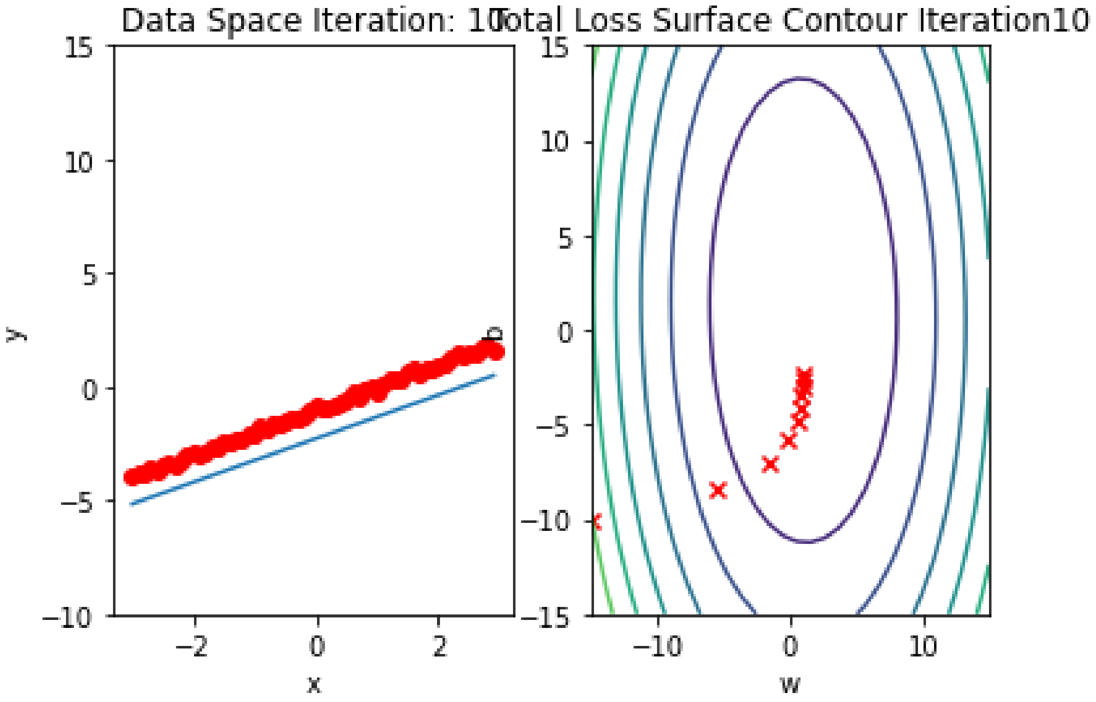
Run 15 iterations of gradient descent: bug data space is 1 iteration ahead of parameter space

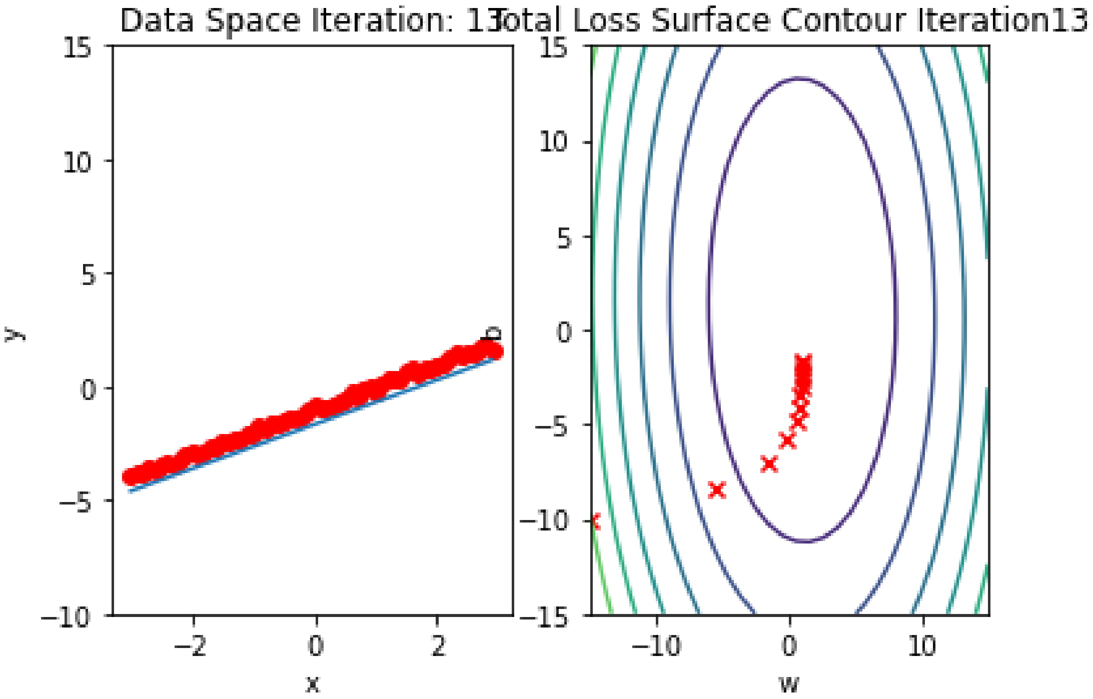
[13]:
```
# Train the model with 15 iterations

train_model(15)
```


Data Space Iteration: 1
Total Loss Surface Contour Iteration1
15
10
5
0
−5
−10
y
−2
0
2
x
b
15
10
5
0
−5
−10
−15
−10
0
10
w


Data Space Iteration: 4
Total Loss Surface Contour Iteration4
15
10
5
0
−5
−10
y
−2
0
2
x
b
15
10
5
0
−5
−10
−15
−10
0
10
w


Data Space Iteration: 7
Total Loss Surface Contour Iteration7
15
10
5
0
−5
−10
−2
0
2
x
y
15
10
5
0
−5
−10
−15
−10
0
10
w
b

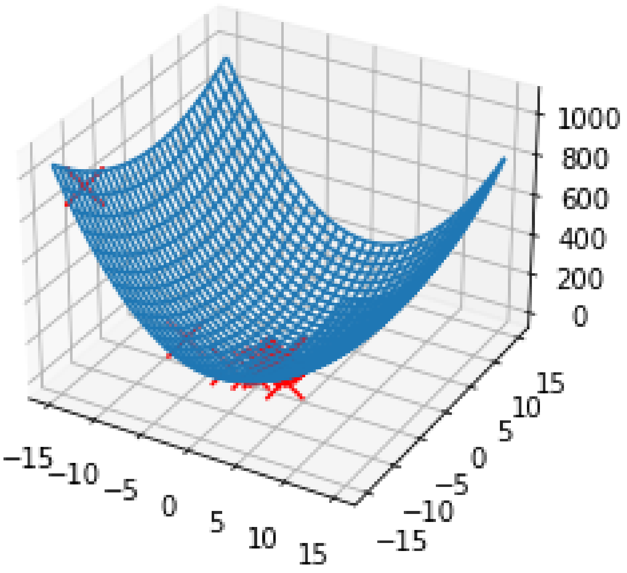

Data Space Iteration: 10
Total Loss Surface Contour Iteration10
15
10
5
0
−5
−10
−2
0
2
x
y
15
10
5
0
−5
−10
−15
−10
0
10
w
b

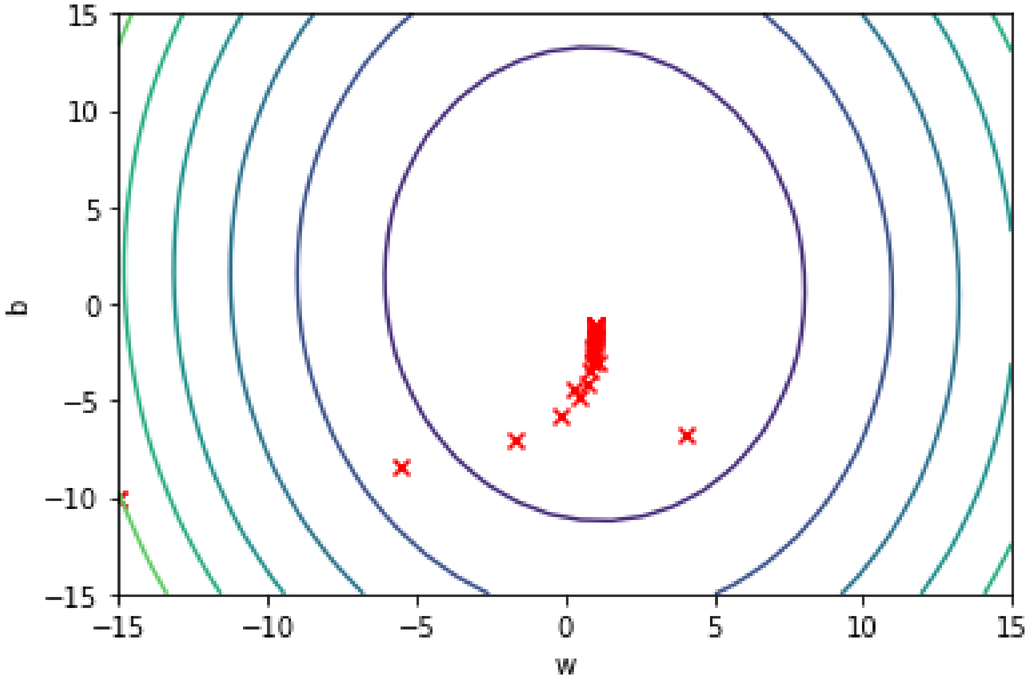


Plot total loss/cost surface with loss values for different parameters in red:

[18]:

```
# Plot out the Loss Result

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

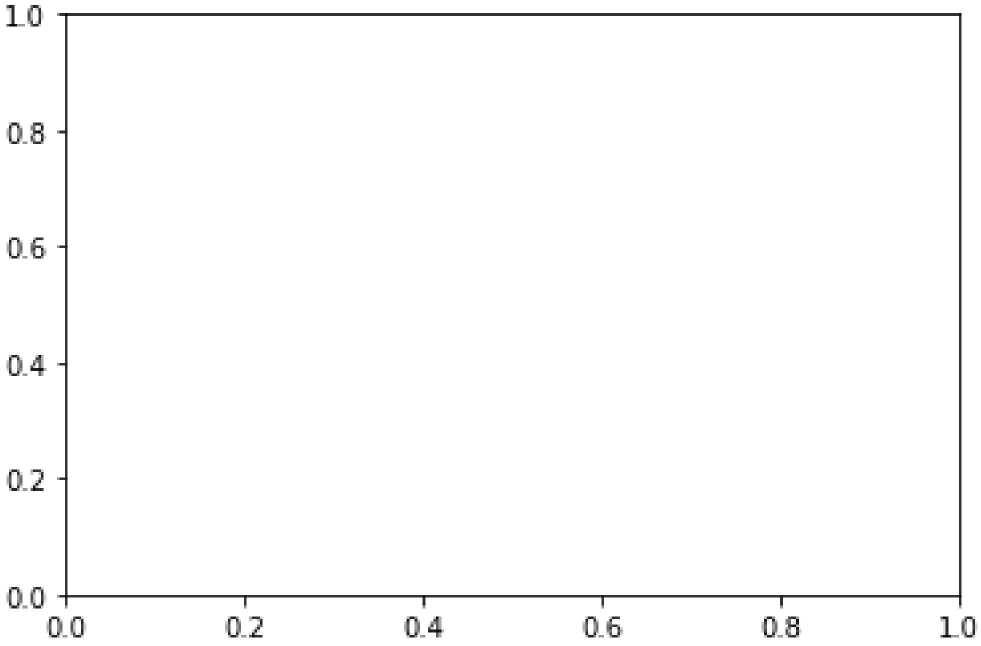




```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_575/3557730018.py in <module>
```

```
      2 
      3 get_surface.final_plot()
----> 4 plt.plot(LOSS)
      5 plt.tight_layout()
      6 plt.xlabel("Epoch/Iterations")

~/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)
    488             y = _check_1d(xy[1])
    489         else:
--> 490             x, y = index_of(xy[-1])
    491 
    492         if self.axes.xaxis is not None:

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/cbook/__init__.py in
 ↪index_of(y)
   1650         pass
   1651     try:
-> 1652         y = _check_1d(y)
   1653     except (np.VisibleDeprecationWarning, ValueError):
   1654         # NumPy 1.19 will warn on ragged input, and we can't actually␣
 ↪use it.
```

```
~/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.
```

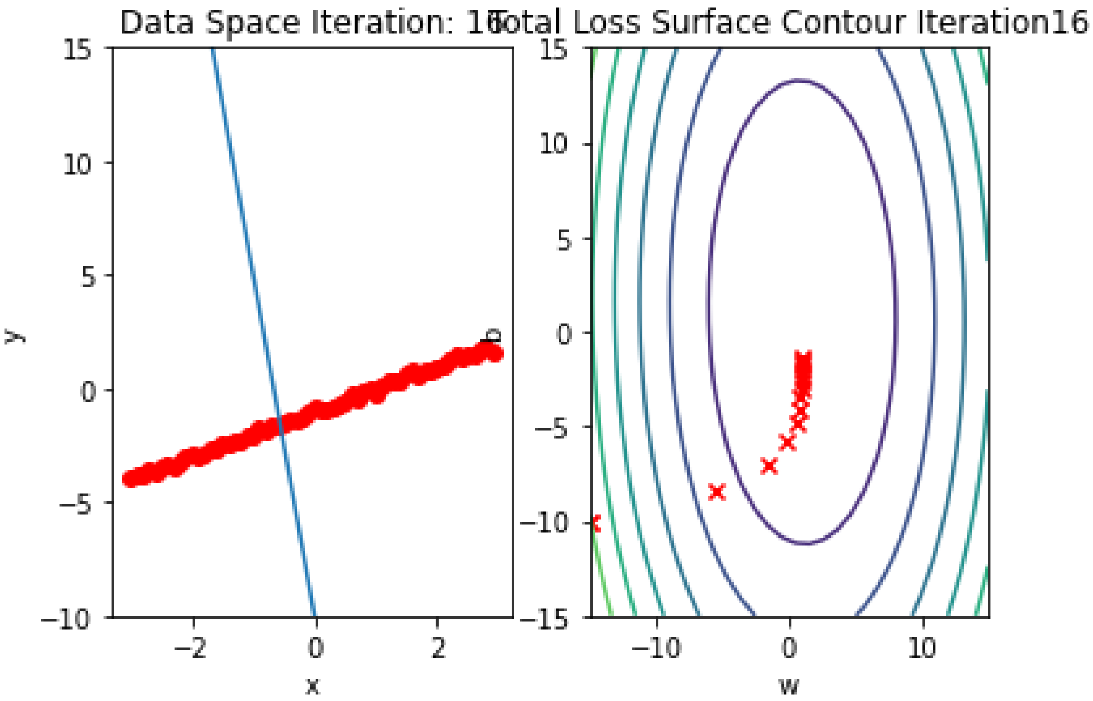


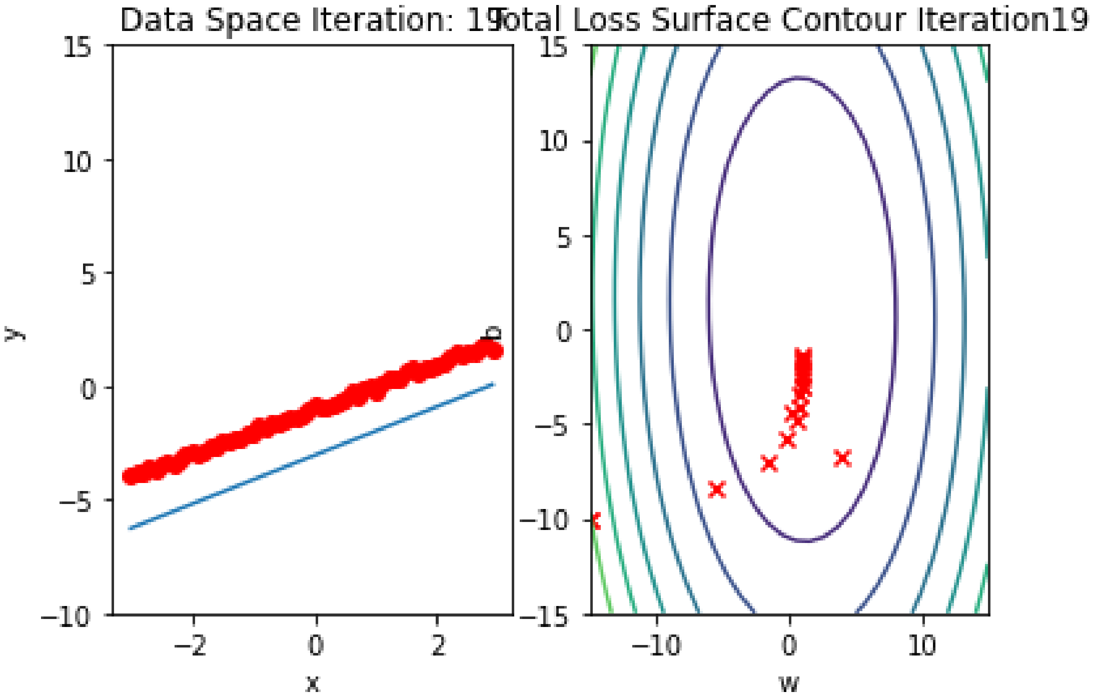
Practice

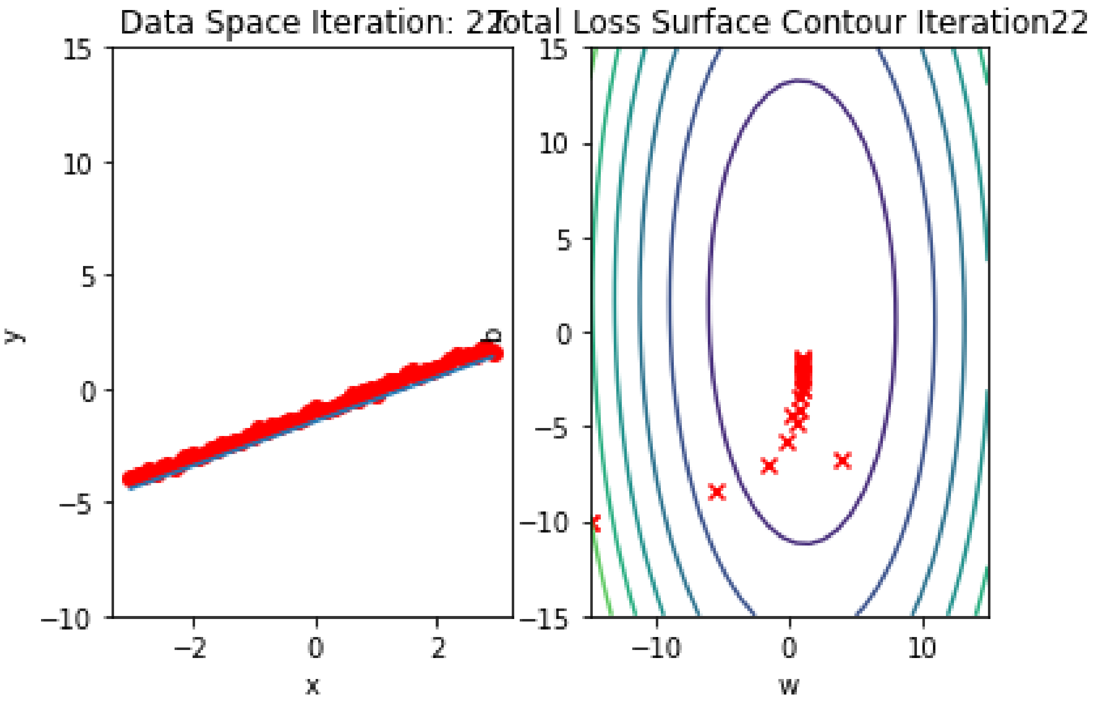
Experiment using s learning rates 0.2 and width the following parameters. Run 15 iterations.

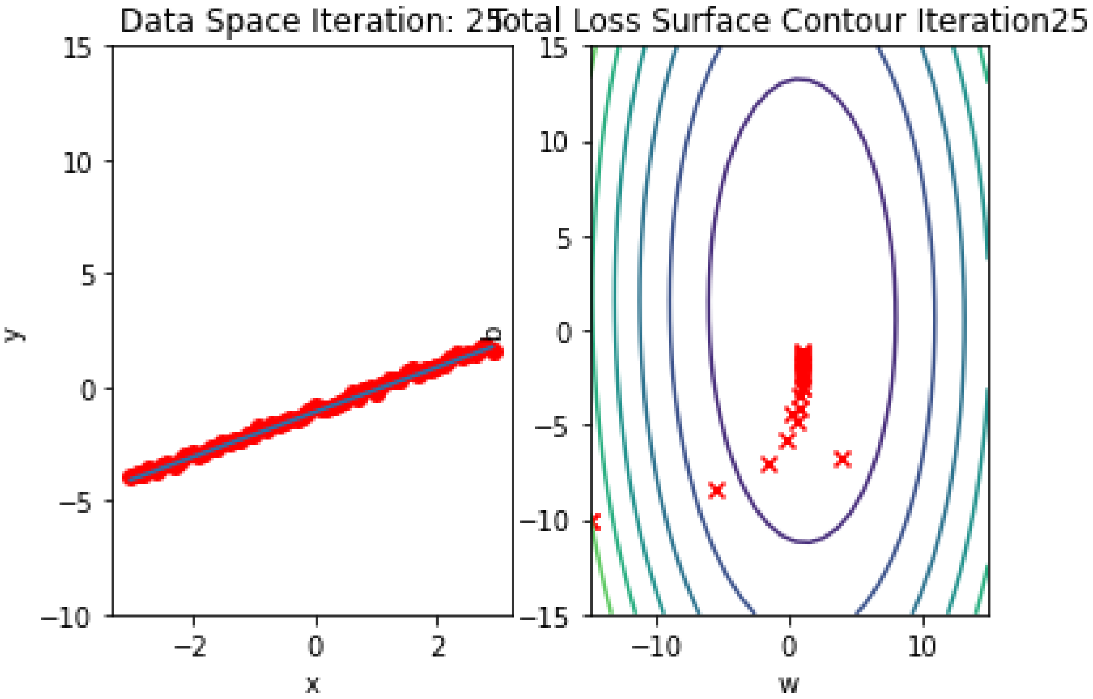
[15]:

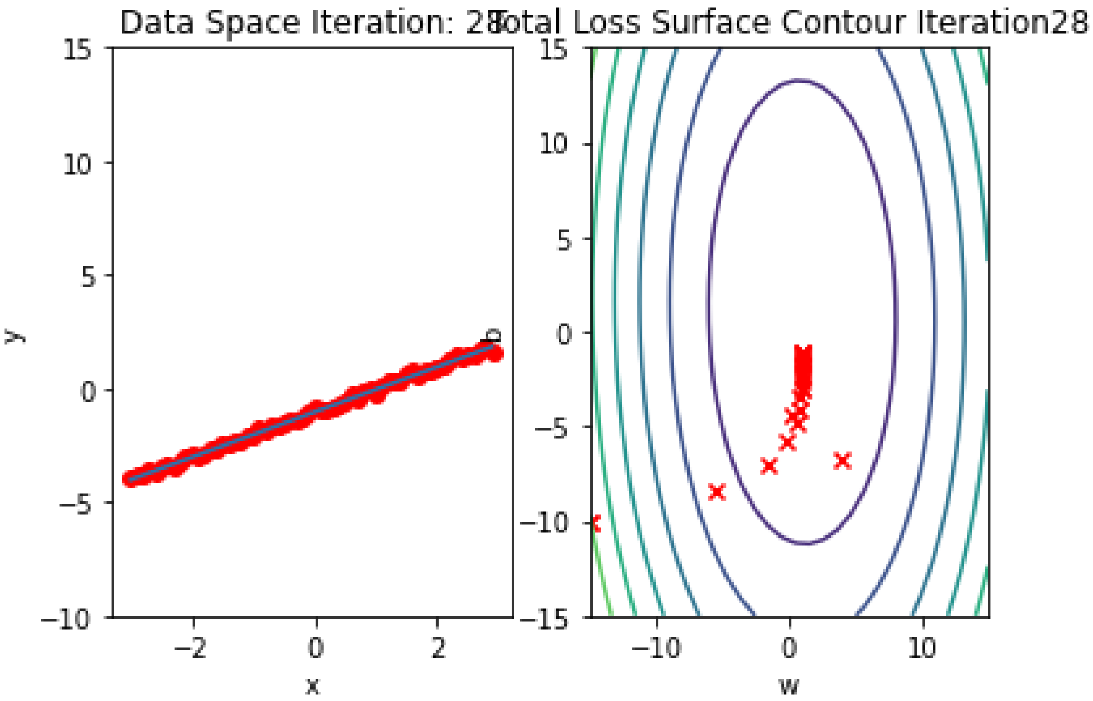
```
# Practice: train and plot the result with lr = 0.2 and the following parameters

w = torch.tensor(-15.0, requires_grad = True)
b = torch.tensor(-10.0, requires_grad = True)
lr = 0.2
LOSS2 = []
def my_train_model(iter):
    for epoch in range(iter):
        Yhat = forward(X)
        loss = criterion(Yhat, Y)
        get_surface.set_para_loss(w.data.tolist(), b.data.tolist(), loss.
 ↪tolist())
        if epoch % 3 == 0:
            get_surface.plot_ps()
        LOSS2.append(loss)
        loss.backward()
        w.data = w.data - lr * w.grad.data
        b.data = b.data - lr * b.grad.data
        w.grad.data.zero_()
        b.grad.data.zero_()
my_train_model(15)
```


Data Space Iteration: 16
Total Loss Surface Contour Iteration16
y
x
15
10
5
0
−5
−10
−2
0
2
b
w
15
10
5
0
−5
−10
−15
−10
0
10


Data Space Iteration: 19
Total Loss Surface Contour Iteration19
y
x
15
10
5
0
−5
−10
−2
0
2
b
w
15
10
5
0
−5
−10
−15
−10
0
10


Data Space Iteration: 22
Total Loss Surface Contour Iteration22
y
x
b
w
15
10
5
0
−5
−10
−15
−2
0
2
−10
0
10


Data Space Iteration: 25
Total Loss Surface Contour Iteration25
y
x
b
w
15
10
5
0
−5
−10
−15
−2
0
2
−10
0
10



Double-click here for the solution.

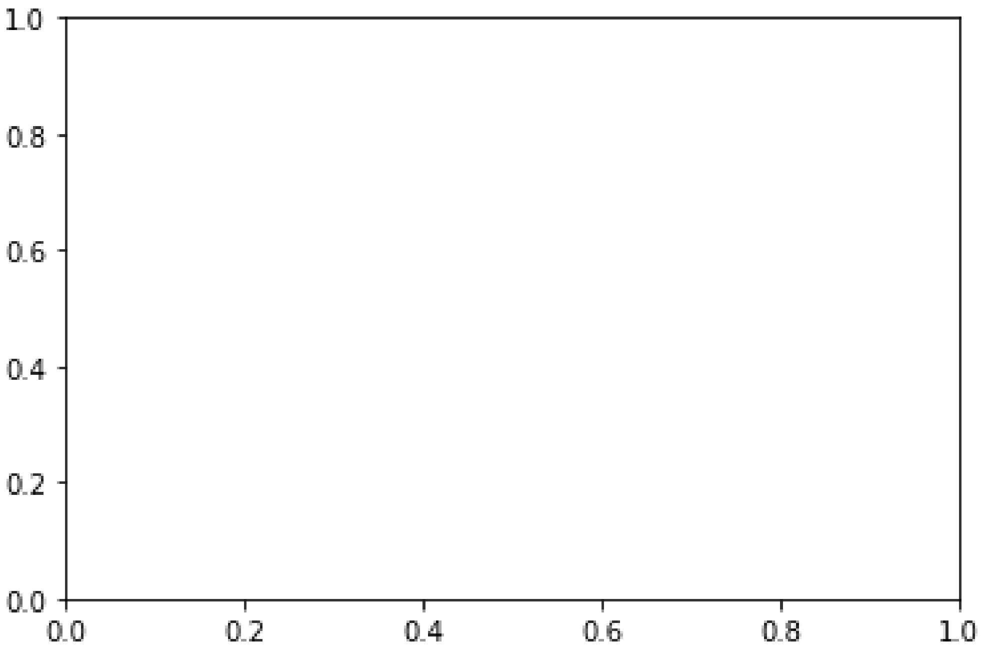
Plot the LOSS and LOSS2

[16]:
```
# Practice: Plot the LOSS and LOSS2 in order to compare the Total 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()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_575/902122170.py in <module>
      2
      3 # Type your code here
----> 4 plt.plot(LOSS, label = "LOSS")
      5 plt.plot(LOSS2, label = "LOSS2")
      6 plt.tight_layout()

~/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)
    488             y = _check_1d(xy[1])
    489         else:
--> 490             x, y = index_of(xy[-1])
    491 
    492         if self.axes.xaxis is not None:

~/conda/envs/python/lib/python3.7/site-packages/matplotlib/cbook/__init__.py in
 ↪index_of(y)
   1650         pass
   1651     try:
-> 1652         y = _check_1d(y)
   1653     except (np.VisibleDeprecationWarning, ValueError):
   1654         # NumPy 1.19 will warn on ragged input, and we can't actually␣
 ↪use it.

~/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.

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

##