SIMPLE-NN Documentation

Release 0.7.0

Kyuhyun Lee
Dongsun Yoo
Wonseok Jeong
Seungwu Han

Jan 12, 2019

CONTENTS:

INSTALL

1.1 Install SIMPLE-NN

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

- Python 2.7, 3.4-3.6

1.1.1 Requirements

In SIMPLE-NN, various Python modules are used. Most of these modules are installed automatically during the install process of SIMPLE-NN. However, you need to install Tensorflow and mpi4py(optional) manually before installing SIMPLE-NN. Detailed method to install Tensorflow and mpi4py is listed below.

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

We support Tensorflow >= r1.6.

Install mpi4py:

```
pip install mpi4py
```

1.1.2 Install from source

You can download a current SIMPLE-NN source package fromlink 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.

```
cd SIMPLE-NN
python setup.py
```

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

1.2 Install LAMMPS implements

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

```
cp SIMPLE-NN/simple_nn/features/symmetry_function/{symmetry_function.*,pair_nn.*} /
↪path/to/lammps/src/
```

CHAPTER TWO

TUTORIALS

2.1 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()
```

2.2 Preparing inputs files

SIMPLE-NN use YAML style input file: input.yaml. input.yaml consists of 3 part: basic parameters, feature-related parameters, and model-related parameters. The basic format of input.yaml is like below:

```
# Basic parameters
generate_features: true
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_epoch: 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 are required. Details of parameters and additional files are listed in *Simple_nn* (basic parameters), *Features* (feature-related parameters) and *Models* (model-related parameters) page.

2.3 Run 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* or SIMPLE-NN/examples/

2.4 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 same information of data/data##.pickle.
- pickle_{training,valid}_list: List of pickle files that includes 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 contains atomic weights.

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

- SAVER.*, checkpoint: Tensorflow save file which contains the network information.
- potential_saved: LAMMPS potential file.

2.5 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.

EXAMPLES

3.1 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: To actually run the example, you need to modify str_list to suit your environment.

3.2 Generate NNP

To generate NNP using symmetry function and neural network, you need 3 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 *Features*. 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:
  method: Adam
  nodes: 30-30
  batch_size: 10
  total_epoch: 50000
  learning_rate: 0.001
```

```
# str_list
SIMPLE-NN/examples/SiO2/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 is in SIMPLE-NN/examples/SiO2/ab_initio_output. In MD trajectory, snapshots are selected with 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 use 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.

3.3 Potential test

3.3.1 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 does not need 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.

3.3.2 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 this folder and change filename to test_list. 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 # continue: weights (if the potential is generated using L-BFGS)
```

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)
```

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

3.3.3 Molecular dynamics

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

3.4 Parameter tuning

3.4.1 GDF

GDF[1] is used to reduce the force error for 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. 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:
        Si: 0.02
        O: 0.02
```

$\rho(\mathbf{G})$ indicates the density of each training points. After calculating $\rho(\mathbf{G})$, histograms of $\rho(\mathbf{G})^{-1}$ also generated in the file of GDFinv_hist_XX.pdf. If the histogram is biased in low $\rho(\mathbf{G})^{-1}$ region or large $\rho(\mathbf{G})^{-1}$ region, modify Gaussian width(σ). 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')
```

where test_result is generated after *Error check* as the output file. The graph of interval-averaged force error 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 atoms, One can use scale function to increase the effect of GDF. In scale function, b icontrols the decaying rate for low $\rho(\mathbf{G})^{-1}$ and c separates highly concentrated and sparsely sampled atoms. To use the scale function, add following lines to the symmetry_function section in input.yaml.

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

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

To check the effect of scale function, use the following script for visualizing the force error distribution according to $\rho(\mathbf{G})^{-1}$.

```
from simple_nn.utils import graph as grp

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

CHAPTER

FOUR

SIMPLE_NN

4.1 Introduction

Simple_nn class is the main wrapper class to use SIMPLE-NN.

4.2 Parameters

- generate_features: (boolean, default=true) If true, feature generation process is performed. In addition, preprocess is set true automatically.
- preprocess: (boolean, default=false) If true, preprocessing process is performed.
- train_model: (boolean, default=true) If true, model training process is performed.
- atom_types: (list) Atom types in the target system.

4.3 Methods

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

Args:

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

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

`run` (*self*, *user_atomic_weights_function=None*, *user_optimizer=None*)

Args:

- user_atomic_weights_function: (object) User defined atomic weight function.
- user_optimizer: (object) Tensorflow optimizer. See *Symmetry function*

Main method to run SIMPLE-NN.

CHAPTER

FIVE

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

5.2.1.2 Parameters

feature vector related parameter

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

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

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

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:

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

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

Detailed information of 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 to check the symmetry function of each structure.

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. If not specified, elements will be processed sequentially.

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_X' is described in below:

```
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(4)]
```

The number inside the indicates the number of parameters.

First column indicates the type of symmetry function. Currently, G2, G4, and G5 are available.

Second and third column indicates the type index of neighbor atoms which starts from 1. 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 parameters are the coefficients 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:

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

```
[ 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). In addition, you can set the structure type of each data to set the structure weight for each structure type.

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

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

Reference

CHAPTER

SIX

MODELS

6.1 Introduction

In 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, default: false) If true, training process restart from save file (SAVER.*, checkpoints). If weights, training process restart 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 weights.

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.

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

```
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.
- double_precision: (boolean, default: true) Switch the double precision(true) and single precision(false).
- stddev: (float, default: 0.3) Standard deviation for weights initialization.

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_epoch: (int, default: 10000) The number of total training epoch.
- 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

- save_interval: (int, default: 1000) Interval for saving the neural network potential file.
- show_interval: (int, default: 100) Interval for printing RMSE in LOG file.
- echeck and fcheck: (boolean, default: true, true) If true, SIMPLE-NN check the selected type of RMSE for the validation set. The network is saved when current RMSE is smaller than the RMSE of the previous save point.
- break_max: (int, default: 10) If RMSE of validation set is larger then that of previous save point, break_count increases. Optimization process terminated when break_count >= break_max.
- 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. Zero means a single thread.
- 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
```

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.

Reference

CHAPTER SEVEN

INDICES AND TABLES

- genindex
- modindex
- search