HDNNP Documentation

Release 0.5.1.dev

masayoshi.ogura

Mar 27, 2019

Contents:

CHAPTER 1

What is HDNNP?

This program is an implementation of HDNNP that is suggested by Behler *et al* [Ref].
HDNNP stands for **High Dimensional Neural Network Potential**.
HDNNP is one of machine learning potentials that is used to reduce calculation cost of DFT(Density Functional Theory) calculation.

Currently, energy and force prediction using symmetry function have been implemented.

CHAPTER 2

How to install HDNNP

2.1 Python installation

We recommend that you install python using pyenv, because non-sudo user can install any python version on any computer.
We confirmed that this program works only with python 3.6.7.

```
(on Linux)
$ git clone https://github.com/yyuu/pyenv.git ~/.pyenv
(on MacOS)
$ brew install pyenv

$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bash_profile
$ echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bash_profile
$ echo 'eval "$(pyenv init -)"' >> ~/.bash_profile
$ source ~/.bash_profile

$ pyenv install 3.6.7
```

2.2 Get source code

Note:

This program is now under development, not uploaded to PyPI.
You have to get source code and install it manually.

```
$ git clone https://github.com/ogura-edu/HDNNP.git
```

2.3 Install dependencies and this program

2.3.1 Via pipenv

```
$ cd HDNNP/
$ pyenv local 3.6.7
$ pip install pipenv
$ pipenv install --dev

(activate)
$ pipenv shell

(for example:)
(HDNNP) $ hdnnpy train

(deactivate)
(HDNNP) $ exit
```

2.3.2 Via anaconda

Anaconda also can be installed by pyenv.

```
$ cd HDNNP/
$ pyenv install anaconda3-xxx
$ pyenv local anaconda3-xxx
$ conda env create -n HDNNP --file condaenv.yaml

(activate)
$ conda activate HDNNP

(for example:)
(HDNNP) $ hdnnpy train

(deactivate)
(HDNNP) $ conda deactivate
```

2.3.3 Via raw pip

You can install all dependent packages manually. The dependent packages are written in `Pipfile`, `condaenv.yaml` or `requirements.txt`.

```
$ cd HDNNP/
$ pip install PKG1 PKG2 ...
$ pip install --editable .
```

CHAPTER 3

How to use HDNNP

3.1 Data generation

Usually, HDNNP is used to reduce cost by learning the result of DFT(Density Functional Theory) calculation that is high accuracy and high cost.
Therefore, first step is to generate training dataset using DFT calculation such as ab-initio MD calculation.

3.2 Pre-processing

HDNNP training application supports only .xyz file format.

We prepare a python script to convert the output file of VASP such as `OUTCAR` to .xyz format file, but in the same way you can convert the output of other DFT calculation program to .xyz format file.
Inside this program, file format conversion is performed using ASE package.

3.3 Training

3.3.1 Configuration

A default configuration file for training is located in `examples/training_config.py`.

`training_config.py` consists of some subclasses that inherits `traitlets.config.Configurable`:

- c.Application.xxx
- c.TrainingApplication.xxx
- c.DatasetConfig.xxx
- c.ModelConfig.xxx
- c.TrainingConfig.xxx

Following configurations are required, and remaining configurations are optional.

- c.DatasetConfig.parameters
- c.ModelConfig.layers
- c.TrainingConfig.data_file
- c.TrainingConfig.batch_size
- c.TrainingConfig.epoch
- c.TrainingConfig.order
- c.TrainingConfig.loss_function
- c.TrainingConfig.interval
- c.TrainingConfig.patients

For details of each setting, see `training_config.py`

3.3.2 Command line interface

Execute the following command in the directory where `training_config.py` is located.

```
$ hdnnpy train
```

Note:

Currently, if output directory set by `c.TrainingConfig.out_dir` already exists, it overwrites the existing file in the directory.
If you want to avoid this, please change `c.TrainingConfig.out_dir` for each execution.

3.4 Prediction

3.4.1 Configuration

A default configuration file for prediction is located in `examples/prediction_config.py`.

`prediction_config.py` consists of some subclasses that inherits `traitlets.config.Configurable`:

- c.Application.xxx
- c.PredictionApplication.xxx
- c.PredictionConfig.xxx

Following configurations are required, and remaining configurations are optional.

- c.PredictionConfig.data_file
- c.PredictionConfig.order

For details of each setting, see `prediction_config.py`

3.4.2 Command line interface

Execute the following command in the directory where `prediction_config.py` is located.

```
$ hdnnpy predict
```

3.5 Post-processing

It is possible to calculate MD simulation with LAMMPS using trained HDNNP.
However, it is also under development.
We welcome your comments and suggestions.

HDNNP-LAMMPS interface program

3.5.1 Command line interface

Execute the following command.

```
$ hdnnpy convert
```

2 command line options are available, and no config file is used in this command.
To see details of these options, use

```
$ hdnnpy convert -h
```

CHAPTER 4

Execution example

4.1 GaN interatomic potential

In this section, show you an execution example of HDNNP training using 1st order differentiation of interatomic potential (e.g. interatomic forces) of GaN

4.1.1 Data file

Prepare a .xyz format file which have some structures with energy and force data.

`GaN.xyz`

```
32
Lattice="6.46474316 0.0 0.0 -3.23237159 5.5986318 0.0 0.0 0.0 10.53232454"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-194.5164333 tag=CrystalGa16N16␣
↪pbc="T T T"
Ga       1.61619000       0.93311000       2.62845000       0.00000300       0.
↪00001200      -0.00570900
Ga       3.23237000       3.73242000       2.62845000       0.00003900      -0.
↪00004700      -0.00571500
```

(continues on next page)

(continued from previous page)

```
Ga       4.84856000       0.93311000       2.62845000       0.00000400      -0.
↪00001100      -0.00563600
Ga      -0.00000000       3.73242000       7.89461000      -0.00003800       0.
↪00003200      -0.00564200
Ga       1.61619000       0.93311000       7.89461000       0.00006100      -0.
↪00001800      -0.00571100
Ga       3.23237000       3.73242000       7.89461000       0.00002100      -0.
↪00006400      -0.00572000
Ga       4.84856000       0.93311000       7.89461000      -0.00003200      -0.
↪00002300      -0.00565600
Ga      -0.00000000       3.73242000       2.62845000       0.00002100      -0.
↪00002000      -0.00565100
Ga      -0.00000000       1.86621000       5.26153000      -0.00006900       0.
↪00005900      -0.00572300
Ga       1.61619000       4.66553000       5.26153000      -0.00002700       0.
↪00008200      -0.00571900
Ga       3.23237000       1.86621000       5.26153000       0.00001800      -0.
↪00001400      -0.00566500
Ga      -1.61619000       4.66553000      10.52769000      -0.00002700      -0.
↪00002600      -0.00566900
Ga      -0.00000000       1.86621000      10.52769000      -0.00002200       0.
↪00008500      -0.00568700
Ga       1.61619000       4.66553000      10.52769000       0.00000600      -0.
↪00002400      -0.00574300
Ga       3.23237000       1.86621000      10.52769000       0.00000100       0.
↪00007600      -0.00564000
Ga      -1.61619000       4.66553000       5.26153000       0.00002200      -0.
↪00000200      -0.00568800
N        1.61619000       0.93311000       4.61253000       0.00005500      -0.
↪00002000      -0.00041000
N        3.23237000       3.73242000       4.61253000       0.00003600      -0.
↪00000900      -0.00037900
N        4.84856000       0.93311000       4.61253000      -0.00004100       0.
↪00000700      -0.00041100
N       -0.00000000       3.73242000       9.87869000      -0.00001300      -0.
↪00003500      -0.00042500
N        1.61619000       0.93311000       9.87869000       0.00001200       0.
↪00002900      -0.00040900
N        3.23237000       3.73242000       9.87869000       0.00002700      -0.
↪00006200      -0.00041700
N        4.84856000       0.93311000       9.87869000      -0.00000400       0.
↪00002500      -0.00041500
N       -0.00000000       3.73242000       4.61253000      -0.00004500      -0.
↪00000400      -0.00041800
N       -0.00000000       1.86621000       1.97945000       0.00000000      -0.
↪00000800      -0.00034400
N        1.61619000       4.66553000       1.97945000      -0.00000200       0.
↪00000500      -0.00033700
N        3.23237000       1.86621000       1.97945000       0.00001700       0.
↪00001600      -0.00036100
N       -1.61619000       4.66553000       7.24561000       0.00002800      -0.
↪00002300      -0.00036000
N       -0.00000000       1.86621000       7.24561000      -0.00008200       0.
↪00001500      -0.00043200
N        1.61619000       4.66553000       7.24561000      -0.00002200       0.
↪00004200      -0.00040100
N        3.23237000       1.86621000       7.24561000       0.00001900      -0.
↪00001200      -0.00039500
```

(continues on next page)

(continued from previous page)

```
N      -1.61619000       4.66553000       1.97945000       0.00000400      -0.
↪00001800      -0.00046000
32
Lattice="6.46474316 0.0 0.0 -3.23237159 5.5986318 0.0 0.0 0.0 10.53232454"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-169.96635976 tag=CrystalGa16N16␣
↪pbc="T T T"
Ga       1.44265000       1.46790000       2.04947000      -0.95595000      -3.
↪56110800       2.54045000
Ga       2.88538000       4.34404000       2.89380000       4.75932000      -2.
↪04809500      -1.43108200
Ga       4.38372000       0.68215000       2.61606000       0.15090500       6.
↪97113700       2.40537400
Ga       0.47836000       3.95213000       7.90284000      -3.31821700      -0.
↪13409600      -0.21437100
Ga       1.82415000       1.43420000       8.18380000      -0.78327100      -2.
↪70531000      -3.50469000
Ga       3.49351000       3.96284000       7.92622000       1.84595600      -0.
↪42627100      -0.16593100
Ga       5.17229000       0.83662000       7.71745000      -0.46937900       1.
↪21688400       1.11923500
Ga      -0.04508000       3.95689000       2.71946000      -3.88117900      -1.
↪84159800       0.64959300
Ga      -0.96518000       1.98086000       5.22137000       1.12890800      -1.
↪31857500      -0.37168600
Ga       1.18573000       3.20454000       5.22045000       1.58317800       1.
↪58466500       0.77557000
Ga       2.91073000       1.45415000       5.60119000      -0.29420600      -1.
↪79185700      -2.55652100
Ga      -0.99634000       4.45389000       0.07004000      -2.39983600       3.
↪43545000       1.27018200
Ga       0.17764000       1.60544000      10.36435000       6.30208700       4.
↪30252400       2.73199900
Ga       2.35420000       4.13573000       0.39168000      -1.28509600      -0.
↪64262000      -3.92936300
...
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.3605335 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.60815000       0.92846000       2.61537000       0.00057000      -0.
↪00032400      -0.00131800
Ga       0.00000000       1.85693000       5.23535000      -0.00055000       0.
↪00030900      -0.00128000
N       1.60815000       0.92846000       4.58958000       0.00038300      -0.
↪00020300       0.00049500
N       0.00000000       1.85693000       1.96960000      -0.00030900       0.
↪00021200       0.00050600
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.04284841 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.56998000       1.01961000       2.64712000       0.37879200      -0.
↪65345000      -0.84588100
Ga       0.00233000       1.78610000       5.21359000       1.53422400       0.
↪01126800       0.83092200
N       1.80998000       0.78162000       4.55671000      -1.91098000       0.
↪49960800      -0.07141600
```

(continues on next page)

(continued from previous page)

```
N      -0.02338000       1.90257000       1.95274000       0.00855700       0.
↪14604000       0.09234500
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.07370026 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.68022000       0.78468000       2.59601000      -0.77026300       1.
↪15126700       0.71828100
Ga      -0.04831000       1.97869000       0.01593000      -1.05203000       0.
↪42443800      -0.31339000
N       1.47544000       1.12447000       4.57171000       1.50854300      -1.
↪32922700      -0.04524600
N       0.01431000       1.77059000       1.98155000       0.31937700      -0.
↪24596800      -0.35639000
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.06789171 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.55216000       1.03346000       2.59780000       1.76477100      -1.
↪33788800       0.62275500
Ga       0.04645000       1.78043000       0.02483000      -0.39888700      -0.
↪84820500      -0.84426800
N       1.59299000       0.75442000       4.54056000       0.36047300       1.
↪45854900       0.51138400
N       0.06265000       1.88907000       1.95951000      -1.73396900       0.
↪72932900      -0.27762300
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.10933618 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.62285000       0.92354000       2.56898000      -0.87387700       0.
↪84344000       1.29437700
Ga      -0.00655000       1.82730000       0.04373000       0.63633100       1.
↪10065300      -1.07564600
N       1.65007000       1.03662000       4.56438000      -0.83168500      -1.
↪16592600       0.26072300
N      -0.08253000       1.92082000       1.98507000       1.07124400      -0.
↪78418500      -0.47994500
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.15961153 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.61929000       0.86275000       2.60668000       0.91655600       0.
↪12884500       0.02524600
Ga      -0.02746000       1.90759000       0.02534000      -0.00425900       0.
↪48361500      -1.32527900
N       1.57325000       1.05930000       4.54898000       0.29235100      -0.
↪94998800       0.25695700
N       0.11613000       1.80106000       1.90435000      -1.21017800       0.
↪33509300       1.05032200
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-23.90497111 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.57753000       1.01962000       2.53889000      -0.58498700       0.
↪38561600       1.95812800
Ga       0.05221000       1.77667000       0.06084000      -0.50913400      -1.
↪39207300      -1.16507600
```

(continues on next page)

(continued from previous page)

```
N       1.60109000       0.71987000       4.62834000       0.25821000       2.
↪35785600      -0.69708500
N      -0.10050000       2.01120000       1.98576000       0.83273600      -1.
↪35617800      -0.10520400
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-24.17936965 tag=CrystalGa2N2 pbc=
↪"T T T"
Ga       1.65588000       0.84325000       2.61391000      -0.48280700       0.
↪58352400      -0.06140200
Ga      -0.05236000       1.91994000       0.00989000       1.13163900       0.
↪73695700      -0.46324400
N       1.63413000       1.09260000       4.55873000      -1.08709100      -1.
↪30806300       0.05205700
N      -0.00295000       1.80336000       1.93549000       0.44154800      -0.
↪01662100       0.47920500
4
Lattice="3.21629013 0.0 0.0 -1.60814507 2.78538896 0.0 0.0 0.0 5.23996246"␣
↪Properties=species:S:1:pos:R:3:forces:R:3 energy=-23.82707164 tag=CrystalGa2N2 pbc=
↪"T T T"
...
```

4.1.2 Config file

`training_config.py` (necessary parts picked up)

```
c.TrainingApplication.verbose = True

c.DatasetConfig.parameters = {
   'type1': [
       (5.0,),
       ],
   'type2': [
       (5.0, 0.01, 2.0),
       (5.0, 0.01, 3.2),
       (5.0, 0.01, 3.8),
       (5.0, 0.1, 2.0),
       (5.0, 0.1, 3.2),
       (5.0, 0.1, 3.8),
       (5.0, 1.0, 2.0),
       (5.0, 1.0, 3.2),
       (5.0, 1.0, 3.8),
       ],
   'type4': [
       (5.0, 0.01, -1, 1),
       (5.0, 0.01, -1, 2),
       (5.0, 0.01, -1, 4),
       (5.0, 0.01, 1, 1),
       (5.0, 0.01, 1, 2),
       (5.0, 0.01, 1, 4),
       (5.0, 0.1, -1, 1),
       (5.0, 0.1, -1, 2),
       (5.0, 0.1, -1, 4),
       (5.0, 0.1, 1, 1),
       (5.0, 0.1, 1, 2),
```

(continues on next page)

(continued from previous page)

```
        (5.0, 0.1, 1, 4),
        (5.0, 1.0, -1, 1),
        (5.0, 1.0, -1, 2),
        (5.0, 1.0, -1, 4),
        (5.0, 1.0, 1, 1),
        (5.0, 1.0, 1, 2),
        (5.0, 1.0, 1, 4),
        ],
    }

c.DatasetConfig.preprocesses = [
    ('pca', (), {}),
    ]

c.ModelConfig.layers = [
    (90, 'tanh'),
    (90, 'tanh'),
    (1, 'identity'),
    ]

c.TrainingConfig.batch_size = 100

c.TrainingConfig.data_file = 'data/GaN.xyz'

c.TrainingConfig.epoch = 1000

c.TrainingConfig.interval = 10

c.TrainingConfig.loss_function = (
    'first_only',
    {}
    )

c.TrainingConfig.lr_decay = 1.0e-6

c.TrainingConfig.order = 1

c.TrainingConfig.out_dir = 'output'

c.TrainingConfig.patients = 5

c.TrainingConfig.scatter_plot = True
```

4.1.3 command line log

Once edited configuration file `training_config.py`, you just do one command `hdnnpy trian`.

```
$ hdnnpy train

Construct sub dataset tagged as "CrystalGa16N16"
Successfully loaded & made needed symmetry_function dataset from <workdir>/data/
↪CrystalGa16N16/symmetry_function.npz
Successfully loaded & made needed interatomic_potential dataset from <workdir>/data/
↪CrystalGa16N16/interatomic_potential.npz
```

(continues on next page)

(continued from previous page)

```
Initialized PCA parameters for Ga
    Feature dimension: 74 => 74
    Cumulative contribution rate = 0.9999999403953552


Initialized PCA parameters for N
    Feature dimension: 74 => 74
    Cumulative contribution rate = 1.0000001192092896

Construct sub dataset tagged as "CrystalGa2N2"
Successfully loaded & made needed symmetry_function dataset from <workdir>/data/
↪CrystalGa2N2/symmetry_function.npz
Successfully loaded & made needed interatomic_potential dataset from <workdir>/data/
↪CrystalGa2N2/interatomic_potential.npz
Saved PCA parameters to <workdir>/output/preprocess/pca.npz.
early stopping: operator is less
epoch       iteration   main/RMSE/force  main/RMSE/total  val/main/RMSE/force  val/
↪main/RMSE/total
1           14          1.20575          1.20575          1.21576              1.21576
2           28          1.08758          1.08758          1.06121              1.06121
3           42          0.895798         0.895798         0.865482             0.
↪865482
4           55          0.685623         0.685623         0.694789             0.
↪694789
5           69          0.560702         0.560702         0.603832             0.
↪603832
6           83          0.509542         0.509542         0.570984             0.
↪570984
7           97          0.486743         0.486743         0.552533             0.
↪552533
8           110         0.468966         0.468966         0.540375             0.
↪540375
9           124         0.458917         0.458917         0.531327             0.
↪531327
10          138         0.448132         0.448132         0.524466             0.
↪524466
...
```

4.1.4 Directory tree

After training, directory tree becomes as follows:

```
workdir
├── data/
│   ├── GaN.xyz
│   ...
├── output/
│   ├── CrystalGa16N16/
│   │   ├── energy.png
│   │   ├── force.png
│   │   └── training.log
│   ├── CrystalGa2N2/
│   │   ├── energy.png
│   │   ├── force.png
│   │   └── training.log
```

(continues on next page)

(continued from previous page)

```
│   ├── master_nnp.npz
│   ├── preprocess/
│   │   └── pca.npz
│   ├── training_config.py
│   └── training_result.yaml
└── training_config.py
```

CHAPTER 5

Modules

5.1 Dataset tools

DatasetGenerator	Deal out datasets as needed.
HDNNPDataset	Combine and preprocess descriptor and property dataset.

5.1.1 DatasetGenerator

class hdnnpy.dataset.dataset_generator.**DatasetGenerator**(**datasets*)

Bases: object

Deal out datasets as needed.

Parameters ***datasets** (HDNNPDataset) – What you want to unite.

all()

Pass all datasets an instance have.

Returns All stored datasets.

Return type list [*HDNNPDataset*]

foreach()

Pass all datasets an instance have one by one.

Returns a stored dataset object.

Return type Iterator [*HDNNPDataset*]

holdout(*ratio*)

Split each dataset at a certain rate and pass it

Parameters **ratio** (*float*) – Specify the rate you want to use as training data. Remains are test data.

Returns All stored dataset split by specified ratio into training and test data.

Return type list [tuple [*HDNNPDataset*, *HDNNPDataset*]]

kfold(*kfold*)
Split each dataset almost equally and pass it for cross validation.

Parameters **kfold** (*int*) – Number of folds to split dataset.

Returns All stored dataset split into training and test data. It iterates k times while changing parts used for test data.

Return type Iterator [list [tuple [*HDNNPDataset*, *HDNNPDataset*]]]

5.1.2 HDNNPDataset

class hdnnpy.dataset.hdnnp_dataset.**HDNNPDataset**(*descriptor*, *property_*, *dataset=None*)
Bases: object

Combine and preprocess descriptor and property dataset.

It is desirable that the type of descriptor and property used for HDNNP is fixed at initialization.
Also, an instance itself does not have any dataset at initialization and you need to execute *construct()*.
If `dataset` is given it will be an instance's own dataset.

Parameters

- **descriptor** (DescriptorDatasetBase) – Descriptor instance you want to use as HDNNP input.
- **property_** (PropertyDatasetBase) – Property instance you want to use as HDNNP label.
- **dataset** (*dict [ndarray], optional*) – If specified, dataset will be initialized with this.

__getitem__(*item*)
Return indexed or sliced dataset as dict data.

__len__()
Redicect to *partial_size*

construct(*all_elements=None*, *preprocesses=None*, *shuffle=True*, *verbose=True*)
Construct an instance's own dataset.

This method does following steps:

- Check compatibility between descriptor and property datasets.
- Expand feature dimension of descriptor dataset according to `all_elements` and pre-process descriptor dataset in a given order and add to its own dataset.
- Add property dataset to its own dataset.
- Clear up the original data in descriptor and property dataset.
- Shuffle the order of the data.

Parameters

- **all_elements** (*list [str], optional*) – If specified, it expands feature dimensions of descriptor dataset according to this.
- **preprocesses** (*list [PreprocessBase], optional*) – If specified, it preprocesses descriptor dataset in a given order.
- **shuffle** (*bool, optional*) – If specified, it shuffles the order of the data.
- **verbose** (*bool, optional*) – Print log to stdout.

Raises AssertionError – If descriptor and property datasets are incompatible.

scatter(*max_buf_len=268435456*)
Scatter dataset by MPI communication.

Each instance is re-initialized with received dataset.

Parameters **max_buf_len** (*int, optional*) – Each data is divided into chunks of this size at maximum.

take(*index*)
Return copied object that has sliced dataset.

Parameters **index** (*int or slice*) – Copied object has dataset indexed or sliced by this.

descriptor
Descriptor dataset instance.

Type *DescriptorDatasetBase*

elemental_composition
Elemental composition of the dataset.

Type list [str]

elements
Elements of the dataset.

Type list [str]

n_input
Number of dimensions of input data.

Type int

n_label
Number of dimensions of label data.

Type int

partial_size
Number of data after scattered by MPI communication.

Type int

property
Property dataset instance.

Type *PropertyDatasetBase*

tag
Unique tag of the dataset.

Usually, it is a form like <any prefix> <chemical formula>. (ex. CrystalGa2N2)

Type str

total_size
Number of data before scattered by MPI communication.

Type int

5.1.3 Descriptor datasets

SymmetryFunctionDataset	Symmetry function dataset for descriptor of HDNNP.

SymmetryFunctionDataset

class hdnnpy.dataset.descriptor.symmetry_function_dataset.**SymmetryFunctionDataset**(*order*, *structures*, ***func_para*

Bases: *hdnnpy.dataset.descriptor.descriptor_dataset_base.DescriptorDatasetBase*

Symmetry function dataset for descriptor of HDNNP.

It accepts 0 or 2 for `order`.

Each symmetry function requires following parameters.
Pass parameters you want to use for the dataset as keyword arguments `func_param_map`.

- type1: R_c
- type2: R_c, η, R_s
- type4: $R_c, \eta, \lambda, \zeta$

Parameters

- **order** (*int*) – passed to super class.
- **structures** (*list [AtomicStructure]*) – passed to super class.
- ****func_param_map** (*list [tuple]*) – parameter sets for each type of symmetry function.

References

Symmetry function was proposed by Behler *et al.* in this paper as a descriptor of HDNNP. Please see here for details of each symmetry function.

__getitem__(*item*)
Return descriptor data this instance has.

If `item` is string, it returns corresponding descriptor. Available keys can be obtained by `descriptors` attribute. Otherwise, it returns a list of descriptor sliced by `item`.

__len__()
Number of atomic structures given at initialization.

calculate_descriptors(*structure*)
Calculate required descriptors for a structure data.

Parameters **structure** (AtomicStructure) – A structure data to calculate descriptors.

Returns Calculated descriptors. The length is the same as `order` given at initialization.

Return type list [ndarray]

clear()
Clear up instance variables to initial state.

differentiate()
Decorator function to differentiate symmetry function.

generate_feature_keys(*elements*)
Generate feature keys from given elements and parameters.

parameters given at initialization are used.
This method is used to initialize instance and expand feature dimension in *HDNNPDataset*.

Parameters **elements** (*list [str]*) – Unique list of elements. It should be sorted alphabetically.

Returns Generated feature keys in a format like `<func_name>:<parameters>:<elements>`.

Return type list [str]

load(*file_path*, *verbose=True*, *remake=False*)
Load dataset from .npz format file.

Only root MPI process load dataset.

It validates following compatibility between loaded dataset and atomic structures given at initialization.

- length of data
- elemental composition
- elements
- tag

It also validates that loaded dataset satisfies requirements.

- feature keys
- order

Parameters

- **file_path** (*Path*) – File path to load dataset.
- **verbose** (*bool, optional*) – Print log to stdout.
- **remake** (*bool, optional*) – If loaded dataset is lacking in any feature key or any descriptor, recalculate dataset from scratch and overwrite it to `file_path`. Otherwise, it raises ValueError.

Raises

- AssertionError – If loaded dataset is incompatible with atomic structures given at initialization.

- `ValueError` – If loaded dataset is lacking in any feature key or any descriptor and `remake=False`.

make (*verbose=True*)
Calculate & retain descriptor dataset

It calculates descriptor dataset by data-parallel using MPI communication.
The calculated dataset is retained in only root MPI process.

Parameters **verbose** (*bool, optional*) – Print log to stdout.

save (*file_path*, *verbose=True*)
Save dataset to .npz format file.

Only root MPI process save dataset.

Parameters

- **file_path** (*Path*) – File path to save dataset.
- **verbose** (*bool, optional*) – Print log to stdout.

Raises `RuntimeError` – If this instance do not have any data.

DESCRIPTORS = ['sym_func', 'derivative', 'second_derivative']
Names of descriptors for each derivative order.

Type list [str]

descriptors
Names of descriptors this instance have.

Type list [str]

elemental_composition
Elemental composition of atomic structures given at initialization.

Type list [str]

elements
Elements of atomic structures given at initialization.

Type list [str]

feature_keys
Unique keys of feature dimension.

Type list [str]

function_names
Names of symmetry functions this instance calculates or has calculated.

Type list [str]

has_data
True if success to load or make dataset, False otherwise.

Type bool

n_feature
Length of feature dimension.

Type int

name = 'symmetry_function'
Name of this descriptor class.

Type str

order
Derivative order of descriptor to calculate.

Type int

params
Mapping from symmetry function name to its parameters.

Type dict [list [tuple]]]

tag
Unique tag of atomic structures given at initialization.

Usually, it is a form like `<any prefix> <chemical formula>`. (ex. `CrystalGa2N2`)

Type str

5.1.4 Property datasets

InteratomicPotentialDataset	Interatomic potential dataset for property of HDNNP.

InteratomicPotentialDataset

class hdnnpy.dataset.property.interatomic_potential_dataset.**InteratomicPotentialDataset**(*ord*
stru
ture

Bases: *hdnnpy.dataset.property.property_dataset_base.PropertyDatasetBase*

Interatomic potential dataset for property of HDNNP.

It accepts 0 or 3 for `order`.

Notes

Currently you cannot use order = 2 or 3, since it is not implemented.

Parameters

- **order** (*int*) – passed to super class.
- **structures** (*list [AtomicStructure]*) – passed to super class.

__getitem__(*item*)
Return property data this instance has.

If `item` is string, it returns corresponding property. Available keys can be obtained by `properties` attribute. Otherwise, it returns a list of property sliced by `item`.

__len__()
Number of atomic structures given at initialization.

calculate_properties(*structure*)
Calculate required properties for a structure data.

Parameters **structure** (AtomicStructure) – A structure data to calculate properties.

Returns Calculated properties. The length is the same as order given at initialization.

Return type list [ndarray]

clear()
Clear up instance variables to initial state.

load(*file_path*, *verbose=True*, *remake=False*)
Load dataset from .npz format file.

Only root MPI process load dataset.

It validates following compatibility between loaded dataset and atomic structures given at initialization.

- length of data
- elemental composition
- elements
- tag

It also validates that loaded dataset satisfies requirements.

- order

Parameters

- **file_path** (*Path*) – File path to load dataset.
- **verbose** (*bool, optional*) – Print log to stdout.
- **remake** (*bool, optional*) – If loaded dataset is lacking in any property, recalculate dataset from scratch and overwrite it to file_path. Otherwise, it raises ValueError.

Raises

- AssertionError – If loaded dataset is incompatible with atomic structures given at initialization.
- ValueError – If loaded dataset is lacking in any property and remake=False.

make(*verbose=True*)
Calculate & retain property dataset

It calculates property dataset by data-parallel using MPI communication.
The calculated dataset is retained in only root MPI process.

Each property values are divided by COEFFICIENTS which is unique to each property dataset class.

Parameters **verbose** (*bool, optional*) – Print log to stdout.

save(*file_path*, *verbose=True*)
Save dataset to .npz format file.

Only root MPI process save dataset.

Parameters

- **file_path** (*Path*) – File path to save dataset.
- **verbose** (*bool, optional*) – Print log to stdout.

Raises `RuntimeError` – If this instance do not have any data.

COEFFICIENTS = [1.0, -1.0, 1.0, 1.0]
Coefficient values of each properties.

Type list [float]

PROPERTIES = ['energy', 'force', 'harmonic', 'third_order']
Names of properties for each derivative order.

Type list [str]

UNITS = ['eV/atom', 'eV/$\\AA$', 'eV/$\\AA$^2', 'eV/$\\AA$^3']
Units of properties for each derivative order.

Type list [str]

coefficients
Coefficient values this instance have.

Type list [float]

elemental_composition
Elemental composition of atomic structures given at initialization.

Type list [str]

elements
Elements of atomic structures given at initialization.

Type list [str]

has_data
True if success to load or make dataset, False otherwise.

Type bool

n_property = 1
Number of dimensions of 0th property.

Type int

name = 'interatomic_potential'
Name of this property class.

Type str

order
Derivative order of property to calculate.

Type int

properties
Names of properties this instance have.

Type list [str]

tag
Unique tag of atomic structures given at initialization.

Usually, it is a form like `<any prefix> <chemical formula>`. (ex. `CrystalGa2N2`)

Type str

units
Units of properties this instance have.

Type list [str]

5.1.5 Dataset base classes

DescriptorDatasetBase	Base class of atomic structure based descriptor dataset.
PropertyDatasetBase	Base class of atomic structure based property dataset.

DescriptorDatasetBase

class hdnnpy.dataset.descriptor.descriptor_dataset_base.**DescriptorDatasetBase**(*order*, *structures*)

Bases: abc.ABC

Base class of atomic structure based descriptor dataset.

Common instance variables for descriptor datasets are initialized.

Parameters

- **order** (*int*) – Derivative order of descriptor to calculate.
- **structures** (*list [AtomicStructure]*) – Descriptors are calculated for these atomic structures.

__getitem__(*item*)
Return descriptor data this instance has.

If `item` is string, it returns corresponding descriptor. Available keys can be obtained by `descriptors` attribute. Otherwise, it returns a list of descriptor sliced by `item`.

__len__()
Number of atomic structures given at initialization.

calculate_descriptors(*structure*)
Calculate required descriptors for a structure data.

This is abstract method. Subclass of this base class have to override.

Parameters **structure** (AtomicStructure) – A structure data to calculate descriptors.

Returns Calculated descriptors. The length is the same as `order` given at initialization.

Return type list [ndarray]

clear()
Clear up instance variables to initial state.

generate_feature_keys(**args*, ***kwargs*)
Generate feature keys of current state.

This is abstract method. Subclass of this base class have to override.

Returns Unique keys of feature dimension.

Return type list [str]

load(*file_path*, *verbose=True*, *remake=False*)
Load dataset from .npz format file.

Only root MPI process load dataset.

It validates following compatibility between loaded dataset and atomic structures given at initialization.

- length of data
- elemental composition
- elements
- tag

It also validates that loaded dataset satisfies requirements.

- feature keys
- order

Parameters

- **file_path** (*Path*) – File path to load dataset.
- **verbose** (*bool, optional*) – Print log to stdout.
- **remake** (*bool, optional*) – If loaded dataset is lacking in any feature key or any descriptor, recalculate dataset from scratch and overwrite it to `file_path`. Otherwise, it raises ValueError.

Raises

- `AssertionError` – If loaded dataset is incompatible with atomic structures given at initialization.
- `ValueError` – If loaded dataset is lacking in any feature key or any descriptor and `remake=False`.

make(*verbose=True*)

Calculate & retain descriptor dataset

It calculates descriptor dataset by data-parallel using MPI communication.
The calculated dataset is retained in only root MPI process.

Parameters **verbose** (*bool, optional*) – Print log to stdout.

save(*file_path*, *verbose=True*)

Save dataset to .npz format file.

Only root MPI process save dataset.

Parameters

- **file_path** (*Path*) – File path to save dataset.
- **verbose** (*bool, optional*) – Print log to stdout.

Raises `RuntimeError` – If this instance do not have any data.

DESCRIPTORS = []

Names of descriptors for each derivative order.

Type list [str]

descriptors

Names of descriptors this instance have.

Type list [str]

elemental_composition
Elemental composition of atomic structures given at initialization.

Type list [str]

elements
Elements of atomic structures given at initialization.

Type list [str]

feature_keys
Unique keys of feature dimension.

Type list [str]

has_data
True if success to load or make dataset, False otherwise.

Type bool

n_feature
Length of feature dimension.

Type int

name = None
Name of this descriptor class.

Type str

order
Derivative order of descriptor to calculate.

Type int

tag
Unique tag of atomic structures given at initialization.

Usually, it is a form like `<any prefix> <chemical formula>`. (ex. `CrystalGa2N2`)

Type str

PropertyDatasetBase

class hdnnpy.dataset.property.property_dataset_base.**PropertyDatasetBase**(*order*, *structures*)

Bases: abc.ABC

Base class of atomic structure based property dataset.

Common instance variables for property datasets are initialized.

Parameters

- **order** (*int*) – Derivative order of property to calculate.
- **structures** (*list [AtomicStructure]*) – Properties are calculated for these atomic structures.

__getitem__(*item*)
Return property data this instance has.

If `item` is string, it returns corresponding property. Available keys can be obtained by `properties` attribute. Otherwise, it returns a list of property sliced by `item`.

__len__()
Number of atomic structures given at initialization.

calculate_properties(*structure*)
Calculate required properties for a structure data.

This is abstract method. Subclass of this base class have to override.

Parameters **structure** (AtomicStructure) – A structure data to calculate properties.

Returns Calculated properties. The length is the same as `order` given at initialization.

Return type list [ndarray]

clear()
Clear up instance variables to initial state.

load(*file_path*, *verbose=True*, *remake=False*)
Load dataset from .npz format file.

Only root MPI process load dataset.

It validates following compatibility between loaded dataset and atomic structures given at initialization.

- length of data
- elemental composition
- elements
- tag

It also validates that loaded dataset satisfies requirements.

- order

Parameters

- **file_path** (*Path*) – File path to load dataset.
- **verbose** (*bool, optional*) – Print log to stdout.
- **remake** (*bool, optional*) – If loaded dataset is lacking in any property, recalculate dataset from scratch and overwrite it to `file_path`. Otherwise, it raises ValueError.

Raises

- AssertionError – If loaded dataset is incompatible with atomic structures given at initialization.
- ValueError – If loaded dataset is lacking in any property and `remake=False`.

make(*verbose=True*)
Calculate & retain property dataset

It calculates property dataset by data-parallel using MPI communication.
The calculated dataset is retained in only root MPI process.

Each property values are divided by `COEFFICIENTS` which is unique to each property dataset class.

Parameters **verbose** (*bool, optional*) – Print log to stdout.

save (*file_path*, *verbose=True*)
Save dataset to .npz format file.

Only root MPI process save dataset.

Parameters

- **file_path** (*Path*) – File path to save dataset.
- **verbose** (*bool, optional*) – Print log to stdout.

Raises RuntimeError – If this instance do not have any data.

COEFFICIENTS = []
Coefficient values of each properties.

Type list [float]

PROPERTIES = []
Names of properties for each derivative order.

Type list [str]

UNITS = []
Units of properties for each derivative order.

Type list [str]

coefficients
Coefficient values this instance have.

Type list [float]

elemental_composition
Elemental composition of atomic structures given at initialization.

Type list [str]

elements
Elements of atomic structures given at initialization.

Type list [str]

has_data
True if success to load or make dataset, False otherwise.

Type bool

n_property = None
Number of dimensions of 0th property.

Type int

name = None
Name of this property class.

Type str

order
Derivative order of property to calculate.

Type int

properties
Names of properties this instance have.

Type list [str]

tag
Unique tag of atomic structures given at initialization.

Usually, it is a form like `<any prefix> <chemical formula>`. (ex. `CrystalGa2N2`)

Type str

units
Units of properties this instance have.

Type list [str]

5.1.6 Atomic structure

AtomicStructure	Wrapper class of ase.Atoms.

AtomicStructure

class hdnnpy.dataset.atomic_structure.**AtomicStructure**(*atoms*)
Bases: object

Wrapper class of ase.Atoms.

It wraps `ase.Atoms` object to define additional methods and attributes.
Before wrapping, it sorts atoms by element alphabetically.
It stores calculated neighbor information such as distance, indices.

Parameters **atoms** (*Atoms*) – an object to wrap.

clear_cache(*cutoff_distance=None*)
Clear up cached neighbor information in this instance.

Parameters **cutoff_distance** (*float, optional*) – It clears the corresponding cached data if specified, otherwise it clears all cached data.

get_neighbor_info(*cutoff_distance*, *geometry_keys*)
Calculate or return cached data.

If there is no cached data, calculate it as necessary.
The calculated result is cached, and retained unless you use *clear_cache()* method.

Parameters

- **cutoff_distance** (*float*) – It calculates the geometry for the neighboring atoms within this value of each atom in a cell.
- **geometry_keys** (*list [str]*) – A list of atomic geometries to calculate between an atom and its neighboring atoms.

Returns Neighbor information required by `geometry_keys` for each atom in a cell.

Return type Iterator [tuple]

classmethod read_xyz (*file_path*)
Read .xyz format file and make a list of instances.

Parses .xyz format file using `ase.io.iread()` and wraps it by this class.

Parameters **file_path** (*Path*) – File path to read atomic structures.

Returns Initialized instances.

Return type list [*AtomicStructure*]

elements
Elements included in a cell.

Type list [str]

5.2 File parsing tools

parse_xyz	Parse a xyz format file and bunch structures by the same tag.

5.2.1 hdnnpy.format.xyz.parse_xyz

hdnnpy.format.xyz.**parse_xyz** (*file_path*, *save=True*, *verbose=True*)
Parse a xyz format file and bunch structures by the same tag.

Parameters

- **file_path** (*Path*) – File path to parse.
- **save** (*bool, optional*) – If True, save the structures bunched by the same tag into files. Otherwise, save into temporarily files.
- **verbose** (*bool, optional*) – Print log to stdout.

Returns

2-element tuple containing:

- tag_xyz_map (dict): Tag to file path mapping.
- **elements (list [str]):** All elements contained in the parsed file.

Return type tuple

5.3 Neural network potential models

HighDimensionalNNP	High dimensional neural network potential.
MasterNNP	Responsible for managing the parameters of each element.
SubNNP	Feed-forward neural network representing one element or atom.

5.3.1 HighDimensionalNNP

class hdnnpy.model.models.**HighDimensionalNNP**(*elemental_composition*, **args*)
Bases: chainer.link.ChainList

High dimensional neural network potential.

This is one implementation of HDNNP that is proposed by Behler *et al* [Ref]. It has a structure in which simple neural networks are arranged in parallel. Each neural network corresponds to one atom and inputs descriptor and outputs property per atom. Total value or property is predicted to sum them up.

Parameters

- **elemental_composition** (*list [str]*) – Create the same number of *SubNNP* instances as this. A *SubNNP* with the same element has the same parameters synchronized.
- ***args** – Positional arguments that is passed to *SubNNP*.

get_by_element(*element*)
Get all *SubNNP* instances that represent the same element.

Parameters **element** (*str*) – Element symbol that you want to get.

Returns All *SubNNP* instances which represent the same `element` in this HDNNP instance.

Return type list [*SubNNP*]

predict(*inputs*, *order*)
Get prediction from input data in a feed-forward way.

It accepts 0 or 2 for `order`.

Notes

0th-order predicted value is not total value, but per-atom value.

Parameters

- **inputs** (*list [ndarray]*) – Length have to equal to `order + 1`. Each element is correspond to `0th-order`, `1st-order`, ...
- **order** (*int*) – Derivative order of prediction by this model.

Returns Predicted values. Each elements is correspond to `0th-order`, `1st-order`, ...

Return type list [Variable]

reduce_grad_to(*master_nnp*)
Collect calculated gradient of parameters into *MasterNNP* for each element.

Parameters **master_nnp** (MasterNNP) – *MasterNNP* instance where you manage parameters.

sync_param_with(*master_nnp*)
Synchronize the parameters with *MasterNNP* for each element.

Parameters **master_nnp** (MasterNNP) – *MasterNNP* instance where you manage parameters.

5.3.2 MasterNNP

class hdnnpy.model.models.**MasterNNP**(*elements*, **args*)
Bases: chainer.link.ChainList

Responsible for managing the parameters of each element.

It is implemented as a simple ChainList of *SubNNP*.

Parameters

- **elements** (*list [str]*) – Element symbols must be unique.
- ***args** – Positional arguments that is passed to *SubNNP*.

dump_params()
Dump its own parameters as str.

Returns Formed parameters.

Return type str

5.3.3 SubNNP

class hdnnpy.model.models.**SubNNP**(*element*, *n_feature*, *hidden_layers*, *n_property*)
Bases: chainer.link.Chain

Feed-forward neural network representing one element or atom.

element is registered as a persistent value.
It consists of repetition of fully connected layer and activation function.
Weight initializer is chainer.initializers.HeNormal.

Parameters

- **element** (*str*) – Element symbol represented by an instance.
- **n_feature** (*int*) – Number of nodes of input layer.
- **hidden_layers** (*list [tuple [int, str]]*) – A neural network structure. Last one is output layer, and the remains are hidden layers. Each element is a tuple (# of nodes, activation function), for example (50, 'sigmoid'). Only activation functions implemented in chainer.functions can be used.
- **n_property** (*int*) – Number of nodes of output layer.

__len__()
Return the number of hidden_layers.

differentiate(*x*, *enable_double_backprop*)
Calculate derivative of the output data w.r.t. input data.

Parameters

- **x** (*Variable*) – Input data which has the shape (n_sample, n_input).
- **enable_double_backprop** (*bool*) – Passed to chainer.grad() to determine whether to create more deep calculation graph or not.

feedforward(*x*)
Propagate input data in a feed-forward way.

Parameters **x** (*Variable*) – Input data which has the shape `(n_sample, n_input)`.

second_differentiate(*x*, *enable_double_backprop*)
Calculate 2nd derivative of the output data w.r.t. input data.

Parameters

- **x** (*Variable*) – Input data which has the shape `(n_sample, n_input)`.
- **enable_double_backprop** (*bool*) – Passed to `chainer.grad()` to determine whether to create more deep calculation graph or not.

5.4 Pre-processing of dataset

PCA	Principal component analysis (PCA).
Scaling	Scale all feature values into the certain range.
Standardization	Scale all feature values to be zero-mean and unit-variance.

5.4.1 PCA

class hdnnpy.preprocess.pca.**PCA**(*n_components=None*)
Bases: *hdnnpy.preprocess.preprocess_base.PreprocessBase*

Principal component analysis (PCA).

The core part of this class uses `sklearn.decomposition.PCA` implementation.

Parameters **n_components** (*int, optional*) – Number of features to keep in decomposition. If `None`, decomposition is not performed.

apply(*dataset*, *elemental_composition*, *verbose=True*)
Apply the same pre-processing for each element to dataset.

It accepts 1 or 2 for length of `dataset`, each element of which is regarded as `0th-order`, `1st-order`, ...

Parameters

- **dataset** (*list [ndarray]*) – Input dataset to be scaled.
- **elemental_composition** (*list [str]*) – Element symbols corresponding to 1st dimension of `dataset`.
- **verbose** (*bool, optional*) – Print log to stdout.

Returns Processed dataset to be zero-mean and unit-variance.

Return type list [ndarray]

dump_params()
Dump its own parameters as `str`.

Returns Formed parameters.

Return type str

load (*file_path*, *verbose=True*)
Load internal parameters for each element.

Only root MPI process loads parameters.

Parameters

- **file_path** (*Path*) – File path to load parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

save (*file_path*, *verbose=True*)
Save internal parameters for each element.

Only root MPI process saves parameters.

Parameters

- **file_path** (*Path*) – File path to save parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

elements
List of elements whose parameters have already been initialized.

Type list [str]

mean
Initialized mean values in each feature dimension and each element.

Type dict [ndarray]

n_components
Number of features to keep in decomposition.

Type int or None

name = 'pca'
Name of this class.

Type str

transform
Initialized transformation matrix in each feature dimension and each element.

Type dict [ndarray]

5.4.2 Scaling

class hdnnpy.preprocess.scaling.**Scaling** (*min_=-1.0*, *max_=1.0*)
Bases: *hdnnpy.preprocess.preprocess_base.PreprocessBase*

Scale all feature values into the certain range.

Parameters

- **min_** (*float*) – Target minimum value of scaling.
- **max_** (*float*) – Target maximum value of scaling.

apply (*dataset*, *elemental_composition*, *verbose=True*)
Apply the same pre-processing for each element to dataset.

It accepts 1 or 2 for length of `dataset`, each element of which is regarded as `0th-order`, `1st-order`, ...

Parameters

- **dataset** (*list [ndarray]*) – Input dataset to be scaled.
- **elemental_composition** (*list [str]*) – Element symbols corresponding to 1st dimension of `dataset`.
- **verbose** (*bool, optional*) – Print log to stdout.

Returns Processed dataset into the same min-max range.

Return type list [ndarray]

dump_params()
Dump its own parameters as `str`.

Returns Formed parameters.

Return type str

load(*file_path*, *verbose=True*)
Load internal parameters for each element.

Only root MPI process loads parameters.

Parameters

- **file_path** (*Path*) – File path to load parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

save(*file_path*, *verbose=True*)
Save internal parameters for each element.

Only root MPI process saves parameters.

Parameters

- **file_path** (*Path*) – File path to save parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

elements
List of elements whose parameters have already been initialized.

Type list [str]

max
Initialized maximum values in each feature dimension and each element.

Type dict [ndarray]

min
Initialized minimum values in each feature dimension and each element.

Type dict [ndarray]

name = 'scaling'
Name of this class.

Type str

target
Target min & max values of scaling.

Type tuple [float, float]

5.4.3 Standardization

class hdnnpy.preprocess.standardization.**Standardization**
Bases: *hdnnpy.preprocess.preprocess_base.PreprocessBase*

Scale all feature values to be zero-mean and unit-variance.

apply(*dataset*, *elemental_composition*, *verbose=True*)
Apply the same pre-processing for each element to dataset.

It accepts 1 or 2 for length of `dataset`, each element of which is regarded as `0th-order`, `1st-order`, ...

Parameters

- **dataset** (*list [ndarray]*) – Input dataset to be scaled.
- **elemental_composition** (*list [str]*) – Element symbols corresponding to 1st dimension of `dataset`.
- **verbose** (*bool, optional*) – Print log to stdout.

Returns Processed dataset to be zero-mean and unit-variance.

Return type list [ndarray]

dump_params()
Dump its own parameters as `str`.

Returns Formed parameters.

Return type str

load(*file_path*, *verbose=True*)
Load internal parameters for each element.

Only root MPI process loads parameters.

Parameters

- **file_path** (*Path*) – File path to load parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

save(*file_path*, *verbose=True*)
Save internal parameters for each element.

Only root MPI process saves parameters.

Parameters

- **file_path** (*Path*) – File path to save parameters.
- **verbose** (*bool, optional*) – Print log to stdout.

elements
List of elements whose parameters have already been initialized.

Type list [str]

mean
Initialized mean values in each feature dimension and each element.

Type dict [ndarray]

name = 'standardization'
Name of this class.

Type str

std
Initialized standard deviation values in each feature dimension and each element.

Type dict [ndarray]

5.4.4 Pre-processing base class

PreprocessBase	Base class of pre-processing.

PreprocessBase

class hdnnpy.preprocess.preprocess_base.**PreprocessBase**
Bases: abc.ABC

Base class of pre-processing.

Initialize private variable `_elements` as a empty `set`.

apply(**args*, ***kwargs*)
Apply the same pre-processing for each element to dataset.

This is abstract method. Subclass of this base class have to override.

dump_params()
Dump its own parameters as `str`.

This is abstract method. Subclass of this base class have to override.

load(**args*, ***kwargs*)
Load internal parameters for each element.

This is abstract method. Subclass of this base class have to override.

save(**args*, ***kwargs*)
Save internal parameters for each element.

This is abstract method. Subclass of this base class have to override.

elements
List of elements whose parameters have already been initialized.

Type list [str]

name = None
Name of this class.

Type str

5.5 Chainer-based training tools

5.5.1 Custom training extensions

ScatterPlot	Trainer extension to output predictions/labels scatter plots.

Continued on next page

Table 10 – continued from previous page

set_log_scale	Change y axis scale as log scale.

ScatterPlot

class hdnnpy.training.extensions.**ScatterPlot**(*dataset*, *model*, *comm*)

Bases: chainer.training.extension.Extension

Trainer extension to output predictions/labels scatter plots.

Parameters

- **dataset** (HDNNPDataset) – Test dataset to plot a scatter plot. It has to have both input dataset and label dataset.
- **model** (HighDimensionalNNP) – HDNNP model to evaluate.
- **comm** (*CommunicatorBase*) – ChainerMN communicator instance.

__call__(*trainer*)

Execute scatter plot extension.

Perform prediction with the parameters of the model when this extension was executed, using the data set at initialization.

Horizontal axis shows the predicted values and vertical axis shows the true values.

Plot configurations are written in `_plot()`.

Parameters **trainer** (*Trainer*) – Trainer object that invokes this extension.

hdnnpy.training.extensions.set_log_scale

hdnnpy.training.extensions.**set_log_scale**(*_*, *a*, *__*)

Change y axis scale as log scale.

5.5.2 Loss functions

Zeroth	Loss function to optimize 0th-order property.
First	Loss function to optimize 0th and 1st-order property.
Potential	Loss function to optimize 0th property as scalar potential.

Zeroth

class hdnnpy.training.loss_function.**Zeroth**(*model*, *properties*, ***_*)

Bases: hdnnpy.training.loss_function.loss_functions_base.LossFunctionBase

Loss function to optimize 0th-order property.

Parameters

- **model** (HighDimensionalNNP) – HDNNP object to optimize parameters.
- **properties** (*list [str]*) – Names of properties to optimize.

eval (***dataset*)
Calculate loss function from given datasets and model.

Parameters ****dataset** (*ndarray*) – Datasets passed as kwargs. Name of each key is in the format 'inputs/N' or 'labels/N'. 'N' is the order of the dataset.

Returns A scalar value calculated with loss function.

Return type Variable

name = 'zeroth'
Name of this loss function class.

Type str

order = {'descriptor': 0, 'property': 0}
Required orders of each dataset to calculate loss function.

Type dict

First

class hdnnpy.training.loss_function.**First**(*model*, *properties*, *mixing_beta*, **_)
Bases: hdnnpy.training.loss_function.loss_functions_base.LossFunctionBase

Loss function to optimize 0th and 1st-order property.

Parameters

- **model** (HighDimensionalNNP) – HDNNP object to optimize parameters.
- **properties** (*list [str]*) – Names of properties to optimize.
- **mixing_beta** (*float*) – Mixing parameter of errors of 0th and 1st order. It accepts 0.0 to 1.0. If 0.0 it optimizes HDNNP by only 0th order property and it is equal to loss function Zeroth. If 1.0 it optimizes HDNNP by only 1st order property.

eval (***dataset*)
Calculate loss function from given datasets and model.

Parameters ****dataset** (*ndarray*) – Datasets passed as kwargs. Name of each key is in the format 'inputs/N' or 'labels/N'. 'N' is the order of the dataset.

Returns A scalar value calculated with loss function.

Return type Variable

name = 'first'
Name of this loss function class.

Type str

order = {'descriptor': 1, 'property': 1}
Required orders of each dataset to calculate loss function.

Type dict

Potential

class hdnnpy.training.loss_function.**Potential**(*model*, *properties*, *mixing_beta*, *summation*, *rotation*, **_)
Bases: hdnnpy.training.loss_function.loss_functions_base.LossFunctionBase

Loss function to optimize 0th property as scalar potential.

Args:

model (HighDimensionalNNP): HDNNP object to optimize parameters.

properties (list [str]): Names of properties to optimize. mixing_beta (float):

Mixing parameter of errors of 0th and 1st order. It accepts 0.0 to 1.0. If 0.0 it optimizes HDNNP by only 0th order property and it is equal to loss function `Zeroth`. If 1.0 it optimizes HDNNP by only 1st order property.

summation (float): Penalty term coefficient parameter for summation of 1st order property. This loss function adds following

penalty to 1st order property vector.

$\sum_{i,lpha} F_{i,lpha} = 0$

rotation (float): Penalty term coefficient parameter for rotation of 1st order property. This loss function adds following

penalty to 1st order property vector.

:math:‘

ot m{F} = 0‘

eval (***dataset*)
Calculate loss function from given datasets and model.

Parameters ****dataset** (*ndarray*) – Datasets passed as kwargs. Name of each key is in the format ‘inputs/N’ or ‘labels/N’. ‘N’ is the order of the dataset.

Returns A scalar value calculated with loss function.

Return type Variable

name = 'potential'
Name of this loss function class.

Type str

order = {'descriptor': 2, 'property': 1}
Required orders of each dataset to calculate loss function.

Type dict

5.5.3 Loss function base class

loss_function.loss_function_base. LossFunctionBase

5.5.4 Training manager

Manager	Context manager to take trainer snapshot and decide whether to train or not.

Manager

class hdnnpy.training.manager.**Manager**(*tag*, *trainer*, *result*, *is_snapshot=True*)
Bases: contextlib.AbstractContextManager

Context manager to take trainer snapshot and decide whether to train or not.

Parameters

- **tag** (*str*) – Tag of dataset used for training.
- **trainer** (*Trainer*) – Trainer object to be managed.
- **result** (*dict*) – Dictionary object containing total elapsed time and metrics value corresponding to the type of loss function. Even when training is stopped / resumed, it is retained.
- **is_snapshot** (*bool, optional*) – Take trainer snapshot if True.

__enter__()
Replace signal handler of SIGINT and SIGTERM.

__exit__(*type_*, *value*, *traceback*)
Restore signal handler of SIGINT and SIGTERM, and record the result of training.

check_to_resume(*resume_tag*)
Decide whether to train or not.

If current tag of dataset is equal to `resume_tag`, restore the state of trainer from snapshot file.

Parameters **resume_tag** (*str*) – Tag of dataset when snapshot was taken last time.

allow_to_run
Whether the given trainer can train with the dataset.

5.5.5 Updater

Updater	Updater for HDNNP training using *HighDimensionalNNP* and *MasterNNP*.

Updater

class hdnnpy.training.updater.**Updater**(**args*, ***kwargs*)
Bases: chainer.training.updaters.standard_updater.StandardUpdater

Updater for HDNNP training using *HighDimensionalNNP* and *MasterNNP*.

update_core()
Calculate gradient of parameters using *HighDimensionalNNP* and collect them in *MasterNNP* and update parameters.

5.6 Utilities

MPI	MPI world communicator and aliases.
pprint	Pretty print function.

5.6.1 MPI

class hdnnpy.utils.**MPI**
Bases: object

MPI world communicator and aliases.

5.6.2 hdnnpy.utils.pprint

hdnnpy.utils.**pprint**(*data=None*, *flush=True*, ***options*)
Pretty print function.

Parameters

- **data** (*str, optional*) – Data to output into stdout.
- **flush** (*bool, optional*) – Flush the stream after output if True.
- ****options** – Other options passed to `print()`.

CHAPTER 6

How to extend HDNNP

6.1 Dataset

HDNNP dataset consists of **Descriptor dataset** and **Property dataset**.

6.1.1 Descriptor dataset

Currently, we have implemented only **symmetry function** dataset.
If you want to use other descriptor dataset, define a class that inherits
`hdnnpy.dataset.descriptor.descriptor_dataset_base.DescriptorDatasetBase`
It defines several instance variables, properties and instance methods for creating a HDNNP dataset.

In addition, override the following abstract method.

- generate_feature_keys

It returns a list of unique keys in feature dimension.

In addition to being able to use it internally, it is also used to expand feature dimension and zero-fill in `hdnnpy.dataset.HDNNPDataset`

- calculate_descriptors

It is main function for calculating descriptors from a atomic structure, which is a wrapper of ase.Atoms object.

6.1.2 Property dataset

Currently, we have implemented only **interatomic potential** dataset.
If you want to use other property dataset, define a class that inherits
`hdnnpy.dataset.property.property_dataset_base.PropertyDatasetBase`
It defines several instance variables, properties and instance methods for creating a HDNNP dataset.

In addition, override the following abstract method.

- calculate_properties

It is main function for getting properties from a atomic structure, which is a wrapper of ase.Atoms object.

6.2 Preprocess

- PCA
- Scaling
- Standardization

6.3 Loss function

Currently, we have implemented following loss function for HDNNP training.

- Zeroth
- First

Each loss function uses a 0th/1st order error of property to optimize HDNNP. `First` uses both 0th/1st order errors of property weighted by parameter `mixing_beta` to optimize HDNNP.

- Potential

It uses 2nd order derivative of descriptor dataset to optimize HDNNP to satisfy following condition:

$$F = 0$$

Then, there is a scalar potential φ:

$$F = \mathrm{grad}\varphi$$

If you want to use other loss function, define a class that inherits
`hdnnpy.training.loss_function.loss_function_base.LossFunctionBase`.
It defines several instance variables, properties and instance methods.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[Ref] https://onlinelibrary.wiley.com/doi/full/10.1002/qua.24890

Python Module Index

h

Index

Symbols

A

C