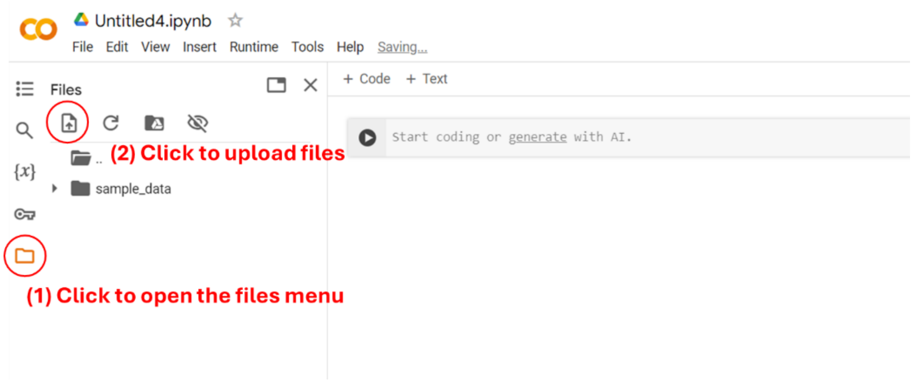
EE6310

Problem Set 13

In this problem set, we will implement various tools we learned in the lecture. We will work with the new regression data se:

Bandgap y	**H%** x_1	**Li%** x_2	**Be%** x_3	**…**	**U%** 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 "HW13.csv." To upload the csv file to Google Colab, Follow the following steps:


Untitled4.ipynb
File Edit View Insert Runtime Tools Help Saving...
Files
+ Code + Text
Start coding or generate with AI.
(2) Click to upload files
sample_data
(1) Click to open the files menu

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

(a) Let's implement the skip connection by filling the code between "### START CODE HERE ###" and "### END CODE HERE ###" In the lecture we used the "+=" to implement it, but turns out it actually does not work. Use a = a + b format.

```
from torch.utils.data import Dataset
import numpy as np
import pandas as pd
import torch
from torch.utils.data import DataLoader, random_split
#%%
# Add the skip connection from the A1 to A3
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)
        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)
        A1 = self.act1(Z1)
        Z2 = self.L2(A1)
        A2 = self.act1(Z2)
        Z3 = self.L3(A2)
        A3 = self.act1(Z3)
        ### START CODE HERE ###  (≈ 1 line of code)
        A3 = A3 + A1
        ### END CODE HERE ###
        Z4 = self.L4(A3)
        return Z4
```

(b) Implement the Dataset class by filling the code between "### START CODE HERE ###" and "### END CODE HERE ###" Read the comment before for the hint!

```
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):
        # The number of training examples needs to be assigned to
        # number_of_data
        ### START CODE HERE ###  (≈ 1 line of code)
        number_of_data =
        ### END CODE HERE ###
        return number_of_data
    def __getitem__(self, idx):
        # Write code below to get the x and y vectors from the
        # self.X and self.Y
        ### START CODE HERE ###  (≈ 2 line of code)
        x =
        y =
        ### END CODE HERE ###
        return x,y
data = MaterialsDataset('HW13.csv')
```

(c) Here let's split the data in to train, validation and test set. We will talk more about using all three in the next lecture. Just splitting the data is enough for now. Fill the blank appropriately between "### START CODE HERE ###" and "### END CODE HERE ###"

```
# We will split the data in to 90% train, 5% validation, and 5%
test.
# It turns out that you can also specify the ratio of the train,
validation,
# instead of the number of data points.
# Define the following parameters below:
# Rtrain: (float) ratio of training set data
# Rval: (float) ratio of validation set data
# Rtest: (float) ratio of test set data
### START CODE HERE ###  (≈ 3 line of code)
Rtrain = 0.9
Rval = 0.05
Rtest = 0.05
### END CODE HERE ###
data_train, data_val, data_test =
random_split(data,[Rtrain,Rval,Rtest])
```

(d) Make DataLoader objects. We will use the batch size of 128. Fill the blank appropriately between "### START CODE HERE ###" and "### END CODE HERE ###"

```
# in the code below, make dataloader for training, validation, and
test set
# with batch size of 128
### START CODE HERE ###  (≈ 3 line of code)
dataloader_train = DataLoader(data_train, batch_size=128,
shuffle=True)
dataloader_val = DataLoader(data_val, batch_size=128, shuffle=True)
dataloader_test = DataLoader(data_test, batch_size=128,
shuffle=True)
### END CODE HERE ###
```

(e) Now we will implement the Adam optimizer. We have the old code from logistic regression which we can modify to implement Adam optimizer. Change appropriately the code between "### MODIFY CODE IN THIS BLOCK ###"

```
# Modify the code below to implement the Adam optimizer
### MODIFY CODE IN THIS BLOCK ###  (≈ 1 line of code to be modified)
import torch.optim as optim
NN = Model(82,32,32,32,1)
criterion = nn.MSELoss()
optimizer = optim.SGD(NN.parameters(), lr=0.01)
### MODIFY CODE IN THIS BLOCK ###
for epoch in range(1000):
    losses = 0.0
    for X, Y in dataloader_train:
        Z = NN(X)
        optimizer.zero_grad()
        loss = criterion(Z,Y)
        loss.backward()
        optimizer.step()
        losses += Z.shape[0] * loss.item()
    print(f'Epoch {epoch+1:4d}: loss = {losses/len(data_train):.4f}')
```