#!/usr/bin/python

from numpy import vstack

import math

from pandas import read\_csv

#from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score,mean\_squared\_error

from torch.utils.data import Dataset

from torch.utils.data import DataLoader

from torch.utils.data import random\_split

from torch import Tensor

import torch

from torch.nn import Sigmoid,Softmax,ReLU,Linear,Tanh

from torch.nn import Module

from torch.optim import SGD,Adam

from torch.nn import BCELoss,NLLLoss,CrossEntropyLoss,MSELoss

from torch.nn.init import kaiming\_uniform\_,xavier\_uniform\_

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import time

from setup import readInput

n\_inputs, frac\_val, frac\_test, H, l\_rate, batch, epochs, test\_datapath,path = readInput()

#n\_inputs = 18

#H = 20 # Number of nodes in a hidden layer (please check if it is 1 hidden layer)

f = open("output\_Amplitude\_train.out","w")

# dataset loading

class CSVDataset(Dataset):

def \_\_init\_\_(self, path,lst,det,flag):

df = read\_csv(path,usecols=lst, header=None)

df\_det = read\_csv(path,usecols=[int(det)], header=None)

if flag:

self.X = df.values[:, :-1]

self.y = df.values[:, -1]

self.det = df\_det.values[:,-1]

self.X = self.X.astype('float32')

self.y = self.y.astype('float32')

self.det = self.det.astype('float32')

self.y = self.y.reshape((len(self.y), 1))

self.det = self.det.reshape((len(self.det), 1))

else:

self.X = df.values

self.det = df\_det.values

self.X = self.X.astype('float32')

self.det = self.det.astype('float32')

self.det = self.det.reshape((len(self.det), 1))

def \_\_len\_\_(self):

return len(self.X)

def \_\_getitem\_\_(self, idx):

if flag:

return [self.X[idx], self.y[idx], self.det[idx]]

else:

return [self.det[idx],self.X[idx]]

def get\_splits(self, n\_test): # spliting of dataset

test\_size = int(round(n\_test \* len(self.X)))

train\_size = int(len(self.X) - test\_size)

return random\_split(self, [train\_size, test\_size])

#------------------------------PREPARE THE DATA--------------------------------------#

def prepare\_data(path,lst,det,frac,flag):

dataset = CSVDataset(path,lst,det,flag)

train, test = dataset.get\_splits(frac)

train\_dl = DataLoader(train, batch\_size=batch, shuffle=False)

test\_dl = DataLoader(test, batch\_size=batch, shuffle=False)

return train\_dl

def prepare\_testdata(path,lst,det,frac,flag):

dataset = CSVDataset(path,lst,det,flag)

train, test = dataset.get\_splits(frac)

test\_dl = DataLoader(test, batch\_size=batch, shuffle=False)

return test\_dl

#====================================================================================#

# prepare the data

#path = 'input\_train\_18sites.csv'

train\_list= [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,21]

det=19

#frac\_val=0.0

flag=True

train\_dl = prepare\_data(path,train\_list,det,frac\_val,flag)

#---------------------------------- MODEL STRUCTURE ----------------------------------#

class Network(Module):

def \_\_init\_\_(self,n\_inputs):

super(Network,self).\_\_init\_\_()

#input descriptor

self.hidden1 = Linear(n\_inputs, H) # input to 1st hidden layer

kaiming\_uniform\_(self.hidden1.weight, nonlinearity='relu')

self.relu = ReLU()

self.output = Linear(H, 1) # 2nd hidden layer to output

xavier\_uniform\_(self.output.weight)

self.relu = ReLU()

def forward(self, X):

X = self.hidden1(X)

X = self.relu(X)

X = self.output(X)

X = self.relu(X)

return X

#====================================================================================#

model = Network(n\_inputs)

#----------------------------------TRAINING------------------------------------------#

def train\_model(train\_dl, model):

criterion = MSELoss() #loss function

optimizer = Adam(model.parameters(), lr=l\_rate, betas=(0.9, 0.999)) #optimizer should be used

for epoch in range(epochs):

running\_loss = 0.0

for i, (inputs, targets, dets) in enumerate(train\_dl):

optimizer.zero\_grad()

yhat = model(inputs)

loss = criterion(yhat, targets)

loss.backward()

optimizer.step()

running\_loss +=loss.item()

epoch\_loss = running\_loss/len(train\_dl)

#print(str(epoch)+" "+str(epoch\_loss))

plt.scatter(epoch, epoch\_loss,c='k')

plt.pause(1e-17)

time.sleep(0.00001)

#print(epcoh,'\t', epoch\_loss)

print('\n\nYep!! The training is done !!')

#==================================================================================#

train\_model(train\_dl, model)

torch.save(model.state\_dict(), "model.pth")

#test\_datapath = "18fciDet\_cutoff.csv"

test\_list= [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]

det=0

#frac\_test=1.0

flag=False

test\_dl = prepare\_testdata(test\_datapath,test\_list,det,frac\_test,flag)

#-----------------------------------EVALUATION-------------------------------------#

def evaluate\_model(test\_dl, model): #send the test dataset through the network

for i, (dets,inputs) in enumerate(test\_dl):

yhat = model(inputs)

yhat = yhat.detach().numpy()

dets = dets.numpy()

for j in range(len(yhat)):

f.write(str(int(dets[j][0]))+" "+str(10\*\*(-1.0\*yhat[j][0]))+"\n")

#print (int(dets[j][0]),yhat[j][0])

return True

#----------------------------------------------------------------------------------#

#==================================================================================#

evaluate\_model(test\_dl, model)

f.close()