4.4.training_multiple_output_linear_regression

March 28, 2022

Linear Regression Multiple Outputs

Objective

How to create a complicated models using pytorch build in functions.

Table of Contents

In this lab, you will create a model the Pytroch way. This will help you as models get more complicated.

Make Some Data

Create the Model and Cost Function the Pytorch way

Train the Model: Batch Gradient Descent

Practice Questions

Estimated Time Needed: 20 min

Import the following libraries:

[1]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch import nn,optim
from mpl_toolkits.mplot3d import Axes3D
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
```

Set the random seed:

[2]:
```
torch.manual_seed(1)
```

[2]:
```
<torch._C.Generator at 0x7f903805a330>
```

Make Some Data

Create a dataset class with two-dimensional features and two targets:

[3]:
```
from torch.utils.data import Dataset, DataLoader
class Data(Dataset):
    def __init__(self):
            self.x=torch.zeros(20,2)
            self.x[:,0]=torch.arange(-1,1,0.1)
            self.x[:,1]=torch.arange(-1,1,0.1)
            self.w=torch.tensor([ [1.0,-1.0],[1.0,3.0]])
            self.b=torch.tensor([[1.0,-1.0]])
            self.f=torch.mm(self.x,self.w)+self.b

            self.y=self.f+0.001*torch.randn((self.x.shape[0],1))
            self.len=self.x.shape[0]

    def __getitem__(self,index):

        return self.x[index],self.y[index]

    def __len__(self):
        return self.len
```

create a dataset object

[4]:
```
data_set=Data()
```

Create the Model, Optimizer, and Total Loss Function (cost)

Create a custom module:

[5]:
```
class linear_regression(nn.Module):
    def __init__(self,input_size,output_size):
        super(linear_regression,self).__init__()
        self.linear=nn.Linear(input_size,output_size)
    def forward(self,x):
        yhat=self.linear(x)
        return yhat
```

Create an optimizer object and set the learning rate to 0.1. **Don't forget to enter the model parameters in the constructor.**

[6]:
```
model=linear_regression(2,2)
```

Create an optimizer object and set the learning rate to 0.1. **Don't forget to enter the model parameters in the constructor.**

[7]:
```
optimizer = optim.SGD(model.parameters(), lr = 0.1)
```

Create the criterion function that calculates the total loss or cost:

[8]:
```
criterion = nn.MSELoss()
```

Create a data loader object and set the batch_size to 5:

[9]:
```
train_loader=DataLoader(dataset=data_set,batch_size=5)
```

Train the Model via Mini-Batch Gradient Descent

Run 100 epochs of Mini-Batch Gradient Descent and store the total loss or cost for every iteration. Remember that this is an approximation of the true total loss or cost.

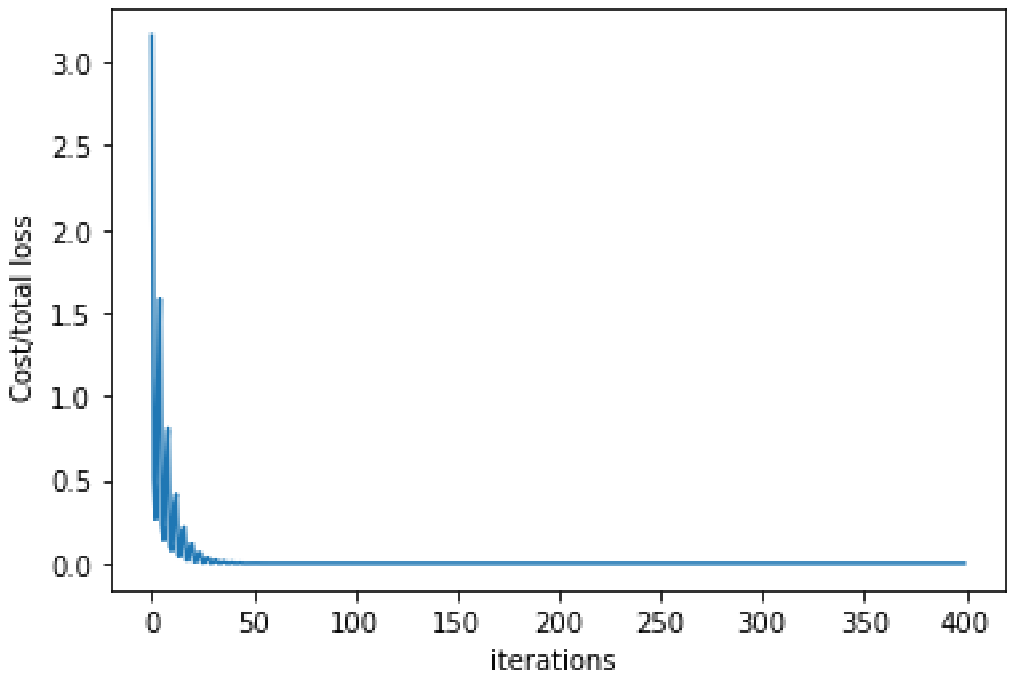
[10]:
```
LOSS=[]

epochs=100

for epoch in range(epochs):
    for x,y in train_loader:
        #make a prediction
        yhat=model(x)
        #calculate the loss
        loss=criterion(yhat,y)
        #store loss/cost
        LOSS.append(loss.item())
        #clear gradient
        optimizer.zero_grad()
        #Backward pass: compute gradient of the loss with respect to all the␣
 ↪learnable parameters
        loss.backward()
        #the step function on an Optimizer makes an update to its parameters
        optimizer.step()
```

Plot the cost:

[11]:
```
plt.plot(LOSS)
plt.xlabel("iterations ")
plt.ylabel("Cost/total loss ")
plt.show()
```



0.0.1 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.

Other contributors: Michelle Carey

0.1 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-23	2.0	Shubham	Migrated Lab to Markdown and added to course repo in GitLab

##