TorchANI Worknotes

Fadjar Fathurrahman

1 Installation

Prerequisites: PyTorch

Recommendation: install to user directory.

The usual steps are applicable:

- `python setup.py build`
- `python setup.py install --user`

TorchANI will be installed at (for example, the version number may vary):

```
$HOME/.local/lib/python3.8/site-packages/torchani-2.1.2-py3.8.egg/
```

To check the installation, go to directory examples and run the `energy_force.py` example. TorchANI will download the necessary files (paramaters for the neural network) so the script might run slowly in the first call. The parameters will be put under the `resources` subdirectory of TorchANI.

There is warning when using Torch v-1.6:

```
/home/efefer/.local/lib/python3.8/site-packages/torchani-2.1.2.dev12+g42442af-py3.8.egg/torchani/aev.py:
↪  UserWarning: This overload of nonzero is deprecated:
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at
        ↪  /opt/conda/conda-bld/pytorch_1595629395347/work/torch/csrc/utils/python_arg_parser.cpp:766.)
in_cutoff = (distances <= cutoff).nonzero()
```

The fix seems to be:

```
in_cutoff = torch.nonzero(distances <= cutoff, as_tuple=False)
```

2 CPU vs GPU comparison

Script `test_cnt_5x5.py`. Timing using time command in Linux.

```
from ase.build import nanotube
from ase.md.langevin import Langevin
from ase.optimize import BFGS
from ase import units

import torch
import my_torchani

# Now let's set up a crystal
atoms = nanotube(5, 5, length=10, bond=1.420, symbol='C')
print(len(atoms), "atoms in the cell")
atoms.set_pbc([True,True,True])
Lz = atoms.cell.array[2,2]
Lx = 20.0
Ly = 20.0
atoms.set_cell([Lx, Ly, Lz])
atoms.center()
atoms.write("STRUCT_cnt_5x5.xsf")

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device = "cpu" # set manually to cpu
print("device = ", device)

# Now let's create a calculator from builtin models:
calculator = my_torchani.models.ANI1ccx().to(device).ase()

# Now let's set the calculator for ``atoms``:
atoms.set_calculator(calculator)

# Now let's minimize the structure:
print("Begin minimizing...")
opt = BFGS(atoms)
opt.run(fmax=0.001)
print()


# Now create a callback function that print interesting physical quantities:
def printenergy(a=atoms):
    # Function to print the potential, kinetic and total energy.
    epot = a.get_potential_energy() / len(a)
    ekin = a.get_kinetic_energy() / len(a)
    print('Energy per atom: Epot = %.3feV  Ekin = %.3feV (T=%3.0fK)  '
          'Etot = %.3feV' % (epot, ekin, ekin / (1.5 * units.kB), epot + ekin))

# We want to run MD with constant energy using the Langevin algorithm
# with a time step of 1 fs, the temperature 300K and the friction
# coefficient to 0.02 atomic units.
dyn = Langevin(atoms, 1 * units.fs, 300 * units.kB, 0.2)
dyn.attach(printenergy, interval=1)
```

```
# Now run the dynamics:
print("Beginning dynamics...")
printenergy()
dyn.run(500)
```

Timing result:

```
# CPU
real        1m35.680s
user        1m26.560s
sys         0m9.126s

# GPU
real        0m18.612s
user        0m15.110s
sys         0m3.454s
```