EE6310

Problem Set 14

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bandgap  y | H%  *x1* | Li%  *x2* | Be%  *x3* | … | U%  *x79* |
| 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:

A screenshot of a computer

Description automatically generated

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

1. 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 |

1. 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[:,1:]    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 = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,15000,20000,25000,30000,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(83,32,32,32,1)  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) |