SIMPLE-NN Documentation

Release 20.2

Kyuhyun Lee
Dongsun Yoo
Wonseok Jeong
Seungwu Han

Jan 27, 2022

Contents

Please download new SIMPLE-NN from here.

This documentation is not updated anymore.

CHAPTER 1

Version history

1.1 v21.1

- Separate function: generate / preprocess works only if each tag turns on in input.yaml.
- Bug fix: Inconsistent position from ASE.
- Memory issue fix: pair_nn.cpp for more atom type without memory leak.

1.2 v20.2

- Replica ensemble: you can estimate the standard deviation of atomic energy to detect the atom out of training set.
- Bug fix: LAMMPS atom type matching error, memory allocation

1.3 v20.1

- `use_stress`: (default false) if you set `use_stress` true in `neural_network` of your `input.yaml` , you can train your neural network with virial stress tensor
- LAMMPS Optimization: Using optimization in for loop and applying 'memoization', it becomes around 2 times faster!

CHAPTER 2

Install

2.1 Install SIMPLE-NN

SIMPLE-NN is tested and supported on the following versions of Python:

- Python `2.7`, `3.4-3.6`

2.1.1 Requirements

In SIMPLE-NN, various Python modules are used. Most of these modules are installed automatically during the install process of SIMPLE-NN. However, we recommend to install Tensorflow manually for better performance. In addition, you need to install mpi4py(optional) manually if you want to use MPI. (MPI is supported only in generate_features and preprocess part. See /simple_nn/Simple_nn section.) Detailed information for installing Tensorflow and mpi4py can be found from the links below.

Install Tensorflow: https://www.tensorflow.org/install/

Tensorflow `r1.6-r1.13` is supported.

Install mpi4py: https://mpi4py.readthedocs.io/en/stable/install.html

2.1.2 Install from source

You can download a current SIMPLE-NN source package from link below. Once you have a zip file, unzip it. This will create SIMPLE-NN directory. After unzipping the file, run the command below to install SIMPLE-NN.

Download SIMPLE-NN: https://github.com/MDIL-SNU/SIMPLE-NN

```
cd SIMPLE-NN
python setup.py install
# If root permission is not available, add --user command like below
# python setup.py install --user
```

Currently, pip install is not supported but will be addressed.

2.2 Install LAMMPS implementation

To utilize LAMMPS package for SIMPLE-NN generated NN potentials, you must copy the source codes to LAMMPS/src directory with the following command and compile LAMMPS package.

```
cp SIMPLE-NN/simple_nn/features/symmetry_function/symmetry_functions.h /path/to/
↪lammps/src/
cp SIMPLE-NN/simple_nn/features/symmetry_function/pair_nn.* /path/to/lammps/src/
```

CHAPTER 3

Tutorials

3.1 Preparing dataset

SIMPLE-NN uses ASE to handle output from *ab initio* programs like VASP or Quantum ESPRESSO. All output types supported by ASE can be used in SIMPLE-NN, but they need to contain essential information such as atom coordinates, lattice parameters, energy, and forces. You can check if the output file contains the appropriate information by using the following command:

```
from ase import io

atoms = io.read('some_output')
# atom coordinates
atoms.get_positions()
# chemical symbols
atoms.get_chemical_symbols()
# lattice parameters
atoms.get_cell()
# structure_energy
atoms.get_potential_energy()
# atomic force
atoms.get_forces()
# structure_stress
atoms.get_stress() (need unit conversion!)
```

3.2 Preparing inputs files

SIMPLE-NN use YAML style input file: `input.yaml`. `input.yaml` consists of three part: basic parameters, feature-related parameters, and model-related parameters. The basic format of `input.yaml` is like below:

```
# Basic parameters
generate_features: true
```

(continues on next page)

(continued from previous page)

```
preprocess: true
train_model: true
atom_types:
  - Si
  - O

# feature-related parameters
symmetry_function: # class name of the feature
params:
  Si: params_Si
  O: params_O

# model-related parameters
neural_network: # class name of the model
  method: Adam
  nodes: 30-30
  batch_size: 10
  total_iteration: 50000
  learning_rate: 0.001
```

Depending on the feature and model classes, additional input files may be required. For example, for `symmetry_function` class, additional files named `str_list` and `params_XX` (name of `params_XX` can be changed) are required. Details of parameters and additional files are listed in /simple_nn/Simple_nn (basic parameters), *Features* (feature-related parameters) and *Models* (model-related parameters) section.

3.3 Running the code

To run SIMPLE-NN, you simply have to run the predefined script `run.py` after preparing all input files. The basic format of `run.py` is described below:

```
# run.py
#
# Usage:
# $ python run.py

from simple_nn import Simple_nn
from simple_nn.features.symmetry_function import Symmetry_function
from simple_nn.models.neural_network import Neural_network

model = Simple_nn('input.yaml',
                  descriptor=Symmetry_function(),
                  model=Neural_network())
model.run()
```

Examples of actual use for the entire process of generating neural network potential can be found in *Examples* section or `SIMPLE-NN/examples/`

3.4 Outputs

The default output file of SIMPLE-NN is `LOG`, which contains the execution log of SIMPLE-NN. In addition to `LOG`, an additional output file is created for each process of SIMPLE-NN. After `Symmetry_function.generate` method, you can find the output files listed below:

- `data/data##.pickle`: (## indicates number) Data file which contains descriptor vectors, a derivative of descriptor vectors, and other parameters per structure.

After `Symmetry_function.preprocess` method, you can find the output files listed below:

- `data/{training,valid}_data_####_to_####.tfrecord`: Packed Training/validation dataset which contains the information in `data/data##.pickle` and additional parameters like atomic weights which are used during training process.
- `pickle_{training,valid}_list`: List of pickle files that are included in `data/{training,valid}_data_####_to_####.tfrecord` file.
- `{train,valid}_list`: List of tfrecord files (used in network optimization process)
- `scale_factor`: Scale factor for symmetry function.
- `atomic_weights`: Data file that contains atomic weights.

After `Neural_network.train` method, you can find the output files listed below:

- `SAVER_iterationXXXX.*`, `checkpoint`: Tensorflow save file which contains the network information at the XXXXth iteration.
- `potential_saved_iterationXXXX`: LAMMPS potential file which contains the network information at the XXXXth iteration.

3.5 MD simulation with LAMMPS

To run MD simulation with LAMMPS, add the lines into the LAMMPS script file.

```
pair_style nn
pair_coeff * * /path/to/potential_saved Si O
```

Regarding the unit system, the NNP trained with VASP output is compatible with the LAMMPS units 'metal'. For outputs from other ab initio programs, however, the appropriate unit should be chosen with the user's discretion.

CHAPTER 4

Examples

4.1 Introduction

This section demonstrate SIMPLE-NN with examples. Example files are in `SIMPLE-NN/examples/`. In this example, snapshots from 500K MD trajectory of amorphous SiO_2 (60 atoms) are used as training set.

Note: Since we set the relative path for reference file in `str_list`, You need to move to the directory indicated in each section below to run the examples.

4.2 Generate NNP

To generate NNP using symmetry function and neural network, you need three types of input file (input.yaml, str_list, params_XX) as described in *Tutorials* section. The example files except params_Si and params_O are introduced below. Detail of params_Si and params_O can be found in *Symmetry function* section. Input files introduced in this section can be found in `SIMPLE-NN/examples/SiO2/generate_NNP`.

```
# input.yaml
generate_features: true
preprocess: true
train_model: true
atom_types:
  - Si
  - O

symmetry_function:
  params:
    Si: params_Si
    O: params_O

neural_network:
```

(continues on next page)

(continued from previous page)

```
  method: Adam
  nodes: 30-30
  batch_size: 10
  total_iteration: 50000
  learning_rate: 0.001
```

```
# str_list
../ab_initio_output/OUTCAR_comp ::10
```

With this input file, SIMPLE-NN calculate feature vectors and its derivatives (`generate_features`), generate training/validation dataset (`preprocess`) and optimize the network (`train_model`). Sample VASP OUTCAR file (the file is compressed to reduce the file size) is in `SIMPLE-NN/examples/SiO2/ab_initio_output`. In MD trajectory, snapshots are sampled in the interval of 10 MD steps. In this example, 70 symmetry functions consist of 8 radial symmetry functions per 2-body combination and 18 angular symmetry functions per 3-body combination. Thus, this model uses 70-30-30-1 network for both Si and O. The network is optimized by Adam optimizer with the 0.001 of learning rate and batch size is 10.

Output files can be found in `SIMPLE-NN/examples/SiO2/generate_NNP/outputs`. In the folder, generated dataset is stored in `data` folder and execution log and energy/force RMSE are stored in `LOG`.

4.3 Potential test

4.3.1 Generate test dataset

Generating a test dataset is same as generating a training/validation dataset. In this example, we use same VASP OUTCAR to generate test dataset. Input files introduced in this section can be found in `SIMPLE-NN/examples/SiO2/generate_test_data`.

```
# input.yaml
generate_features: true
preprocess: true
train_model: false
atom_types:
  - Si
  - O

symmetry_function:
  params:
    Si: params_Si
    O: params_O
  valid_rate: 0.
```

In this case, `train_model` is set to `false` because training process is not required to generate test dataset. In addition, valid_rate also set to 0. `str_list` is same as *Generate NNP* section.

Note: To prevent overwriting of the existing training/validation dataset, create a new folder and create a test dataset.

4.3.2 Error check

To check the error for test dataset, use the setting below. And for running test mode, you need to copy the `train_list` file generated in *Generate test dataset* section to `SIMPLE-NN/examples/SiO2/error_check`

and change filename to `test_list`. Edit the path to data directory in `test_list` file accordingly. For example, it should be changed from `./data/training_data_0000_to_0006.tfrecord` to `../generate_test_data/data/training_data_0000_to_0006.tfrecord` in this example. Also, copy `scale_factor` and `params_*` to the current directory. These files contain information on data set, so you have to carry them with the data set. Input files introduced in this section can be found in `SIMPLE-NN/examples/SiO2/error_check`.

```
# input.yaml
generate_features: false
preprocess: false
train_model: true
atom_types:
  - Si
  - O

symmetry_function:
  params:
    Si: params_Si
    O: params_O

neural_network:
  method: Adam
    nodes: 30-30
  batch_size: 10
  train: false
  test: true
  continue: true
```

Note: You need to change the filename from `SAVER_iterationXXXX.*` to `SAVER.*` to use the option `continue: true` and modify the checkpoints file (remove '_iterationXXXX' in the text). If you use the option `continue: weights`, change the filename from `potential_saved_iterationXXXX` to `potential_saved`.

After running SIMPLE-NN with the setting above, new output file named `test_result` is generated. The file is pickle format and you can open this file with python code of below:

```
from six.moves import cPickle as pickle

with open('test_result') as fil:
    res = pickle.load(fil) # For Python 2

with open('test_result', 'rb') as fil:
    res = pickle.load(fil, encoding='latin1') # For Python 3
```

In the file, DFT energies/forces, NNP energies/forces are included.

4.4 Molecular dynamics

Please check in *Tutorials* section for detailed LAMMPS script writing.

4.5 Principal component analysis

SIMPLE-NN provides principal component analysis (PCA) as a method for preprocessing input descriptor vector. Input descriptor vector, including Behler-type symmetry functions, often has high correlation between components. In that case, decorrelating input descriptor vector using PCA before feeding it to a machine-learning model can give much faster convergence.

In order to use PCA, add following lines in `input.yaml` when you do preprocess and when you do training and testing. For detailed descriptions of input parameters, see *here*.

```
neural_network:
   pca: true
   pca_whiten: true
   pca_min_whiten_level: 1.0e-8
```

A pickle file named `pca` will be generated during the preprocessing. You need to copy `pca` file to where you run SIMPLE-NN with trained model, just like `scale_factor` file.

4.6 Parameter tuning

4.6.1 GDF

GDF[1] is used to reduce the force errors of the sparsely sampled atoms. To use GDF, you need to calculate the $\rho(\mathbf{G})$ by adding the following lines to the `symmetry_function` section in `input.yaml`. SIMPLE-NN supports automatic parameter generation scheme for σ and c. Use the setting `sigma: Auto` to get a robust σ and c (values are stored in LOG file). Input files introduced in this section can be found in `SIMPLE-NN/examples/SiO2/parameter_tuning_GDF`.

```
#symmetry_function:
  #continue: true # if individual pickle file is not deleted
  atomic_weights:
    type: gdf
    params:
      sigma: Auto
      # for manual setting
      #  Si: 0.02
      #  O: 0.02
```

$\rho(\mathbf{G})$ indicates the density of each training point. After calculating $\rho(\mathbf{G})$, histograms of $\rho(\mathbf{G})^{-1}$ are also saved as in the file of `GDFinv_hist_XX.pdf`.

Note: If there is a peak in high $\rho(\mathbf{G})^{-1}$ region in the histogram, increasing the Gaussian weight(σ) is recommended until the peak is removed. On the contrary, if multiple peaks are shown in low $\rho(\mathbf{G})^{-1}$ region in the histogram, reduce σ is recommended until the peaks are combined.

In the default setting, the group of $\rho(\mathbf{G})^{-1}$ is scaled to have average value of 1. The interval-averaged force error with respect to the $\rho(\mathbf{G})^{-1}$ can be visualized with the following script.

```
from simple_nn.utils import graph as grp

grp.plot_error_vs_gdfinv(['Si','O'], 'test_result')
```

[1] W. Jeong, K. Lee, D. Yoo, D. Lee and S. Han, J. Phys. Chem. C 122 (2018) 22790

where `test_result` is generated after *Error check* as the output file. The graph of interval-averaged force errors with respect to the $\rho(\mathbf{G})^{-1}$ is generated as `ferror_vs_GDFinv_XX.pdf`

If default GDF is not sufficient to reduce the force error of sparsely sampled training points, One can use scale function to increase the effect of GDF. In scale function, b controls the decaying rate for low $\rho(\mathbf{G})^{-1}$ and c separates highly concentrated and sparsely sampled training points. To use the scale function, add following lines to the `symmetry_function` section in `input.yaml`.

```
#symmetry_function:
  weight_modifier:
    type: modified sigmoid
    params:
      Si:
        b: 0.02
        c: 3500.
      O:
        b: 0.02
        c: 10000.
```

For our experience, $b = 1.0$ and automatically selected c shows reasonable results. To check the effect of scale function, use the following script for visualizing the force error distribution according to $\rho(\mathbf{G})^{-1}$. In the script below, `test_result_noscale` is the test result file from the training without scale function and `test_result_wscale` is the test result file from the training with scale function.

```
from simple_nn.utils import graph as grp

grp.plot_error_vs_gdfinv(['Si','O'], 'test_result_noscale', 'test_result_wscale')
```

4.7 Uncertainty Estimation

Replica ensemble[2] is used to estimate the atomic-resolution uncertainty. Please read above paper for details. We recommend you to make independent directories for each step

---

Note: Before following steps, you have prepared `*.pickle` in `path/data/`. If not, please run with below options first.

```
#input.yaml
generate_feature: true
preprocess: false
train_model: false

symmetry_function:
  remain_pickle: true (default: false)
```

4.7.1 Step 1. Extract the atomic energy

Extract the atomic energy that will be used for reference of replicas. Make `test_list` as described in *Potential test* and prepare the `potential_saved`

[2] W. Jeong, D. Yoo, K. Lee, J. Jung and S. Han, J. Phys. Chem. Lett. 2020, 11, 6090-6096

```
#input.yaml
generate_feature: false
preprocess: false
train_model: true

neural_network:
  NNP_to_pickle: true
  test: false
  train: false
  continue: true (or weights)
```

4.7.2 Step 2. Write the data into tfrecord

Convert `*.pickles` into `tfrecord` to feed input data during training

```
#input.yaml
generate_feature: false
preprocess: true
train_model: false

symmetry_function:
  add_NNP_ref: true
  continue: true
```

4.7.3 Step 3. Train with atomic energy

Train model with atomic energy only to speed up (`use_force` and `use_stress` are `false`). Choose a suitable the number of nodes and standard deviation of initial weight. Repeat this step several times by changing the number of nodes.

```
#input.yaml
generate_feature: false
preprocess: false
train_model: true

neural_network:
  NNP_to_pickle: false
  use_force: false
  use_stress: false
  nodes: (user's choice)
  test: false
  train: true
  continue: false
  E_loss: 3
  weight_initializer:
    params:
      stddev: (user's choice)

symmetry_function:
  add_NNP_ref: true
  continue: true
```

4.7.4 Step 4. Molecular dynamics

Note: Before this step, you have to compile your LAMMPS with `pair_nn_replica.cpp` and `pair_nn_replica.h`.

LAMMPS can calculate the atomic uncertainty through standard deviation of atomic energies. Because our NNP do not deal with charged system, atomic uncertainty can be written as atomic charge. Prepare your data file as charge format and please modify your LAMMPS input as below example.

```
atom_style  charge
pair_style  nn/r (# of replica potentials)
pair_coeff  * * (reference potential) (element1) (element2) ... &
            (replica potential_#1) &
            (replica_potential_#2) &
            ...
compute     (ID) (group-ID) property/atom q
```

CHAPTER 5

Features

5.1 Introduction

In this section, you can find the information of descriptor vectors that are used in SIMPLE-NN.

5.2 Feature list

5.2.1 Symmetry function

5.2.1.1 Introduction

SIMPLE-NN uses atom-centered symmetry function[1] as a default descriptor vector. Radial symmetry function G2 and angular symmetry functions G4 and G5 are used.

5.2.1.2 Parameters

feature vector related parameter

- `params`: Defines the name of a text file which which contains parameters of symmetry function for each atom types:

```
params:
  Si: params_Si
  O: params_O
```

- `refdata_format`: (string, default: 'vasp_out') Define the file format to read. In SIMPLE-NN, ASE is used to read the reference file. The file format that is supported in ASE is listed in ASE documents.

[1] J. Behler, J. Chem. Phys. 134 (2011) 074106

- `compress_outcar`: (boolean, default: true) If `true`, VASP OUTCAR file is automatically compressed before handling it. This flag does not change the original file. The tag is only activated when `refdata_format: true`.

Atomic weight related parameter

- `atomic_weights`: (dictionary) Dictionary for atomic weights. To use GDF, set this parameter as below:

```
atomic_weights:
  type: gdf
  params:
    sigma:
      Si: 0.02
```

- `weight_modifier`: (dictionary) Dictionary for weight modifier. Detailed setting is like below:

```
weight_modifier:
  type: modified sigmoid
  params:
    Si:
      b: 0.02
      c: 1000.
```

Detailed information about GDF usage can be found in this paper[2],

preprocessing related parameter

- `valid_rate`: (float, default: 0.1) The ratio of validation set relative to entire dataset.
- `remain_pickle`: (boolean, default: false) If `true`, pickle files containing symmetry functions and its derivatives are not removed after generating tfrecord files. Currently, we do not support any methods to read tfrecord file externally. Thus, set this parameter `true` if you want to perfrom further analysis with the symmetry function values.
- `shuffle`: (boolean, default: true) If `false`, the order of pickle files to be written in tfrecord files is identical to the order of structure list in `str_list`. Thus, set this parameter `false` if you want to use former structures as training data and latter ones as validation data

tfrecord related parameter

- `data_per_tfrecord`: (int, default: 100) The number of structures that is packed into one tfrecord file.
- `num_parallel_calls`: (int, default: 5) The number elements processed in parallel. We recommend the value that is same as the number of cores in your CPU.

5.2.1.3 Inputs

To use symmetry function as input vector, you need additional input file 'params_XX' and 'str_list'

params_XX contains the coefficients for symmetry functions. XX is an atom type which included in the target system. The detailed format of 'param_XX' is described in below:

[2] W. Jeong, K. Lee, D. Yoo, D. Lee and S. Han, J. Phys. Chem. C 122 (2018) 22790

```
2 1 0 6.0 0.003214 0.0 0.0
2 1 0 6.0 0.035711 0.0 0.0
4 1 1 6.0 0.000357 1.0 -1.0
4 1 1 6.0 0.028569 1.0 -1.0
4 1 1 6.0 0.089277 1.0 -1.0
```

Each parameter indicates (SF means symmetry function)

```
[type of SF(1)] [atom type index(2)] [cutoff distance(1)] [coefficients for SF(3)]
```

The number inside the indicates the number of parameters.

First column indicates the type of symmetry function. Currently, G2 (2), G4 (4), and G5 (5) are available.

Second and third column indicates the type index of neighbor atoms which starts from 1. (The order of type index need to be the same as the order of the `atom_types` tag indicated in `input.yaml`) For radial symmetry function, only one neighbor atom needed to calculate the symmetry function value, thus third parameter is set to zero. For angular symmetry function, two neighbor atoms are needed. The order of second and third column do not affect the calculation result.

The fourth column means the cutoff radius for cutoff function.

The remaining columns are the parameters applied to each symmetry function. For radial symmetry function, the fifth and sixth column indicates η and $\mathrm{R_s}$. The value in last column is dummy value. For angular symmetry function, η, ζ, and λ are listed in order.

str_list contains the location of reference calculation data. The format is described below:

```
[ structure_type_1 ]
/location/of/calculation/data/oneshot_output_file :
/location/of/calculation/data/MDtrajectory_output_file 100:2000:20

[ structure_type_2 : 3.0 ]
/location/of/calculation/data/same_folder_format{1..10}/oneshot_output_file :
```

You can use the format of braceexpand to set a path to reference file (like last line). The part which is written after the path indicates the index of snapshots. (format is 'start:end:interval'. ':' means all snapshots.) You can group structures like above for convenience ([structure_group_name] above the pathes of reference file). If `print_structure_rmse` is true, RMSEs for each structure type are also prited in LOG file (see *High-dimensional neural network* section) In addition, you can set the weights for each structure type ([structure_group_name : weights], default: 1.0).

5.2.1.4 Methods

__init__(*self*, *inputs*, *descriptor=None*, *model=None*)

Args:

- `inputs`: (str) Name of the input file.
- `descriptor`: (object) Object of the feature class
- `model`: (object) Object of the model class

Initiator of Simple-nn class. It takes feature and model object and set the default parameters of SIMPLE-NN.

generate(*self*)

Method for generating symmetry functions and its derivatives.

preprocess(*self*, *calc_scale=True*, *use_force=False*, *get_atomic_weights=None*, ***kwargs*)

Args:

- `calc_scale`: (boolean)
- `use_force`: (boolean)
- `get_atomic_weights`: (object) Object of model class

Method for preprocessing the training data. Process like calculating scaling factor and calculating atomic weights are contained in this method.

References

CHAPTER 6

Models

6.1 Introduction

This section contains the information of ML models used in SIMPLE-NN.

6.2 Model list

6.2.1 High-dimensional neural network

6.2.1.1 Introduction

SIMPLE-NN use High-Dimensional Neural Network(HDNN)[1] as a default machine learning model.

6.2.1.2 Parameters

Function related parameter

- `train`: (boolean, default: true) If `true`, training process proceeds.
- `test`: (boolean, default: false) If `true`, predicted energy and forces for test set are calculated.
- `continue`: (boolean or string, default: false) If `true`, training process restarts from save file (SAVER.*, checkpoints). If `continue: weights`, training process restarts from the LAMMPS potential file (potential_saved).

Note: `potential_saved` only contains the weights and bias of the network. Thus, hyperparameter used in Adam optimizer is reset with `continue: weights`.

[1] J. Behler, M. Parrinello, Phys. Rev. Lett. 98 (2007) 146401

Network related parameter

- `nodes`: (str or dictionary, default: 30-30) String value to indicate the network architecture. 30-30 means 2 hidden layers and each hidden layer has 30 hidden nodes. You can use different network structure for different atom types. For example:

```
nodes:
  Si: 30-30
  O: 15-15
```

- `regularization`: (dictionary) Regularization setting. Currently, L1 and L2 regularization is available.

```
regularization:
  type: l2 # l1 and l2 is available
  params:
    coeff: 1e-6
```

- `use_force`: (boolean, default: false) If `true`, both energy and force are used for training.
- `force_coeff`: (float, default: 0.1) Ratio of force loss to energy loss in total loss
- `use_stress`: (boolean, default: false) If `true`, both energy and stress are used for training. (Caution : The unit of stress RMSE written in LOG file is kbar.)
- `stress_coeff`: (float, default: 0.00001) Ratio of stress loss to energy loss in total loss
- `stddev`: (float, default: 0.3) Standard deviation for weights initialization.

PCA-related parameters

- `pca`: When set to true, PCA is applied to the input descriptor vector.

  Default:

```
pca: false
```

- `pca_whiten`: When set to true, PCA-transformed vector is whitened (the variances of principal components are normalized to unity). The whitening is different than original scheme. See *below* for more details on the difference.

  Default:

```
pca_whiten: true
```

- `pca_min_whiten_level`: This option can be used to suppress whitening of principal components with small variances. Originally, whitening normalize variances of all principal components to unity:

$$PC_{i,\text{whiten}} = \frac{PC_i}{\sqrt{\text{Var}_i}}$$

  A small constant a is added to the variance to suppress principal components with small variances:

$$PC_{i,\text{whiten}} = \frac{PC_i}{\sqrt{\text{Var}_i + a}}$$

  In practice, this can be used to reduce the generalization error.

  Default:

```
pca_min_whiten_level: 1.0e-8
```

Optimization related parameter

- `method`: (str, default: Adam) Optimization method. You can choose Adam or L-BFGS.
- `batch_size`: (int, default: 64) The number of samples in the batch training set.
- `full_batch`: (boolean, default: true) If `true`, full batch mode is enabled.

Note: In the full_batch mode, `batch_size` behaves differently. In full_batch mode, the entire dataset must be considered in one iteration, but this often causes out of memory problems. Therefore, in SIMPLE-NN, a batch dataset with the size of `batch_size` is processed at once, and this process is repeated to perform operations on the entire data set.

- `total_iteration`: (int, default: 10000) The number of total training iteration. If negative, early termination scheme is activated (See `break_max` below).
- `learning_rate`: (float, default: 0.0001, *Exponential decay*) Learning rate for gradient descendent based optimization algorithm.
- `force_coeff` and `energy_coeff`: (float, default: 0.1 and 1., *Exponential decay*) Scaling coefficient for force and energy loss.
- `loss_scale`: (float, default: 1.) Scaling coefficient for the entire loss function.
- `optimizer`: (dictionary) additional parameters for user-defined optimizer

Logging & saving related parameters

- `show_interval`: (int, default: 100) Interval for printing RMSE in LOG file.
- `save_interval`: (int, default: 1000) Interval for saving the neural network potential file.
- `save_criteria`: (list, default: []) Criteria for saving the neural network potential file. Energy error for validation set (`v_E`), force error for validation set (`v_F`), and force error for validation set for sparsely sampled training points (`v_F_XX_sparse`) are possible. A network is saved only when all values in the criteria are smaller than previous save points. If not, `break_count` is increased (See `break_max` below).

Note: In SIMPLE-NN, save conditions(`save_interval` and `save_critera`) are checked every multiple of `show_interval`. Thus, it is recommended to set `save_interval` to multiples of `show_interval`.

Note: Every multiple of `show_interval`, SIMPLE-NN calculates energies and forces for entire validation set. so the process takes a lot of time in general. Thus, small `show_interval` may slow down the training speed.

- `break_max`: (int, default: 10) If save criteria is not satisfied in current save points, `break_count` increases. Optimization process is terminated when `break_count` >= `break_max`. This tag is only activated when total_iteration is negative.
- `print_structure_rmse`: (boolean, default: false) If `true`, RMSEs for each structure type are also printed in LOG file.

Performance related parameters

- `inter_op_parallelism_threads` and `intra_op_parallelism_threads`: (int, default: 0, 0) The number of threads for CPU. Default is 0, which results the values set to the number of logical cores. The recommended values are the number of physical cores for intra_op_parallelism_threads and the number of sockets for inter_op_parallelism_threads. intra_op_parallelism_threads should be equal to OMP_NUM_THREADS.
- `cache`: (boolean, default: false) If `true`, batch dataset is temporarily saved using caches. Calculation speed may increase but larger memory is needed.

Exponential decay

Some parameters in neural_network may need to decrease exponentially during the optimization process. In those cases, you can use this format instead of float value. More information can be found in Tensorflow homepage

```
parameter_name:
    learning_rate: 1.
    decay_rate: 0.95
    decay_steps: 10000
    staircase: false
```

Note: If `continue: true`, `global_step` (see the link above) of save points is also loaded. Thus, you need to consider the `global_step` to calculate the values from `exponential_decay`. On the contrary, `global_step` is reset when `continue: weights`

6.2.1.3 Methods

__init__(*self*)
Initiator of Neural_network class.

train(*self*, *user_optimizer=None*, *aw_modifier=None*)

Args:

- `user_optimizer`: User defined optimizer. Can be set in the script run.py
- `aw_modifier`: scale function for atomic weights.

Method for optimizing neural network potential.

References

CHAPTER 7

simple_nn package

7.1 Subpackages

7.1.1 simple_nn.features package

7.1.1.1 Subpackages

simple_nn.features.symmetry_function package

Module contents

7.1.1.2 Module contents

7.1.2 simple_nn.models package

7.1.2.1 Submodules

simple_nn.models.neural_network module

7.1.2.2 Module contents

7.1.3 simple_nn.utils package

7.1.3.1 Module contents

7.2 Module contents

Index

Symbols

G

P

T