**Briefing 3**

**Fusion computing of quantum computing and network computing**

ANI-1 combines quantum computing and network computing, and is 6 orders of magnitude faster than DFT.

ANI combines quantum computing and network computing. There are mainly two keys: a database obtained by calculating a large number of molecules by density functional theory DFT. Another difficulty is that the input of the network must be a vector. It is easy to input an image, but it is more difficult to input a molecule. ANI establishes an AEV vector as input.

So when ANI calculates molecules, it no longer needs to do quantum calculations, only network calculations.

Database: ANI-1 (ANI=ANAKIN-ME) is a database labeled with the energy of all molecules composed of 8 heavy atoms (H, C, N, O) calculated by DFT. ANI-1 is GDB-11 A subset contains 57,951 molecules. The energy is calculated for neutral molecules in the singlet spin state (single point energy).

**ani-2x library: Molecules containing elements H, C, N, O, S, F, Cl.**

Therefore, ANI-1 cannot solve the Schrodinger equation to obtain eigenfunctions and eigenvalues. However, the potential energy surface, force constant, and vibration frequency can be obtained. Then the geometric configuration can be obtained from its minimum point, and the reaction energy can be obtained from the depth of its potential well. Obtain the transition state, infrared spectrum, Raman spectrum, etc. from its saddle point.

If the database is changed to material data, it should also be used for material calculation.

Running software: Torchani requires anaconda3, pytorch, pytorch\_env, ase, torchani and other software.

**NNP network:** The network consists of 3-4 hidden layers, each with 32-128 neurons is the best. ANI uses a fully connected neural network. The regression function neural network is used to predict the molecular potential energy surface.

Specifically, 768 input values, then a 128-node hidden layer, then another 128-node hidden layer, a 64-node hidden layer, and finally an output node, each single atom has a total of 124,033 The number of parameters can be optimized for the potential of the neural network.

**Network input:** The coordinate q=(q1,q2,q3) of each atom x in the molecule must be transformed into the environment vector of the atom (AEV=atom evironment vector) G\_i={G1,G2,…,Gm}. Where Gm reflects The radial and angular interaction environment around an atom must also set a cutoff function for its radial action.

**Cutoff function**

Cut-off radius Rc ~ 4.6

*f*C（*R ij*）is a continuous function with a continuous first derivative.

**Radial function**

**#** The coordinates are in **Angstrom,** and the energies you get are

# in **Hartree**

**(1) energy.py**

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

# To begin with, let's first import the modules we will use:

import torch

import torchani

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

# Let's now manually specify the device we want TorchANI to run:

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

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

# load dataset ANI2x

model = torchani.models.ANI2x(periodic\_table\_index=True).to(device)

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

# define the coordinate and species

coordinates = torch.tensor([[[ -1.21370, 0.04250, 0.69270],

[ -0.41780, -1.19830, 0.94390],

[ 0.95970, -1.20890, 1.05430],

[ -1.20960, -2.46350, 1.06580],

[ 1.77330, 0.02190, 1.29950],

[ 1.73240, -2.48610, 0.93780],

[ -0.83330, 0.96200, -0.31410],

[ -1.60670, 2.09760, -0.58340],

[ -2.77860, 2.34640, 0.14950],

[ -3.17520, 1.43990, 1.14700],

[ -2.40790, 0.29700, 1.40890],

[ -1.03260, -3.31370, 2.18260],

[ -1.74590, -4.50920, 2.30710],

[ -2.66230, -4.92150, 1.31330],

[ -2.86770, -4.05320, 0.21460],

[ -2.17090, -2.84480, 0.10140],

[ 1.40210, 0.95300, 2.29920],

[ 2.18910, 2.08060, 2.56280],

[ 3.36680, 2.30930, 1.83220],

[ 3.75470, 1.39060, 0.84220],

[ 2.97340, 0.25580, 0.58590],

[ 1.54130, -3.33790, -0.17540],

[ 2.23720, -4.54360, -0.29690],

[ 3.14930, -4.96480, 0.69680],

[ 3.36620, -4.09770, 1.79380],

[ 2.68820, -2.87820, 1.90320],

[ -3.36040, -6.23110, 1.41650],

[ 3.83280, -6.28240, 0.59340],

[ -3.83700, -6.71240, 2.65710],

[ -4.45650, -7.96530, 2.75670],

[ -4.61840, -8.76900, 1.61570],

[ -4.15550, -8.30310, 0.37580],

[ -3.53820, -7.04950, 0.27610],

[ 4.30960, -6.76410, -0.64680],

[ 4.91450, -8.02380, -0.74890],

[ 5.05990, -8.83410, 0.38930],

[ 4.59710, -8.36770, 1.62910],

[ 3.99530, -7.10690, 1.73160],

[ 0.07300, 0.77380, -0.89350],

[ -1.29420, 2.79020, -1.36870],

[ -3.38050, 3.23480, -0.05770],

[ -4.08890, 1.62050, 1.72010],

[ -2.73240, -0.41060, 2.17520],

[ -0.31630, -3.03030, 2.95590],

[ -1.55950, -5.15280, 3.16880],

[ -3.59540, -4.32560, -0.55350],

[ -2.35800, -2.19480, -0.75570],

[ 0.49200, 0.77960, 2.87710],

[ 1.88220, 2.78250, 3.34230],

[ 3.97830, 3.19190, 2.03490],

[ 4.67190, 1.55670, 0.27050],

[ 3.29170, -0.46060, -0.17490],

[ 0.82720, -3.04710, -0.94810],

[ 2.04130, -5.18720, -1.15670],

[ 4.09110, -4.37840, 2.56160],

[ 2.88590, -2.22780, 2.75770],

[ -3.73640, -6.08890, 3.54850],

[ -4.81930, -8.31260, 3.72740],

[ -5.09760, -9.74750, 1.69440],

[ -4.26770, -8.92250, -0.51780],

[ -3.15370, -6.71320, -0.68910],

[ 4.22200, -6.13460, -1.53520],

[ 5.27880, -8.37120, -1.71900],

[ 5.52720, -9.81820, 0.30910],

[ 4.69620, -8.99330, 2.51980],

[ 3.61060, -6.76950, 2.69640],

[ 3.61060, -6.76950, 2.69640]]],

requires\_grad=True, device=device)

# In periodic table, C = 6 and H = 1

species = torch.tensor([[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]], device=device)

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

# compute energy and force:

energy = model((species, coordinates)).energies

derivative = torch.autograd.grad(energy.sum(), coordinates)[0]

force = -derivative

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

# print to see the result:

print('Energy:', energy.item())

print('Force:', force.squeeze())

**(2) vibration.py**

import ase

import ase.optimize

import torch

import torchani

import math

from ase import Atoms

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

# the device we want TorchANI to run:

device = torch.device('cpu')

model = torchani.models.ANI1x(periodic\_table\_index=True).to(device).double()

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

# construct a molecule and do structure optimization:

molecule = ase.Atoms('C38H28', positions=[

(-1.21370, 0.04250, 0.69270),

(-0.41780, -1.19830, 0.94390),

(0.95970, -1.20890, 1.05430),

(-1.20960, -2.46350, 1.06580),

(1.77330, 0.02190, 1.29950),

(1.73240, -2.48610, 0.93780),

(-0.83330, 0.96200, -0.31410),

(-1.60670, 2.09760, -0.58340),

(-2.77860, 2.34640, 0.14950),

(-3.17520, 1.43990, 1.14700),

(-2.40790, 0.29700, 1.40890),

(-1.03260, -3.31370, 2.18260),

(-1.74590, -4.50920, 2.30710),

(-2.66230, -4.92150, 1.31330),

(-2.86770, -4.05320, 0.21460),

(-2.17090, -2.84480, 0.10140),

(1.40210, 0.95300, 2.29920),

(2.18910, 2.08060, 2.56280),

(3.36680, 2.30930, 1.83220),

(3.75470, 1.39060, 0.84220),

(2.97340, 0.25580, 0.58590),

(1.54130, -3.33790, -0.17540),

(2.23720, -4.54360, -0.29690),

(3.14930, -4.96480, 0.69680),

(3.36620, -4.09770, 1.79380),

(2.68820, -2.87820, 1.90320),

(-3.36040, -6.23110, 1.41650),

(3.83280, -6.28240, 0.59340),

(-3.83700, -6.71240, 2.65710),

(-4.45650, -7.96530, 2.75670),

(-4.61840, -8.76900, 1.61570),

(-4.15550, -8.30310, 0.37580),

(-3.53820, -7.04950, 0.27610),

(4.30960, -6.76410, -0.64680),

(4.91450, -8.02380, -0.74890),

(5.05990, -8.83410, 0.38930),

(4.59710, -8.36770, 1.62910),

(3.99530, -7.10690, 1.73160),

(0.07300, 0.77380, -0.89350),

(-1.29420, 2.79020, -1.36870),

(-3.38050, 3.23480, -0.05770),

(-4.08890, 1.62050, 1.72010),

(-2.73240, -0.41060, 2.17520),

(-0.31630, -3.03030, 2.95590),

(-1.55950, -5.15280, 3.16880),

(-3.59540, -4.32560, -0.55350),

(-2.35800, -2.19480, -0.75570),

(0.49200, 0.77960, 2.87710),

(1.88220, 2.78250, 3.34230),

(3.97830, 3.19190, 2.03490),

(4.67190, 1.55670, 0.27050),

(3.29170, -0.46060, -0.17490),

(0.82720, -3.04710, -0.94810),

(2.04130, -5.18720, -1.15670),

(4.09110, -4.37840, 2.56160),

(2.88590, -2.22780, 2.75770),

(-3.73640, -6.08890, 3.54850),

(-4.81930, -8.31260, 3.72740),

(-5.09760, -9.74750, 1.69440),

(-4.26770, -8.92250, -0.51780),

(-3.15370, -6.71320, -0.68910),

(4.22200, -6.13460, -1.53520),

(5.27880, -8.37120, -1.71900),

(5.52720, -9.81820, 0.30910),

(4.69620, -8.99330, 2.51980),

(3.61060, -6.76950, 2.69640) ], calculator=model.ase())

opt = ase.optimize.BFGS(molecule)

opt.run(fmax=1e-6)

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

# extract coordinates and species from ASE to use it directly with TorchANI:

species = torch.tensor(molecule.get\_atomic\_numbers(), device=device, dtype=torch.long).unsqueeze(0)

coordinates = torch.from\_numpy(molecule.get\_positions()).unsqueeze(0).requires\_grad\_(True)

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

# TorchANI needs the masses of elements in AMU to compute vibrations.

# The masses in AMU can be obtained from a tensor with atomic numbers # by using this utility:

masses = torchani.utils.get\_atomic\_masses(species)

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

# To do vibration analysis

energies = model((species, coordinates)).energies

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

# use the energy graph to compute analytical **Hessian matrix**:

hessian = torchani.utils.hessian(coordinates, energies=energies)

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

# The Hessian matrix should have shape `(1, 9, 9)`, where 1 means there is

# only one molecule to compute, 9 means `3 atoms \* 3D space = 9 degree # of freedom`.

print(hessian.shape)

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

# compute vibrational frequencies. unit is cm^-1

freq, modes, fconstants, rmasses = torchani.utils.vibrational\_analysis(masses, hessian, mode\_type='MDU')

torch.set\_printoptions(precision=3, sci\_mode=False)

print('Frequencies (cm^-1):', freq[6:])

print('Force Constants (mDyne/A):', fconstants[6:])

print('Reduced masses (AMU):', rmasses[6:])

print('Modes:', modes[6:])

) are adjustable parameters

The parameter η is used to change the width of the Gaussian distribution, and the purpose of R s is to move the center of the peak.

**Angular function**

, are adjustable parameters

**Network output:** Since the label of the data set ANI-1 used is energy, the regression output is the energy of the input molecule. Potential energy surface - here, an important result is that E\_n (R) is both the electron energy of the system when the nuclear is fixed, and Is the potential energy of nuclear motion. This is the potential energy surface. For a molecule composed of N atoms, the potential energy is a super curve with 3N coordinates. Omit the 6 coordinates of translation and rotation, it is 3N-6 coordinates (linear molecule is 3N-5 coordinates). Fixing the nuclear position R, the calculated energy of the electronic system is called the single-point energy. Because the given R is only the value of one point of the potential energy. There are many R values under the potential energy. Since R is 3N-6 coordinates, N is the number of atoms. Generally, we can only scan a certain variable of the molecule to obtain the curve of the potential energy or single point energy with this variable.

**Examples: energy\_force.py**

"""

Computing Energy and Force Using Models Inside Model Zoo

========================================================

TorchANI has a model zoo trained by NeuroChem. These models are shipped with

TorchANI and can be used directly.

NeuroChem run GPU using C++

"""

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

# step 1: give modules and device

# To begin with, let's first import the modules we will use:

import torch

import torchani

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

# Let's now manually specify the device we want TorchANI to run:

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

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

# Let's now load the built-in ANI-1ccx models. The builtin ANI-1ccx contains 8

# models trained with diffrent initialization. Predicting the energy and force

# using the average of the 8 models outperform using a single model, so it is

# always recommended to use an ensemble, unless the speed of computation is an

# issue in your application.

# The ``periodic\_table\_index`` arguments tells TorchANI to use element index

# in periodic table to index species. If not specified, you need to use

# 0, 1, 2, 3, ... to index species

# step 2: load dataset

model = torchani.models.ANI2x(periodic\_table\_index=True).to(device) # ANI2x is dataset with HCHOSFCl 7 elements

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

# Now let's define the coordinate and species. If you just want to compute the

# energy and force for a single structure like in this example, you need to

# make the coordinate tensor has shape ``(1, Na, 3)`` and species has shape

# ``(1, Na)``, where ``Na`` is the number of atoms in the molecule, the

# preceding ``1`` in the shape is here to support batch processing like in

# training. If you have ``N`` different structures to compute, then make it

# ``N``.

# .. note:: The coordinates are in Angstrom, and the energies you get are in Hartree

# step 3: set up molecule and species

coordinates = torch.tensor([[[0.03192167, 0.00638559, 0.01301679],

[-0.83140486, 0.39370209, -0.26395324],

[-0.66518241, -0.84461308, 0.20759389],

[0.45554739, 0.54289633, 0.81170881],

[0.66091919, -0.16799635, -0.91037834]]],

requires\_grad=True, device=device)

# In periodic table, C = 6 and H = 1

species = torch.tensor([[6, 1, 1, 1, 1]], device=device) # CH4甲烷

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

# Now let's compute energy and force:

# step 4: run computing for energy and force

energy = model((species, coordinates)).energies

derivative = torch.autograd.grad(energy.sum(), coordinates)[0]

force = -derivative

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

# And print to see the result:

# step 5: display results

print('Energy:', energy.item())

print('Force:', force.squeeze())