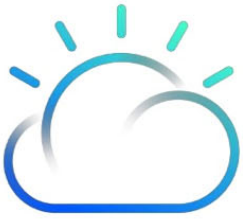


Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

Linear regression: Training and Validation Data

Table of Contents

In this lab, you will perform early stopping and save the model that minimizes the total loss on the validation data for every iteration.
*(**Note:** Early Stopping is a general term. We will focus on the variant where we use the validation data. You can also use a pre-determined number iterations*.)

Estimated Time Needed: **15 min**

Preparation

We'll need the following libraries, and set the random seed.

In [1]:

```
# Import the libraries and set random seed

from torch import nn
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch import nn,optim
from torch.utils.data import Dataset, DataLoader

torch.manual_seed(1)
```

Out[1]:

```
<torch._C.Generator at 0x15a19679290>
```

Make Some Data

First let's create some artificial data, in a dataset class. The class will include the option to produce training data or validation data. The training data includes outliers.

In [2]:

```
# Create Data Class

class Data(Dataset):

    # Constructor
    def __init__(self, train = True):
        if train == True:
            self.x = torch.arange(-3, 3, 0.1).view(-1, 1)
            self.f = -3 * self.x + 1
            self.y = self.f + 0.1 * torch.randn(self.x.size())
            self.len = self.x.shape[0]
            if train == True:
                self.y[50:] = 20
        else:
            self.x = torch.arange(-3, 3, 0.1).view(-1, 1)
            self.y = -3 * self.x + 1
            self.len = self.x.shape[0]

    # Getter
    def __getitem__(self, index):
        return self.x[index], self.y[index]

    # Get Length
    def __len__(self):
        return self.len
```

We create two objects, one that contains training data and a second that contains validation data, we will assume the training data has the outliers.

In [3]:

```
#Create train_data object and val_data object

train_data = Data()
val_data = Data(train = False)
```

We overlay the training points in red over the function that generated the data and the test data. Notice the outliers are at x=-3 and around x=2

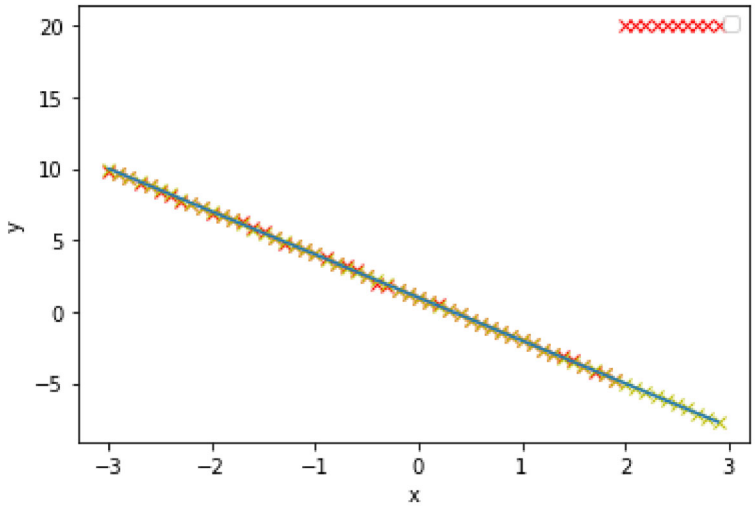
In [5]:

```
# Plot the training data points

plt.plot(train_data.x.numpy(), train_data.y.numpy(), 'xr')

plt.plot(val_data.x.numpy(), val_data.y.numpy(), 'xy')
plt.plot(train_data.x.numpy(), train_data.f.numpy())
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc = 'upper right')
plt.show()
label = 'training cost'
```

```
No handles with labels found to put in legend.
```



Create a Linear Regression Class, Object, Data Loader, Criterion Function

Create linear regression model class.

In [6]:

```
# Create linear regression model class

from torch import nn

class linear_regression(nn.Module):

    # Constructor
    def __init__(self, input_size, output_size):
        super(linear_regression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    # Predition
    def forward(self, x):
        yhat = self.linear(x)
        return yhat
```

Create the model object

In [7]:

```
# Create the model object

model = linear_regression(1, 1)
```

We create the optimizer, the criterion function and a Data Loader object.

In [8]:

```
# Create optimizer, cost function and data loader object

optimizer = optim.SGD(model.parameters(), lr = 0.1)
criterion = nn.MSELoss()
trainloader = DataLoader(dataset = train_data, batch_size = 1)
```

Early Stopping and Saving the Mode

Run several epochs of gradient descent and save the model that performs best on the validation data.

In [9]:

```
# Train the model

LOSS_TRAIN = []
LOSS_VAL = []
n=1;
min_loss = 1000

def train_model_early_stopping(epochs, min_loss):
    for epoch in range(epochs):
        for x, y in trainloader:
            yhat = model(x)
            loss = criterion(yhat, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_train = criterion(model(train_data.x), train_data.y).item()
            loss_val = criterion(model(val_data.x), val_data.y).item()
            LOSS_TRAIN.append(loss_train)
            LOSS_VAL.append(loss_val)
            if loss_val < min_loss:
                value = epoch
                min_loss = loss_val
                torch.save(model.state_dict(), 'best_model.pt')

train_model_early_stopping(20, min_loss)
```

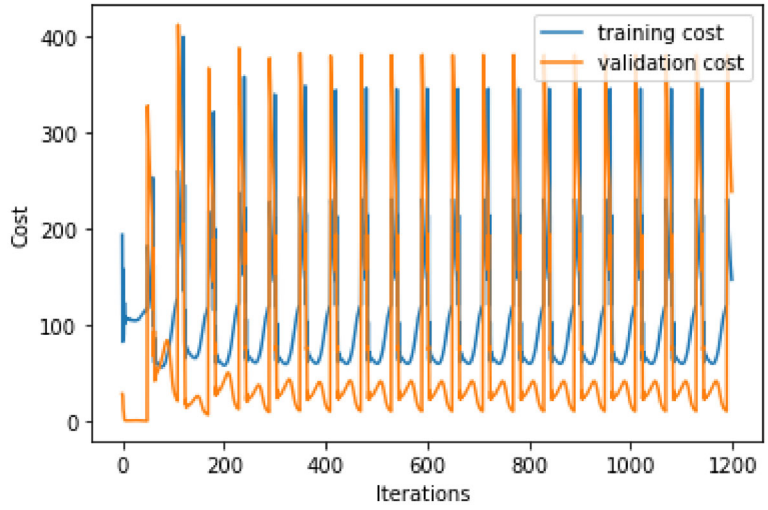
View Results

View the loss for every iteration on the training set and validation set.

In [10]:

```
# Plot the loss

plt.plot(LOSS_TRAIN, label = 'training cost')
plt.plot(LOSS_VAL, label = 'validation cost')
plt.xlabel("Iterations ")
plt.ylabel("Cost")
plt.legend(loc = 'upper right')
plt.show()
```



We will create a new linear regression object; we will use the parameters saved in the early stopping. The model must be the same input dimension and output dimension as the original model.

In [11]:

```
# Create a new linear regression model object

model_best = linear_regression(1,1)
```

Load the model parameters `torch.load()` , then assign them to the object `model_best` using the method `load_state_dict` .

In [12]:

```
# Assign the best model to model_best

model_best.load_state_dict(torch.load('best_model.pt'))
```

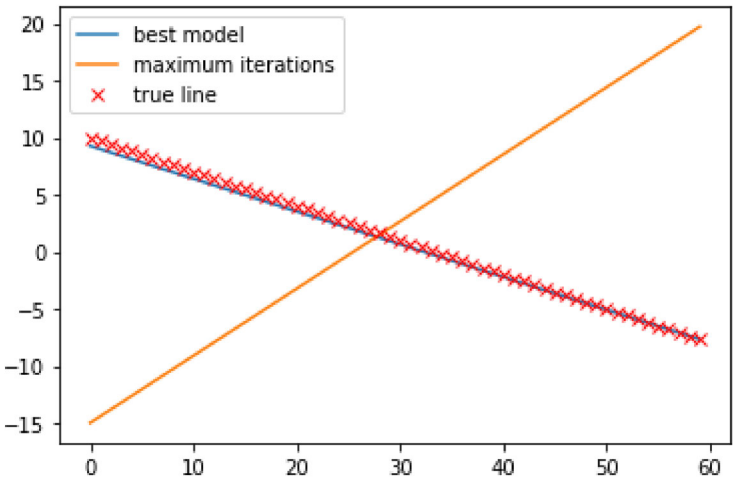
Out[12]:

```
<All keys matched successfully>
```

Let's compare the prediction from the model obtained using early stopping and the model derived from using the maximum number of iterations.

In [13]:

```
plt.plot(model_best(val_data.x).data.numpy(), label = 'best model')
plt.plot(model(val_data.x).data.numpy(), label = 'maximum iterations')
plt.plot(val_data.y.numpy(), 'rx', label = 'true line')
plt.legend()
plt.show()
```



We can see the model obtained via early stopping fits the data points much better. For more variations of early stopping see:

Prechelt, Lutz. *"Early stopping-but when?." Neural Networks: Tricks of the trade. Springer, Berlin, Heidelberg, 1998. 55-69.*

Inference

About the Authors:

Joseph Santarcangelo (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/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)
