Usage of the code for NN-supported GRASP computations

Pavlo Bilous

July 4, 2023

1 Preparation for computations

1.1 Get the code

The code can be cloned from the GitHub repo:
`https://github.com/pavlobilous/neural_grasp`

The main file here is the file `neural_grasp.py` containing the class `NeuralGraspManager` responsible for NN-suppoted computations. The user is supposed to write his/her own Python scripts importing and using this class. There is an auxiliary script `process_grasp_inp.py` for creating a NumPy array for the CSF set to be sorted out (see below). The user is supposed to write also a Python script employing `process_grasp_inp.py`. In this demo example, we consider computations of Re ground state. The exemplary user files are called `Re187_....`

1.2 Rebuild GRASP2018

The code works with GRASP2018, but GRASP has to be rebuild after the file `src/appl/rmcdhf90_mpi/csfwgt.f90` is replaced with the attached file `csfwgt_rmcdhf_mpi.f90`. With this, the weights of all CSFs will be written in the file `csfwgt_rmcdhf_mpi.dat` each time the `rmcdhf_mpi` program is run (this file will be placed in the folder `tmp_mpi/000`). The order of the CSFs in this file is the same as in the input `rcsf` file.

1.3 GRASP input

Attached are the `isodata` file and the radial wave function file `rwfn.prim.inp`. Place them in the folder with the future NN-supported computations. The `rcsf` files should be created in the following way.

First, create `rcsf`-files from the attached CSF-files `gendata1-2` and `gendata1` by feeding them into the `rcsfgenerate` program. The CSF set originating from `gendata1-2` (let's call the corresponding file `rcsf1-2.inp`) is the "big" CSF set which we cannot computed directly (in this case we can, but only because this is a demo version). The CSF set originating from `gendata1` (let's call the corresponding file `rcsf1.inp`) is the so called *primary subset* of the "big" set which will be always included in the GRASP runs and won't be exposed to the NN.

The "big" `rcsf`-file has to be now restructured such that the primary CSFs come first. Use the `rcsfzerofirst` program on the files `rcsf1-2.inp` and `rcsf1.inp` to achieve this. Rename the obtained file to `rcsf.full` and the primary set file called before `rcsf1.inp` to `rcsf.head`.

1.4 Creating NumPy file with the “big” set

Run the Python script `Re187_gen_rcsf.py`. The input files are `rcsf.full` and `rcsf.head`, the output is the file `rcsf.npy` needed for the NN-supported computations. Note that in this Python script there is additional input $2J_{\mathrm{total}} = 5$, which should be changed correspondingly for computations other than the Re ground state.

2 NN-supported computations

2.1 Initial iteration on random CSFs

Since at the beginning the NN is untrained and thus useless, the first iteration is performed by picking randomly CSFs from the “big” set (or “pool”) to be sorted out (of course, excluding the primary CSFs). Run the script `Re187_init_on_random.py` which will pick randomly 1% of CSFs from the pool and write them to an `rcsf.inp` file on top of the primary subset. At the same time, it creates a blanc NN already at this stage, which is saved in the folder `save_init_random` together with some NumPy arrays needed for further computations. Note that in this script the NN is not trained, since the CSF weights are not known yet.

As soon as the script is finished, GRASP should be run by the user on the formed `rcsf.inp` file. After that, copy the `csfwgt_rmcdhf_mpi.dat` file from the folder `tmp_mpi/000` to the folder with the computations — this is required for running the next Python script. Store the obtained energy value.

Note that in this demo example, the Python code never runs GRASP by itself (though it can, but this is good only for small computations). It forms only `rcsf.inp` files, and the user is expected to run GRASP “manually” on it. The idea behind this splitting is that for bigger computations, the NN training can be performed on a GPU, whereas GRASP requires many CPU nodes. The user would then submit two separate jobs requesting different kinds of resources.

2.2 Performing the first NN-supported iteration

Now we do the first NN-supported iteration by running the script `Re187_iter_on_cutlog.py`. This script has the cutoff value $10^{-8.6}$ (see the variable `cutlog`) which determined which CSFs are *important* and which are *unimportant*. First, the NN and the internal NumPy arrays are read from the folder `save_init_random`. The NN trains on the CSFs randomly picked in the previous iteration, for which the weights (and thus the binary classification as important / unimportant) are already known. Then the trained NN is applied to the whole pool and the next `rcsf.inp` file is formed. The trained NN and the internal NumPy arrays is saved in the folder `save_cutlog_-8.6` for the next iteration.

Now run GRASP, copy the `csfwgt_rmcdhf_mpi.dat` file from the folder `tmp_mpi/000` to the folder with the computations, and store the obtained energy value.

2.3 Performing subsequent NN-supported iteration

In the same manner we can run further NN-supported iterations. For this, the script `Re187_iter_on_cutlog.py` needs to be changed at two places

- The input folder should be now not `save_init_random` but `save_cutlog_-8.6` (or the last folder saved). Edit the line `ngm.load_state("save_init_random",fit_params)` correspondingly.

- The cutoff value has to be changed. Edit the line `cutlog=-8.6` for the values -9.2, -9.8, -10.4, -11.0 for the subsequent iterations.

As usually, after this Python script is finished, run GRASP, copy the `csfwgt_rmcdhf_mpi.dat` file from the folder `tmp_mpi/000` to the folder with the computations, and store the obtained energy value. Make sure that the obtained energies converge to the “true” value on the full CSF set.