#!/usr/bin/python

from numpy import vstack, loadtxt, argsort

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

from bitstring import Bits, BitStream, BitArray, ConstBitStream

from setup import readInput

model, nSite, subSpace, nStates, s2Target, maxItr, startSpinTargetItr, energyTola, spinTola, beta, jVal, det, Ms, posibleDet, bondOrder, outputfile, restart, saveBasis = readInput()

'''

nSite = 18

ciCutoff = 0.001

subSpace = 500

s2Target = 0

outputfile = "s0-6666-sub2000.in.out"

'''

H = nSite

nCycle = 500

testSize = 0.10

fTrain = outputfile + ".accVsPreTrain.dat"

fTest = outputfile + ".accVsPreTest.dat"

errorFile = outputfile +".error.dat"

################ For Error Calculations ###########

def error(data):

actual = []

prediction = []

for i in range(len(data)):

actual.append(data[i][0])

prediction.append(data[i][1])

acc = mean\_squared\_error(prediction, actual)

return acc

##############################################################

class CSVDataset(Dataset):

def \_\_init\_\_(self, path):

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

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

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

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

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

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

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

return len(self.X)

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

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

def get\_splits(self, n\_test = testSize): # 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])

class CSVDatasetPredict(Dataset):

def \_\_init\_\_(self, path):

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

self.X = df.values

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

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

return len(self.X)

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

return self.X[idx]

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#################################################

class Network(Module):

def \_\_init\_\_(self,nSite):

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

#input descriptor

self.hidden1 = Linear(nSite, H)

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)

# Define sigmoid activation and softmax output

self.relu = ReLU()

def forward(self, X):

# Pass the input tensor through each of our operations

X = self.hidden1(X)

X = self.relu(X)

X = self.output(X)

X = self.relu(X)

return X

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

######################################### validation ############################

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

predictions, actuals = list(), list()

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

yhat = model(inputs)

yhat = yhat.detach().numpy()

actual = targets.numpy()

actual = actual.reshape((len(actual), 1))

predictions.append(yhat)

actuals.append(actual)

predictions, actuals = vstack(predictions), vstack(actuals)

# calculate accuracy

acc = mean\_squared\_error(actuals, predictions)

return acc

#######################TRAINING ################################

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

criterion = MSELoss()

optimizer = Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))

for epoch in range(nCycle):

running\_loss = 0.0

for i, (inputs, targets) 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)

validation\_error = validation(test\_dl, model)

#print("epoch", epoch, "epoch\_loss", epoch\_loss, "validation\_error", validation\_error)

#print(epoch, epoch\_loss)#, validation\_error)

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###############

#################################################

def evaluate\_model(test\_dl, model, fl):

predictions, actuals = list(), list()

f = open(fl,"w")

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

yhat = model(inputs)

yhat = yhat.detach().numpy()

actual = targets.numpy()

actual = actual.reshape((len(actual), 1))

for j in range(len(actual)):

newline = ("%f %f\n")% (10.0 \*\* (actual[j][0] \*- 1), 10.0 \*\* (yhat[j][0] \*- 1))

f.write(newline)

predictions.append(yhat)

actuals.append(actual)

predictions, actuals = vstack(predictions), vstack(actuals)

# calculate accuracy

acc = mean\_squared\_error(actuals, predictions)

return acc

def predict\_model(test\_dl, model):

detList = []

predictValue = []

for inputs in test\_dl:

yhat = model(inputs)

yhat = yhat.detach().numpy()

for j in range(len(yhat)):

predictValue.append(yhat[j][0])

sort\_index = argsort(predictValue)

for j in range (len(sort\_index)):

inputList = inputs[sort\_index[j]].tolist()

inputStr = BitArray('0b'+''.join([str(int((elem + 1)/2)) for elem in inputList]))

if inputStr not in detList:

detList.append(inputStr)

if (Ms[0] == 0):

detList.append(~inputStr)

return detList

def enrich\_model(test\_dl, model):

allDet = []

ci = []

for inputs in test\_dl:

yhat = model(inputs)

yhat = yhat.detach().numpy()

for j in range(len(yhat)):

inputList = inputs[j].tolist()

inputStr = BitArray('0b'+''.join([str(int((elem + 1)/2)) for elem in inputList]))

if inputStr not in allDet:

allDet.append(inputStr)

ci.append(10.0 \*\* (yhat[j][0] \*- 1))

return allDet, ci

#####################################

# prepare the dataset

def prepare\_data(path):

dataset = CSVDataset(path)

train, test = dataset.get\_splits()

train\_dl = DataLoader(train, batch\_size=500, shuffle=True)

test\_dl = DataLoader(test, batch\_size=500, shuffle=True)

return train\_dl, test\_dl

def prepare\_predict\_data(path, size\_x):

dataset = CSVDatasetPredict(path)

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

return train\_dl

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

##########################################

def ann\_train(dataFile, predictDataFile):

# prepare the data

path = dataFile

train\_dl, test\_dl = prepare\_data(path)

# define the network

model = Network(nSite)

# train the models

train\_model (train\_dl, test\_dl, model)

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

acc1 = evaluate\_model(train\_dl, model, fTrain)

acc2 = evaluate\_model(test\_dl, model, fTest)

data1 = loadtxt(fTrain, usecols = [0, 1], dtype = float)

data2 = loadtxt(fTest, usecols = [0, 1], dtype = float)

with open (errorFile, "a") as fout:

newline = ("Train Error- %lf\t Test Error - %lf\n")%(error(data1), error(data2))

fout.write(newline)

#print("Train Error", error(data1))

#print("Test Error", error(data2))

path = predictDataFile

with open(predictDataFile, 'r') as fp: # to get line no of predictDataFile

size\_x = len(fp.readlines())

predict\_dl = prepare\_predict\_data(path, size\_x)

detList = predict\_model(predict\_dl, model)

#print (impDet)

return detList

def ann\_enrich(enrichDataFile):

path = enrichDataFile

with open(path, 'r') as fp: # to get line no of enrichDataFile

size\_x = len(fp.readlines())

enrich\_dl = prepare\_predict\_data(path, size\_x)

model = Network(nSite)

model.load\_state\_dict(torch.load(outputfile+".model.pth"))

allDet, ci = enrich\_model(enrich\_dl, model)

return allDet, ci

#ann\_train("TrainData\_subSpace500\_spin0.0.csv", "PredictData\_subSpace500\_spin0.0.csv")

#ann\_enrich("PredictData\_subSpace500\_spin0.0.csv")