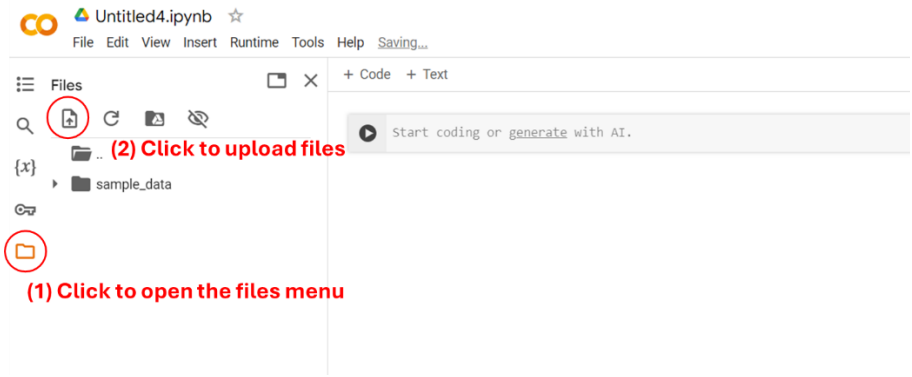
EE6310

Problem Set 14

In this problem set, we will work with the new regression data se:

Bandgap	**H%**	**Li%**	**Be%**	**…**	**U%**
y	x_1	x_2	x_3		x_{79}
7.657	0.333	0	0	…	0
0	0.789	0	0	…	0
0	0	0.028	0	…	0
…	…	…	…	…	

The full data can be downloaded from the weekly learning menu named "HW14.csv." To upload the csv file to Google Colab, Follow the following steps:



You will have to run all the proceeding code blocks to be able to run the code.

(a) Implement the batch normalizatoin by filling the code between "### START CODE HERE ###" and "### END CODE HERE ###"

```
import torch.nn as nn
class Model(nn.Module):
    def __init__(self,n_0,n_1,n_2,n_3,n_4):
        super().__init__()
        self.L1 = nn.Linear(n_0,n_1)
        # Add a batch norm layer
        ### START CODE HERE ###  (≈ 1 line of code)

        ### END CODE HERE ###
        self.L2 = nn.Linear(n_1,n_2)
        self.L3 = nn.Linear(n_2,n_3)
        self.L4 = nn.Linear(n_3,n_4)
        self.act1 = nn.ReLU()

    def forward(self, X):
        Z1 = self.L1(X)
        ### START CODE HERE ###  (≈ 1 line of code)

        ### END CODE HERE ###
        A1 = self.act1(Z1)
        Z2 = self.L2(A1)
        A2 = self.act1(Z2)
        Z3 = self.L3(A2)
        A3 = self.act1(Z3)
        Z4 = self.L4(A3)
        return Z4
```

(b) The code below draws the learning curve. Run the code in (a) first and run the code block below. It may take about 10 minutes to run. Does the model have high bias and variance? What would you need to do to improve your model?

```
from torch.utils.data import Dataset
import numpy as np
import pandas as pd
import torch
from torch.utils.data import DataLoader, random_split, Subset
import torch.optim as optim
class MaterialsDataset(Dataset):
    def __init__(self, path):
        df = pd.read_csv(path)
        data = df.to_numpy()
        data = torch.tensor(data,dtype=torch.float32)
        self.Y = data[:,:1]
        self.X = data[:,2:]

    def __len__(self):
        number_of_data = self.Y.shape[0]
        return number_of_data
    def __getitem__(self, idx):
        x = self.X[idx,:]
        y = self.Y[idx,:]
        return x,y
data = MaterialsDataset('HW14.csv')

# split data
data_train, data_val, data_test = random_split(data,[0.9,0.05,0.05])
dataloader_val = DataLoader(data_val, batch_size=64, shuffle=True)
dataloader_test = DataLoader(data_test, batch_size=64, shuffle=True)

num_data_samples =
[1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,15000,20000,2500
0,30000,len(data)]
val_losses = []
train_losses = []
for num_data in num_data_samples:
    print(num_data)
    indices = list(range(len(data_train)))
    np.random.shuffle(indices)
    indices = indices[:num_data]
    data_train_subset = Subset(data_train,indices)
    dataloader_train = DataLoader(data_train_subset, batch_size=64,
shuffle=True)
```

```
    max_epoch = 100
    NN = Model(82,32,32,32,1)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(NN.parameters(), lr=0.01)
    min_val_loss = torch.Tensor([float('Inf')])
    train_loss_at_min_val_loss = 0
    for i in range(max_epoch):
        train_loss = 0
        for X, Y in dataloader_train:
            Z= NN(X)
            optimizer.zero_grad()
            loss = criterion(Z,Y)
            loss.backward()
            optimizer.step()
            train_loss += loss*Y.shape[0]
        train_loss = train_loss/len(data_train)
        loss_val = 0
        for X, Y in dataloader_val:
            Z = NN(X)
            loss_val += criterion(Z,Y)*Y.shape[0]
        loss_val = loss_val/len(data_val)
        print(f'epoch {i+1:4d} : train_loss {train_loss:.3f}
val_loss {loss_val:.3f}',end=' ')
        if loss_val < min_val_loss:
            torch.save(NN.state_dict(),'best.pth.tar')
            min_val_loss = loss_val
            train_loss_at_min_val_loss = train_loss
            print('<-new best',end='')
        print('')
    val_losses.append(min_val_loss.detach().numpy())
    train_losses.append(train_loss_at_min_val_loss.detach().numpy())
    test_Ys = []
    test_y_hat = []
    NN.load_state_dict(torch.load('best.pth.tar'))
    for X, Y in dataloader_test:
        Z = NN(X)
        test_Ys.append(Y)
        test_y_hat.append(Z)
    test_Ys = torch.cat(test_Ys)
    test_y_hat = torch.cat(test_y_hat).detach()
    print(torch.mean(torch.abs(test_y_hat-test_Ys)))
import matplotlib.pyplot as plt

plt.scatter(num_data_samples,val_losses)
plt.scatter(num_data_samples,train_losses)
```