import sys

import numpy as np

from bitstring import Bits, BitStream, BitArray, ConstBitStream

import random

import statistics as stat

import math

import os

import numba

import itertools as it

from numba import jit, njit,config, threading\_layer, prange, cuda, int64

from numba.typed import List

#from HeisenHam import Hamiltonian

import net\_nstates

from newGeneration import mutation, reflection, makeNewGeneration, makeNewMlGeneration

from convergence import checkConvergence, checkFinalConv, makeFitGeneration, convInitializer, update, updateDeterminatList

from spinCalculator import spinCalculator, stateFinder

from setup import readInput

from ciRegressionFun import ann\_train

import torch

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

if model == 'HB':

from HeisenHam import Hamiltonian

if model == 'GM':

from GhoshMajumHam import Hamiltonian

mlStart = 4

mlPerSpace = 3

spaceIncrease = 200

dataFile = outputfile + ".TrainData\_subSpace.csv"

def performMCCI():

convReach = False

subBasis = []

if restart:

with open(saveBasis,"r") as fsaveB:

for i in range(subSpace):

line = fsaveB.readline()

det0 = BitArray(bin=line.strip())

subBasis.append(det0)

if (restart == False):

for i in range(len(det)):

det0 = det[i]

random.shuffle(det0)

subBasis.append(det0)

if (Ms[0] == 0):

subBasis.append(~det0)

detCopy0 = BitArray()

for i in range(len(det)):

while len(subBasis) < int(subSpace \* (i + 1) /len(det)) :

detCopy0 = subBasis[i \*2].copy()

random.shuffle(detCopy0)

if detCopy0 not in list(subBasis):

subBasis.append(detCopy0)

if (Ms[0] == 0):

if ~detCopy0 not in list(subBasis):

subBasis.append(~detCopy0)

subHam = Hamiltonian(subBasis)

lenSB = len(subBasis)

energy = np.zeros(lenSB)

ciCoef = np.zeros((nStates \* lenSB))

net\_nstates.diagonalization(hamil = subHam, n = lenSB, n1 = 3 \* lenSB, n2 = nStates, ehamil = energy, vec = ciCoef)

energyMin = energy[ 0 ]

ciCoefMin = ciCoef[ 0 : lenSB]

#print(energy[ 0 : nStates])

s2ValLIst = List()

s2ValMin = 100

targetState, s2ValDiff, energyChange = ([], ) \* 3

targetState, s2ValList, s2ValDiff, energyChange, spinChange = convInitializer()

## to store all the det and their CI coef to a data file

allDet = []

allCicoef = []

kValue = [0, 0] # to check if space size increased or not

for i in range(maxItr):

# for dynamic sub space size

k = max(0, math.floor((i - mlStart - 3)/mlPerSpace))

newSize = k \* spaceIncrease + subSpace

kValue[0] = kValue[1]

kValue[1] = k

kDiff = kValue[1] - kValue[0]

# creation oif new generation

if (i <= mlStart):

newGen, lenNewGen = makeNewGeneration(subBasis)

if (i == mlStart +1):

newline = ("\nStarting Active-Learning Protocal \n")

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

fout.write(newline)

if (i > mlStart):

newGen, lenNewGen, allDet, allCicoef = makeNewMlGeneration(subBasis, dataFile, newSize, allDet, allCicoef, k)

#print("i, lenNewGen", i, lenNewGen)

newGenHam = Hamiltonian(newGen )

energy = np.zeros(lenNewGen )

ciCoef = np.zeros(lenNewGen \* nStates )

net\_nstates.diagonalization(hamil = newGenHam, n= lenNewGen, n1 = 3 \* lenNewGen, n2 = nStates, ehamil = energy , vec = ciCoef)

s2ValList = spinCalculator(newGen, energy[ 0 : nStates ], ciCoef, lenNewGen, convReach)

#s2ValList = [s2List.append(0.0) for x in range(nStates)]

#print("end diag")

if (i < startSpinTargetItr):

targetState[1] = 0;

s2ValDiff = [10, 10] # dont want to inculde spin information on optimizations

if (i == startSpinTargetItr): # for smmoth transition from non spin target to spin target cacluations

targetState[1], s2ValDiff[1] = stateFinder(s2ValList,s2Target) # for first state of a particular spin

newline = ("\nStarting Optimization W.R.T Spin, Target State Spin Value -> %f \n\n")%(s2Target)

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

fout.write(newline)

energyMin = energy[ targetState [ 1 ] ]

s2ValDiff[0] = s2ValDiff[1]

if (i > startSpinTargetItr):

targetState[1], s2ValDiff[1] = stateFinder(s2ValList,s2Target) # for first state of a particular spin

ciCoefNew = ciCoef[(lenNewGen) \* targetState[1] : (lenNewGen) \* (targetState[1] +1)]

energyNew = energy[ targetState [ 1 ] ]

s2ValNew = s2ValList [targetState [ 1 ]]

Eith = energyMin

#print("yes")

# updata the determinants and and theie CI value list, and store them in a file

#print("length of allDet before", len(allDet))

allDet, allCicoef = updateDeterminatList(allDet, allCicoef, newGen, ciCoefNew, dataFile, kDiff)

#print("length of allDet after", len(allDet))

basis\_print = newGen.copy()

ci\_print = ciCoefNew[: lenNewGen]

#print("yes1")

subBasis, energyMin, ciCoefMin, s2ValDiff, s2ValMin, energyUpdate = checkConvergence( energyMin, energyNew, ciCoefMin, ciCoefNew, s2ValMin, s2ValNew, targetState, newGen, s2ValDiff, i, newSize)

#print("yes2")

energyChange, spinChange, convReach = checkFinalConv( energyChange, spinChange, Eith, energyMin, s2ValDiff[0], convReach)

if energyUpdate :

energyFinal, ciFinal, basisFinal = update( energy[0 : nStates], ciCoef, newGen, len(newGen) )

if (convReach == True) or (i == maxItr - 1):

if convReach:

newline = ("\nIteration Converged.\n")

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

fout.write(newline)

# print(newline)

else:

newline = ("\nReach Max Iteration Number.\n")

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

fout.write(newline)

convReach = True

#print(newline)

## Final Calculation

spinCalculator(basisFinal, energyFinal, ciFinal, len(basisFinal), convReach)

break

bF, cF = makeFitGeneration(basisFinal, ciFinal[: len(basisFinal)], len(basisFinal)) # for ordered

#bF, cF = makeFitGenerationAL(basis\_print, ci\_print, lenNewGen) # for ordered

with open( (str(outputfile) + '.basis'), "w") as fbasis:

for element in bF:

fbasis.write(element.bin +'\n')

with open( (str(outputfile) + '.ci'), "w") as fci:

for element in cF:

fci.write(str(round(float(element),6)) +'\n')