Computer Physics Communications 242 (2019) 95–103

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

SIMPLE-NN: An efficient package for training and executing neural-network interatomic potentials☆

Kyuhyun Lee, Dongsun Yoo, Wonseok Jeong, Seungwu Han *

Department of Materials Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea

ARTICLE INFO

Article history:
Received 12 January 2019
Received in revised form 5 April 2019
Accepted 21 April 2019
Available online 4 May 2019

Keywords:
Potential energy surface
Machine learning potential
Neural network
Molecular dynamics

ABSTRACT

The molecular dynamics (MD) simulation is a favored method in materials science for understanding and predicting material properties from atomistic motions. In classical MD simulations, the interaction between atoms is described by an empirical interatomic potential, so the reliability of the simulation hinges on the accuracy of the underlying potential. Recently, machine learning (ML) based interatomic potentials are gaining attention as they can reproduce potential energy surfaces (PES) of *ab initio* calculations, with a much lower computational cost. Therefore, an efficient code for training ML potentials and inferencing PES in new configurations would widen the application range of MD simulations. Here, we announce an open-source package, SNU Interatomic Machine-learning PotentiaL packagE-version Neural Network (SIMPLE-NN) that generates and utilizes the ML potential based on the artificial neural network with the Behler–Parrinello type symmetry function as descriptors for the chemical environments. SIMPLE-NN uses the Atomic Simulation Environment (ASE) package and Google Tensorflow for high expandability and efficient training, and also supports the in-house code for quasi-Newton method. Notably, the package features a weighting scheme based on the Gaussian density function (GDF), which significantly improves accuracy and reliability of ML potentials by resolving sampling bias that exists in typical training sets. For MD simulations, SIMPLE-NN interfaces with the LAMMPS package. We demonstrate the performance and usage of SIMPLE-NN with examples of SiO_2.

Program summary

Program Title: SIMPLE-NN
Program Files doi: http://dx.doi.org/10.17632/pjv2yr7pvr.1
Licensing provisions: GPLv3
Programming language: Python/C++
Nature of problem: Inferencing the potential energy surface for the given system with accuracy comparable to *ab initio* methods but with much lower computational costs.
Solution method: Calculate descriptor vectors that encode local chemical environment. High-dimensional neural network is used to predict the total energy from the descriptor vectors. The trained neural network can be used for molecular dynamics simulations.

1. Introduction

Molecular dynamics (MD) simulations are widely used in various studies such as predicting material properties and structures [1–4], understanding the atomic motion that is difficult to reveal in experiments [5,6], and identifying mechanism in chemical reactions [7]. Depending on whether the electronic structure is explicitly calculated or not, there are two types of MD: *ab initio* and classical. The *ab initio* MD, which is usually based on the density functional theory (DFT), gives accurate and reliable results. [8–10] However, the method is limited to a system size and timescale less than a few hundreds of atoms and picoseconds, respectively. It is because of the heavy computational costs in solving the Kohn–Sham equation and $O(N^3)$ scaling with respect to the number of atoms N. In contrast, the classical MD describes interatomic interactions with a model potential that consists of analytic functions. [11–13] The method is suitable for large-scale simulations owing to fast evaluation of the potential energy surface and its gradients, with a linear scaling with the system size. However, construction of the proper model function is a formidable task as it requires long experience in development, as well as deep understanding on the given system.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: hansw@snu.ac.kr (S. Han).

https://doi.org/10.1016/j.cpc.2019.04.014
0010-4655/

Recently there has been a surge of interest in applying machine learning (ML) techniques to potential development. [14–17] Unlike traditional interatomic potentials, the ML potentials build upon a purely mathematical structures like neural network or Gaussian process. The parameters in the model are trained by minimizing the error between given *ab initio* data and prediction from the ML potential. The computational cost of MD utilizing ML potentials is far lower than that of *ab initio* MD, enabling large-scale simulations for diverse materials with high accuracy. [18–26]

Among various ML models, the artificial neural network (ANN) and Gaussian process regression (GPR) are currently favored in developing ML potentials. [14–17] While both approaches deliver a similar level of accuracy, each model has its own pros and cons. To be specific, GPR has an explicit solution to parameter fitting such that the model training takes less efforts than ANN. On the other hand, inferencing from the ML potential is generally faster in ANN than GPR since the computational cost of evaluating a GPR model progressively increases with the size of the training set while that of the ANN model is unrelated to the training set. [27] Therefore, the neural network potential (NNP) is more suited for simulating complex phenomena that need a large dataset. (However, we note that recent GPRs [28,29] adopt sparsification schemes like the CUR matrix decomposition [30], which significantly reduces the computational cost.) As of now, several NNP packages have been published: Amp [31], ænet [32], DeePMD-kit [33], ANI-1 [34], PROPhet [35], and ruNNer [36]. (ruNNer is not open to public domain.)

In this paper, we introduce an open-source package, SNU Interatomic Machine-learning PotentiaL packagE-version Neural Network (SIMPLE-NN) for generating NNPs and performing MD simulations based on them. In developing the package, we particularly aim at high performance in training so that a large dataset can be handled efficiently. In the following, we first discuss theories related to NNP, and introduce the structure of SIMPLE-NN. We also demonstrate usage and performance of the package using exemplary systems of SiO_2.

2. Theory

A ML potential is essentially a regression model on the relationship between atomic configurations and total energies. If the model simply maps atomic positions to the total energy, it can only deal with structures with the same number of atoms because the length of input vectors is fixed in most ML models. To overcome this limitation, ML potentials first predict atomic energies from local environments and the total energy is given as a sum of them. The local atomic configurations around certain atoms are encoded into descriptor vectors or matrices which then be inputs of ML models. In SIMPLE-NN, we use the high-dimensional neural network [15] as a ML model and atom-centered symmetry functions [36] as descriptors. Theoretical details are discussed below.

2.1. Descriptor: atom-centered symmetry function

The efficiency and accuracy of ML potentials critically depend on the descriptor that represents chemical environment of an atom. Since the total energy is identical under operations such as rotation and translation of the whole system, and permutation of the atoms of the same chemical species, a proper descriptor should be invariant to these operations. Descriptors such as atom-centered symmetry function [36] and smooth-overlap-of-atomic-positions (SOAP) [37] satisfy the invariant conditions and have been widely used in ML models on material properties. (The Coulomb matrix [38] is also favored in predicting material properties but it may not be appropriate for ML potentials since the Coulomb matrix does not provide enough resolution among similar chemical environments and is hard to differentiate.) In the present package, we support Behler–Parrinello type symmetry function descriptors with one radial and two angular components as in the following:

$$G_i^{\text{radial}} = \sum_j e^{-\eta(R_s - R_{ij})^2} \cdot f_c(R_{ij}), \tag{1}$$

$$G_i^{\text{angular},1} = 2^{1-\zeta} \sum_{j,k\neq j} (1+\lambda\cos\theta_{ijk})^{\zeta} \cdot e^{-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}), \tag{2}$$

$$G_i^{\text{angular},2} = 2^{1-\zeta} \sum_{j,k\neq j} (1+\lambda\cos\theta_{ijk})^{\zeta} \cdot e^{-\eta(R_{ij}^2+R_{ik}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}), \tag{3}$$

where i is the index of the center atom, j and k are those for neighboring atoms, and R_{ij}, R_{ik}, and R_{jk} are distances between them. θ_{ijk} is the angle between a vector from the ith atom to jth atom and a vector from ith atom to kth atom. In Eq. (1), η and R_s determine the width and center of Gaussian functions, respectively, while ζ and λ in Eqs. (2) and (3) change the shape of angular functions. In Eqs. (1)–(3), f_c is a cutoff function with the radius of R_c:

$$f_c(R_{ij}) = \begin{cases} \frac{1}{2}\cos\left(\pi\frac{R_{ij}}{R_c}\right) + \frac{1}{2} & (R_{ij} \le R_c) \\ 0 & (R_{ij} > R_c) \end{cases}. \tag{4}$$

Therefore, $f_c(R_{ij})$ smoothly decreases to zero as R_{ij} approaches R_c and the local environment depends on atoms within R_c. A set of symmetry functions with various choices of (η, R_s, ζ, λ) constitutes a vector $\mathbf{G}_i(\{\mathbf{R}_i\})$ (denoted as $\mathbf{G}_i$ for convenience) that encodes the local atomic environment around the ith atom, where $\mathbf{R}_i$ is the coordinate of ith atom. For training the neural network, every component of $\mathbf{G}$ is scaled into [-1,1] (unless the component has a very narrow distribution), and derivatives of symmetry functions are also scaled according to the relevant symmetry function value.

2.2. High-dimensional neural network

The high-dimensional neural network (HDNN), the base model of SIMPLE-NN, has a structure shown in Fig. 1. It consists of atomic neural network (NN) for each atomic species and the same atomic NN is used for every atom with the same chemical species [see Fig. 1(a)]. For the sake of simplicity, we assume a single-component system in the following discussions but the package can deal with multi-component materials as well. The atomic NN takes the descriptor vector $\mathbf{G}$ of each atom as an input and predicts the atomic energy as an output. Fig. 1(b) shows the detailed structure of atomic NN. There are hidden layers between the input and output layers, and adjacent layers are connected by weights. The values in the kth layer propagate to the next layer as follows:

$$x_j^{k+1} = f\left(\sum_{i=1}^{N^k} x_i^k w_{ij}^k + b^k\right), \tag{5}$$

where i (j) is the index of node in the kth [(k+1)th] layer, N_k the number of nodes in the kth layer, w_{ij}^k the connection weight between x_i^k and x_j^{k+1}, and b^k the bias for the kth layer. In Eq. (5), f is the activation function that attributes nonlinearity to the model. We use a sigmoid function as the activation function except for between the last hidden and output layers where the activation is not applied.

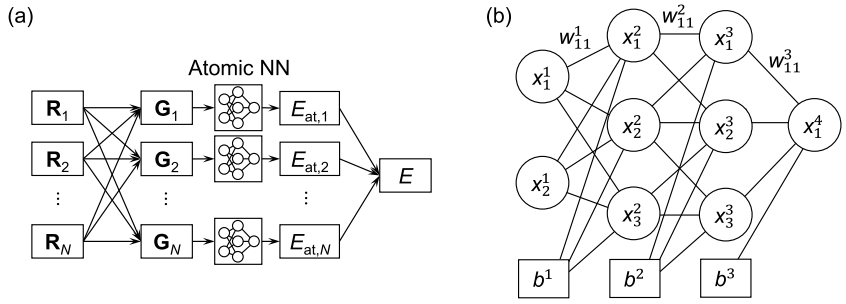

Fig. 1. (a) Schematic diagram of HDNN. $\mathbf{R}_i$ and $\mathbf{G}_i$ indicate the coordinate and corresponding descriptor vector of the ith atom, respectively. $E_{\mathrm{at},i}$ and E are the atomic energy of the ith atom and the total energy, respectively. (b) Schematic diagram of atomic neural network. x_i^k indicates the ith node in the kth layer and w_{ij}^k is the connection weight between x_i^k and x_j^{k+1}. b^k is the bias for the kth layer.

The total energy (E) is then given by summation of atomic energies:

$$E = \sum_{i=1}^{N_a} E_{\mathrm{at}}(\mathbf{G}_i), \tag{6}$$

where N_a is the number of atoms in the given structure and $E_{\mathrm{at}}(\mathbf{G}_i)$ (denoted as $E_{\mathrm{at},i}$ for convenience) is the atomic energy of the ith atom that is calculated by the atomic NN described in the above. The atomic forces are calculated by analytically differentiating E with respect to the atomic position:

$$F_{k,\alpha} = -\frac{\partial E}{\partial R_{k,\alpha}} = -\sum_{i=1}^{N_a}\sum_{s=1}^{N_s} \frac{\partial E_{\mathrm{at},i}}{\partial G_{i,s}} \frac{\partial G_{i,s}}{\partial R_{k,\alpha}}, \tag{7}$$

where $F_{k,\alpha}$ and $R_{k,\alpha}$ are the α-component ($\alpha = x$, y and z) of the force and position vectors of the kth atom, respectively, and N_s is the dimension of $\mathbf{G}$.

The loss function Γ which is target function to be minimized during the training process is a sum of mean squared errors in the total energy and atomic forces between DFT and NNP:

$$\Gamma = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{E_i^{\mathrm{DFT}} - E_i^{\mathrm{NNP}}}{N_i}\right)^2 + \frac{\mu}{3\sum_{i=1}^{M} N_i}\sum_{i=1}^{M}\sum_{j=1}^{N_i} |\mathbf{F}_{ij}^{\mathrm{DFT}} - \mathbf{F}_{ij}^{\mathrm{NNP}}|^2 \tag{8}$$

$$= \mathrm{RMSE(energy)}^2 + \frac{\mu}{3}\mathrm{RMSE(force)}^2, \tag{9}$$

where M is the total number of structures in the training set, N_i is the number of atoms in the ith structure, and $E_i^{\mathrm{DFT(NNP)}}$ and $\mathbf{F}_{ij}^{\mathrm{DFT(NNP)}}$ are the total energy and atomic force (of the jth atom) from DFT (NNP), respectively. In Eq. (8), the scaling parameter μ controls relative importance between the energy and force error. The loss function can also be expressed in terms of root-mean-square error (RMSE) for energy and force as in Eq. (9)

2.3. Weighting scheme for uniform training

In Ref. [39], we have shown that the training points are distributed in the $\mathbf{G}$ space in a highly inhomogeneous fashion, which undermines accuracy and reliability of NNP. For example, during MD simulations, atoms vibrate around equilibrium positions most of the time, and so the training set constructed from the MD trajectories is concentrated around specific $\mathbf{G}$'s. For another example, defects are usually modeled together with a large number of bulk atoms, so the training set is heavily weighted toward the bulk configuration although description on the defect is also crucial. Atomic NNs trained over such biased datasets retain large errors for the under-sampled configurations, which often lead to catastrophic failure in MD.

In order to cure the sampling bias, we proposed in Ref. [39] a method to balance the training level by exploiting the Gaussian density function (GDF; $\rho(\mathbf{G})$). GDF quantifies the sampling density by assigning a Gaussian function to each $\mathbf{G}$ point in the training set:

$$\rho(\mathbf{G}) = \frac{1}{N_{\mathrm{tot}}}\sum_{i=1}^{M}\sum_{j=1}^{N_i} \exp\left(-\frac{1}{2\sigma^2}\frac{|\mathbf{G} - \mathbf{G}_{ij}|^2}{N_s}\right), \tag{10}$$

where σ is the Gaussian width, $\mathbf{G}_{ij}$ is the $\mathbf{G}$ vector of jth atom in the ith structure and N_{tot} indicates the total number of atoms in the entire training set. Weighting factors are then multiplied to the force error such that the weights apply differently on individual atoms according to the sampling density. The modified loss function is as follows:

$$\Gamma = \frac{1}{M}\sum_{i=1}^{M}\eta_i\left(\frac{E_i^{\mathrm{DFT}} - E_i^{\mathrm{NNP}}}{N_i}\right)^2 + \frac{\mu}{3\sum_{i=1}^{M} N_i}\sum_{i=1}^{M}\sum_{j=1}^{N_i}\Theta(\rho^{-1}(\mathbf{G}_{ij}))|\mathbf{F}_{ij}^{\mathrm{DFT}} - \mathbf{F}_{ij}^{\mathrm{NNP}}|^2, \tag{11}$$

where the scaling function $\Theta(\rho^{-1}(\mathbf{G}_{ij}))$ is the atomic weights built upon GDF values and η_i's are additional structural weights which can be used optionally. In Eq. (11), $\Theta(x)$ is a monotonically increasing function that effectively suppresses the sampling bias. From extensive tests, we find that a modified sigmoid function serves well for most cases:

$$\Theta(x) = \frac{Ax}{1 + \mathrm{e}^{-b(x-c)}}, \tag{12}$$

where A is the normalization constant that makes the mean of $\Theta(x)$ to be 1, and b and c are constants determining the function shape. In Eq. (12), we multiply x to the original sigmoid function to lift the upper bound of the scaling function. By doing that, training points with low (high) GDF values (the inverse value of GDF is used in the modified sigmoid) have large (zero) weights, which amplifies effects of the GDF weighting.

In applying the GDF weighting scheme, one needs to carefully choose hyper-parameters such as σ in GDF, and b and c in the scaling function $\Theta(x)$. A proper value of σ should link the Gaussians among similar local structures but still separate those under distinct chemical environments. In Fig. 2(a), we demonstrate this using the distribution of training points ($\mathbf{G}_{ij}$) for Si in SiO_2 systems (see the next section for the details in the training set).[1] When σ is too big, say 0.1, the GDF value is large for most of the training points. (Note the log-scale on the x-axis.) On the other

[1] DFT calculations are performed with the VASP package using the GGA-PBE functional. Further details for the DFT calculation are provided in Section 4.1.

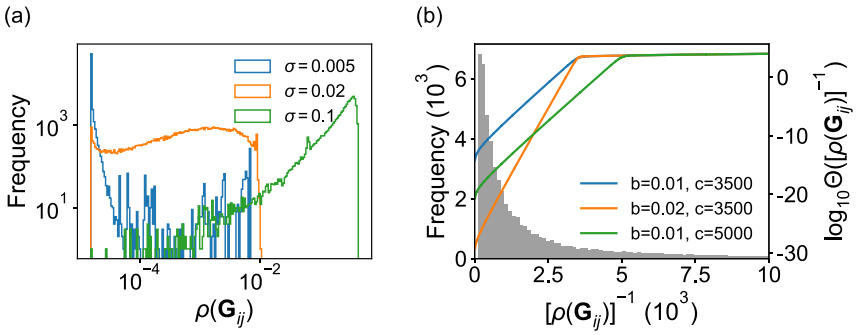

Fig. 2. Guideline for choosing hyper-parameters in GDF weighting. The training set is obtained from model SiO_2 systems and training points ($\{\mathbf{G}_{ij}\}$) are for Si. (a) The distribution of GDF values for the training points with different σ in the Gaussian. ($\rho(\mathbf{G}_{ij})$'s are binned regularly in the log scale.) (b) Frequency of $\rho^{-1}(\mathbf{G}_{ij})$ and scaling function ($\Theta(\rho^{-1}(\mathbf{G}_{ij}))$) with different combinations of b and c are shown in histogram and lines, respectively.

hand, if σ is too small (0.005), the GDF value is too small even for configurations that are over-sampled. With σ = 0.02, both over- and under-sampled configurations are well represented by high and low GDF values, respectively. On the other hand, c in $\Theta(x)$ effectively separates points between highly concentrated and sparsely sampled training points. In Fig. 2(b), it is seen that the training points with $\rho^{-1}(\mathbf{G}_{ij})$ value lower than c are weighted with very small numbers. The b value in $\Theta(x)$ controls the weighting magnitude. It is chosen to determine the degree to which the function decays in the lower $\rho^{-1}(\mathbf{G}_{ij})$ region (compare b = 0.01 with b = 0.02 in Fig. 2(b)). To choose a proper value for b, it is necessary to examine final distributions of force errors after optimization. (In most cases, $b = 1$ gives reasonable results and an automatic parameter selection for σ and c is available in the SIMPLE-NN code.) The effect of GDF will be explained in the next section. For more information of sampling bias and effects of the GDF weighting, we refer to Ref. [39].

3. SIMPLE-NN code

In this section, we introduce the development philosophy and the code structure of SIMPLE-NN. In developing the present package, we concentrate on computational performance in training as well as expandability to other descriptors or ML models. Due to the high dimensionality of ML model, a strong optimization algorithm based on approximate Hessians such as quasi-Newton methods is required. Since the Tensorflow [40] library as of now provides only gradient-descent-based algorithms, we additionally provide a built-in code of Limited Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) that can interoperate with Tensorflow. In addition, we implement the GDF weighting scheme for reliable and balanced training (see above).

SIMPLE-NN is mostly written in Python. However, computationally intensive parts such as calculations of descriptors, their derivatives, and GDF weighting factors, are written in C/C++ to enhance performance. Furthermore, the computation of **G** and its derivatives from atomic positions, which takes significant CPU time, is parallelized with the message-passing interface (MPI). For training NNP, we take advantage of hardware-level optimization supported by Tensorflow. In detail, each epoch of NNP optimization consists of data preparation and model training. Tensorflow utilizes CPUs for data preparation and GPUs for model training. In a synchronous implementation, GPU remains idle when CPU is preparing data, and vice versa. By using a data-pipelining feature supported by Tensorflow, SIMPLE-NN executes the two tasks in an asynchronous way such that the model training for the present epoch and data preparation for the next epoch are carried out concurrently by GPU and CPU, respectively. In addition, the data preparation process is parallelized for multicore CPU. To perform MD simulations with the trained NNP, we implement a new pair style for LAMMPS package [41]. The parallelized LAMMPS code achieves a good scalability with respect to the number of atoms or the number of CPU cores as shown in Fig. 3.

SIMPLE-NN is also easily extendable to various input formats and ML models. For parsing the reference DFT data, we use Atomic Simulation Environment (ASE) package [42] that supports output formats of popular *ab initio* programs such as VASP [43], Quantum espresso [44], and Gaussian [45]. In addition, subroutines for calculating descriptors and optimizing neural network have a modular structure, and so users can implement their own descriptors or ML models into SIMPLE-NN. Since Tensorflow supports built-in functions for handling various neural network models such as convolutional NN and recurrent NN, efforts for implementing a new model would be low. However, the interface with LAMMPS is not modularized, which will be addressed in a future release.

Fig. 4 shows the schematic workflow of SIMPLE-NN. The generation of NNP starts with calculating **G** vectors and their derivatives from the DFT data. Next, in the preprocessing part, parameters for scaling **G** and GDF weighting are calculated and the dataset is split into a training set and a validation set. Based on the preprocessed dataset, NNP is optimized using Tensorflow until the root-mean-square errors (RMSE) of energy and force for the validation set reach certain values that may vary with the system. The parameters of optimized NNP are stored as a text file, which is then used for MD simulations within the LAMMPS package. Below, we discuss on the detailed usage of the program.

Usage. The following script named `run.py` executes the workflow introduced above:

```
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()
```

In the script, `Simple_nn` class is initialized with the instances of `Symmetry_function` class and `Neural_network` class. These classes also define the methods for calculating descriptor vectors and optimizing ML model. In the `run` method of `Simple_nn` class, the workflow above (except the MD simulation) is executed according to the detailed setting defined in `input.yaml` file.

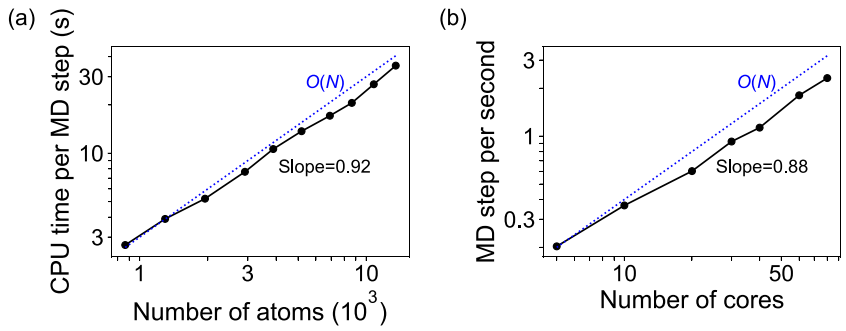

Fig. 3. The scalability of LAMMPS-MD simulations using NNP with respect to (a) the number of atoms (with 40 cores) and (b) the number of CPU cores (with 13500 atoms). Total CPU time and wall time is measured for (a) and (b), respectively. The hypothetical linear scaling is indicated by dotted lines. The calculation is performed on Intel Xeon Gold 6138 CPUs (2.00 GHz) that contain 20 cores. The test system is SiO_2.

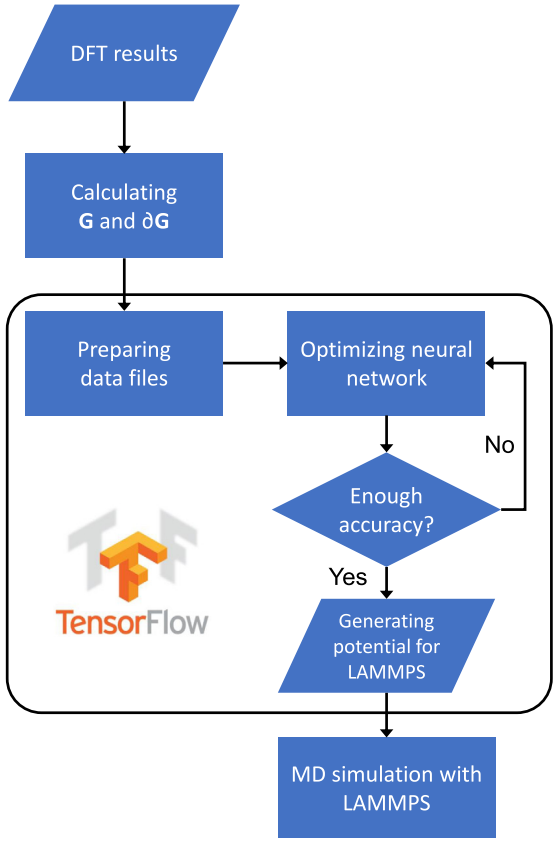

Fig. 4. Schematic plot for overall process of SIMPLE-NN. The round box envelops parts that make use of Tensorflow. LAMMPS package is used for MD simulation.

The basic format of `input.yaml` is as follows:

```
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
```

The `input.yaml` includes information about the target system, whether each procedure is executed, as well as parameters for the feature and model classes. Detailed description on the parameters in `input.yaml` can be found in the online manual [46]. To run `generate_feature` and `preprocess`, one needs additional files named '`params_XX`' (where XX is the atom type) and '`str_list`' which include parameters for symmetry functions and the paths to the reference *ab initio* data files, respectively. Descriptor vectors and their derivatives are first saved in the pickle format (binary) separately for each structure, which is then packed into a binary format of tfrecord with additional values such as GDF weighting parameters calculated in the preprocessing part. The dataset preparation entails optimization of the parameters in the NN model. The detailed settings such as network sizes, optimization algorithm, and the type of loss function are controlled through `input.yaml`. Because of the heavy computational cost in calculating the atomic forces, the optimization that reduces both energy and force errors requires much longer time than minimizing the energy-only loss function. To avoid the huge computational cost in large datasets, one can use the energy-only loss function in the initial training steps, and then add the force errors to the loss function afterwards. In this way, the computational cost is saved substantially without compromising the accuracy of the NNP. Throughout the optimization process, the neural network model is saved with a Tensorflow format (`SAVER.*` and `checkpoints`) and LAMMPS potential file (`potential_saved`). To note, our experience indicates that the loss function with only energy errors is vulnerable to the risk of overfitting while training with forces only suffers from large errors in the total energy. Thus, we recommend training with both energy and forces for robust NNP. One can use the obtained NNP in LAMMPS by adding the commands like below in the LAMMPS input file:

```
pair_style nn
pair_coeff * * potential_saved Ni
```

'`potential_saved`' is the optimized potential file from the NNP optimization part and one needs to specify the element name for each atom type like other pair styles in LAMMPS. 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 at users' discretion.

4. Example

4.1. Model system and training NNP

We use SiO_2 as a test example to demonstrate actual procedures of using SIMPLE-NN. For generating the training set within

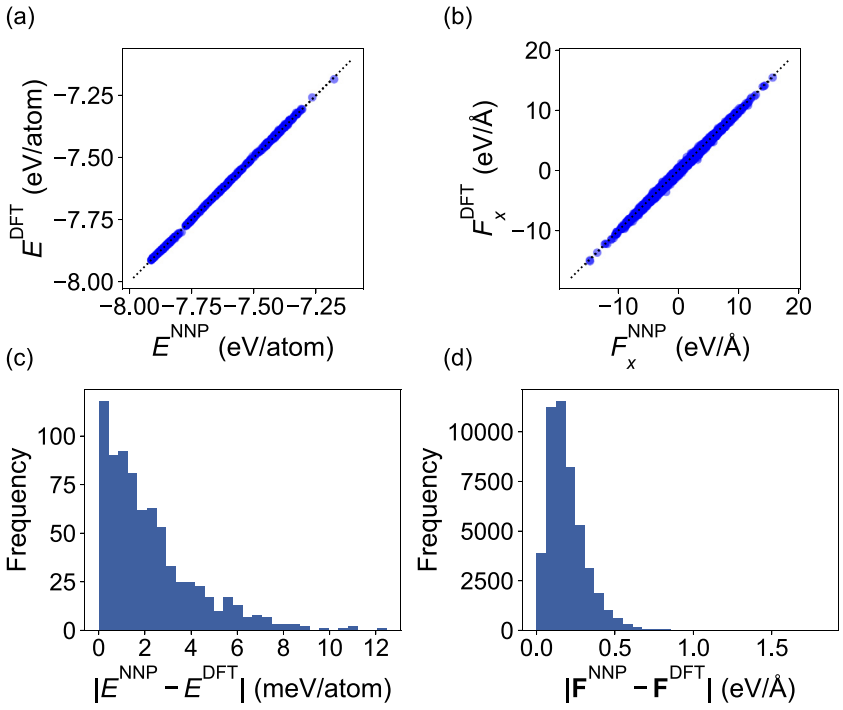

Fig. 5. Comparison of DFT and optimized NNP for the validation set. Correlation graphs between (a) E^{NNP} and E^{DFT} and (b) F_x^{NNP} and F_x^{DFT}. Histograms for (c) $|E^{NNP} - E^{DFT}|$ and (d) $|\mathbf{F}^{NNP} - \mathbf{F}^{DFT}|$.

DFT, we employ VASP [43] with GGA-PBE for the exchange–correlation functional [47]. The cutoff energy of 500 eV is used and convergence tests for **k**-point mesh are carried out to ensure the convergence within 10 meV/atom. Three types of crystals (α-quartz, α-cristobalite, tridymite), amorphous, and liquid phase are used for the training set. In detail, we sample snapshots every 40 fs from MD trajectories at 3000 and 2500 K for liquids, 500 K for crystals, and from 2500 to 300 K (cooling rate of 73 K/ps) from liquid to amorphous structures. In addition, crystal structures that are distorted by isotropic compression/expansion, volume-conserving mono-axial strain, and shear strain are also added. In total, the training set contains 3,048 structures that consist of 48,978 Si and 97,956 O atoms.

As descriptors, 70 symmetry functions – 8 radial components per two-body combination and 18 angular components per three-body combination – are used for each atom type. We use a network that consists of two hidden layers with 30 nodes for each layer. Thus, our network contains 3,030 weights and 61 biases in total. The Adaptive Moment Estimation (Adam) optimizer with a batch size of 10 is used for the optimization, and we train NNP by minimizing energy and force errors simultaneously. We randomly choose 10% of dataset and use them as the validation set.

After optimization, NNP shows RMSE of 3 meV/atom (energy) and 0.23 eV/Å (force) for the validation set. Fig. 5 shows the distribution of the errors. Linear correlations between DFT and NNP results are notable for the energy [Fig. 5(a)] and x-component of atomic force [Fig. 5(b)] (other components also show similar behaviors). Fig. 5(c) and Fig. 5(d) display the frequency of absolute errors in the energy and force, respectively. It is found that the 80th percentile of the validation set shows errors that are less than 5 meV/atom and 0.3 eV/Å.

4.2. MD Simulations

The quality of NNP is further tested by MD simulations. Fig. 6(a) shows the energy of the amorphous SiO_2 (108 atoms) at 500 K performed with NNP for 20 ps (solid line). From MD trajectory, snapshots are sampled for every 200 fs and the total energy is recalculated with DFT (dots) using the same setting as in Section 4.1. It is seen that energies from DFT and NNP agree within 3 meV/atom, confirming that NNP can faithfully reproduce potential energy surfaces of DFT.

In addition, we perform melt-quench simulations using DFT and NNP independently and compare structural properties of the amorphous phase. We use supercells with 108 atoms for DFT and 108 and 480 atoms for NNP. Starting from the pre-melted structures at 3000 K, the structure is melted at 2500 K for 30 ps and quenched from 2500 K to 300 K for 30 ps, and the final optimization are performed for the quenched structure. The radial distribution function (RDF) and the angular distribution function (ADF) during 500 K MD of amorphous phases are shown in Fig. 6(b)–(d). Overall, the DFT and NNP results are in good agreements and also consistent with a previous calculation [48]. It is seen in Fig. 6(c) that NNP reproduces a sharp peak at 109.5° that comes from the tetrahedral bond angle for Si atoms. For O atoms, a broad ADF is noticeable, which reflects flexible Si-O-Si bond in amorphous silica. A small peak at 90° originates from the 4-membered ring structure.

4.3. GDF Weighting

In the above example, we optimized NNP without applying GDF weighting. In order to demonstrate the effect of GDF weighting, we set the hyper-parameters of GDF function and the scaling function $\Theta(x)$ according to the guideline described in the previous section. ($b = 0.02$ for both Si and O, and $c = 3500$ and 1000 for Si and O, respectively). Figs. 7(a) and 7(b) plot magnitudes of force errors with respect to GDF values of the corresponding atom. The force RMSEs from NNPs with or without GDF weighting are identically 0.21 and 0.14 eV/Å for Si and O, respectively. However, force errors for atoms with low GDF values are reduced when GDF weighting is applied (from 0.36 (0.30) to 0.31 (0.24) eV/Å for Si (O) atoms). If the force error in the region with low GDF is not sufficiently reduced, it is recommended to increase b value in the

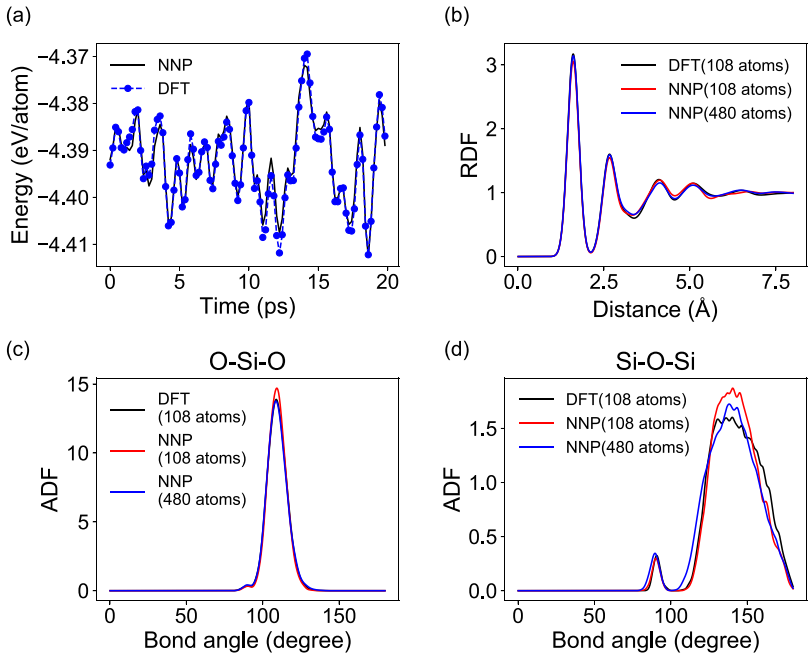

Fig. 6. (a) Comparison of NNP (solid line) and DFT (solid disks) energies along the MD trajectory of amorphous SiO_2 at 500 K. The trajectory is calculated by NNP and configurations for DFT calculations are sampled every 200 fs. (b)–(d) Comparison of structural properties of amorphous SiO_2 between DFT and NNP. The structures are generated by independent melt-quench simulations. (b) Total radial distribution function (RDF), angle distribution functions (ADFs) for (c) O-Si-O and (d) Si-O-Si. For NNP, supercells of two different sizes (108 and 480 atoms) are used.

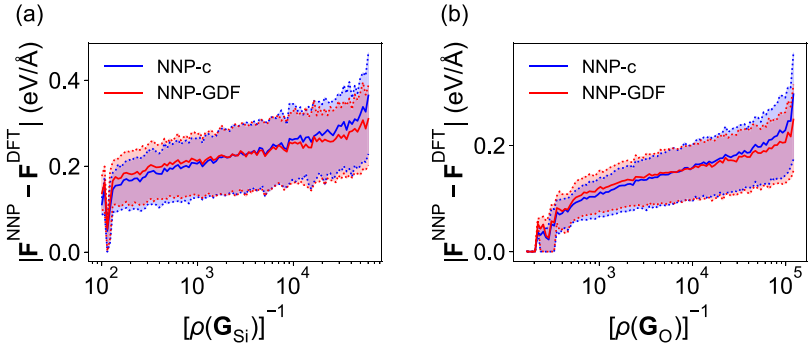

Fig. 7. Comparison of force errors between conventional NNP (NNP-c) and NNP with GDF weighting (NNP-GDF) for (a) Si and (b) O atoms in the case of SiO_2. The errors are interval-averaged by regular binning in log-scale. The solid lines indicate the mean value within each bin and semi-transparent shades envelope ranges between the first and third quartiles of the distribution.

scaling function. In the present example, the training set includes mostly homogeneous structures, and so the sampling bias is not severe and may not need GDF weighting. However, for structures including defects or bond breaking/forming, the GDF weighting is highly beneficial. [39] SIMPLE-NN provides utility figures for force errors with respect to GDF like Fig. 7, and we recommend the users to monitor them to check the problems related to the sampling bias.

4.4. L-BFGS

In the above example of SiO_2, the Adam minimizer was effective for optimizing NNP. However, for more complicated systems involving various bonding types, we often find that gradient-descent methods such as Adam fail to attain enough accuracy even after a large number of epochs. In this case, L-BFGS is more effective than Adam optimizer. Fig. 8 compares the performance of L-BFGS and Adam with either mini-batch or full-batch style. Since the CPU time required for each epoch is different among the methods, RMSE is plotted as a function of elapsed CPU time for the fair comparison. For the test, we use 500 snapshots from MD trajectories of amorphous SiO_2 at 500 K. It is clear that L-BFGS shows faster convergence than Adam methods. One disadvantage in L-BFGS is that it becomes computationally expensive with the size of the training set. Nevertheless, our experiences indicate that L-BFGS significantly outperforms the Adam optimization in most cases.

5. Conclusion

In summary, we introduced an efficient package named SIMPLE-NN that can build up neural network potentials at the accuracy comparable to *ab initio* methods and carry out MD simulations on a large scale. We optimized the code performance for NNP training by incorporating various features such as parallelization, Hessian-based optimizer (L-BFGS), and utilizing GPU, which are either supported by Tensorflow or in-house codes. The MD simulations are carried out through LAMMPS. SIMPLE-NN provides high expandability via ASE, Tensorflow, and modularized code architecture. The program also features the GDF

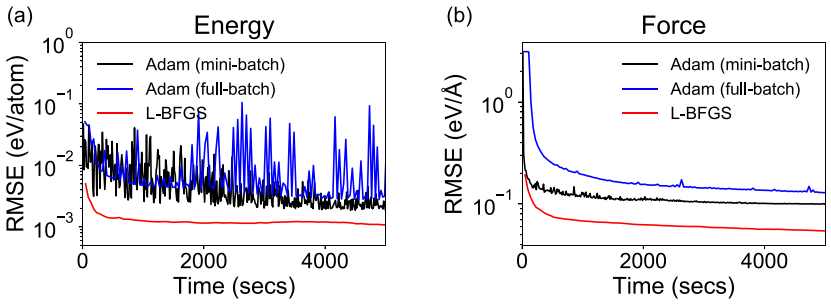

Fig. 8. Comparison of performance among different optimizers. RMSEs in (a) energy and (b) force are compared with respect to the elapsed time. 500 snapshots during MD simulation of SiO_2 are used as the training set.

weighting scheme that can effectively overcome the sampling bias that is serious in most training sets. The performance of SIMPLE-NN was demonstrated with the SiO_2 system including crystal and amorphous structures. By providing an efficient NNP code for general purposes, the present work will contribute to expanding the application areas of MD simulations based on the machine learning potentials.

Acknowledgments

Kyuhyun Lee and Dongsun Yoo contributed equally to this work. This work was supported by the Technology Innovation Program (or Industrial Strategic Technology Development Program [10052925, Atomistic process and device modeling of sub-10 nm scale transistors]) funded By the Ministry of Trade, Industry & Energy (MOTIE, Korea) and Creative Materials Discovery Program through the National Research Foundation of Korea (NRF) funded by Ministry of Science and ICT (2017M3D1A1040689). The computations were performed at the KISTI supercomputing center (Grant No. KSC-2018-C3-0022).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cpc.2019.04.014.

References

[1] B.M. Foley, V. Stavila, C.D. Spataru, F. Léonard, A.A. Talin, P.E. Hopkins, M.E. Foster, R.E. Jones, K.J. Erickson, M.D. Allendorf, Adv. Mater. 27 (22) (2015) 3453–3459, http://dx.doi.org/10.1002/adma.201501078.
[2] X. Zhang, H. Xie, M. Hu, H. Bao, S. Yue, G. Qin, G. Su, Phys. Rev. B 89 (5) (2014) 1–7, http://dx.doi.org/10.1103/PhysRevB.89.054310.
[3] H. Song, Y. Kang, H.-H. Nahm, S. Han, Phys. Status Solidi b 252 (8) (2015) 1872–1876, http://dx.doi.org/10.1002/pssb.201451767.
[4] S. Le Roux, A. Bouzid, K.Y. Kim, S. Han, A. Zeidler, P.S. Salmon, C. Massobrio, J. Chem. Phys. 145 (8) (2016) 084502, http://dx.doi.org/10.1063/1.4961265.
[5] X. He, Y. Zhu, Y. Mo, Nature Commun. 8 (May) (2017) 1–7, http://dx.doi.org/10.1038/ncomms15893.
[6] Y. Youn, D. Yoo, H. Song, Y. Kang, K.Y. Kim, S.H. Jeon, Y. Cho, K. Chae, S. Han, J. Mater. Chem. C 6 (5) (2018) 1015–1022, http://dx.doi.org/10.1039/C7TC05278B.
[7] M. Ghoussoub, S. Yadav, K.K. Ghuman, G.A. Ozin, C.V. Singh, ACS Catal. 6 (10) (2016) 7109–7117, http://dx.doi.org/10.1021/acscatal.6b01545.
[8] M. Born, R. Oppenheimer, Ann. Phys. 389 (20) (1927) 457–484, http://dx.doi.org/10.1002/andp.19273892002.
[9] W. Kohn, L.J. Sham, Phys. Rev. 140 (4A) (1965) A1133–A1138, http://dx.doi.org/10.1103/PhysRev.140.A1133.
[10] R. Car, M. Parrinello, Phys. Rev. Lett. 55 (22) (1985) 2471–2474, http://dx.doi.org/10.1103/PhysRevLett.55.2471.
[11] J. Tersoff, Phys. Rev. B 37 (12) (1988) 6991–7000, http://dx.doi.org/10.1103/PhysRevB.37.6991.
[12] S.M. Foiles, M.I. Baskes, M.S. Daw, Phys. Rev. B 33 (12) (1986) 7983–7991, http://dx.doi.org/10.1103/PhysRevB.33.7983.
[13] T.P. Senftle, S. Hong, M.M. Islam, S.B. Kylasa, Y. Zheng, Y.K. Shin, C. Junkermeier, R. Engel-Herbert, M.J. Janik, H.M. Aktulga, T. Verstraelen, A. Grama, A.C.T. van Duin, NPJ Comput. Mater. 2 (1) (2016) 15011, http://dx.doi.org/10.1038/npjcompumats.2015.11.
[14] T.B. Blank, S.D. Brown, A.W. Calhoun, D.J. Doren, J. Chem. Phys. 103 (10) (1995) 4129–4137, http://dx.doi.org/10.1063/1.469597.
[15] J. Behler, M. Parrinello, Phys. Rev. Lett. 98 (14) (2007) 146401, http://dx.doi.org/10.1103/PhysRevLett.98.146401.
[16] A.P. Bartók, M.C. Payne, R. Kondor, G. Csányi, Phys. Rev. Lett. 104 (13) (2010) 136403, http://dx.doi.org/10.1103/PhysRevLett.104.136403.
[17] A.P. Bartók, G. Csányi, Int. J. Quantum Chem. 115 (16) (2015) 1051–1057, http://dx.doi.org/10.1002/qua.24927.
[18] J.R. Boes, J.R. Kitchin, Mol. Simul. 43 (5–6) (2017) 346–354, http://dx.doi.org/10.1080/08927022.2016.1274984.
[19] W. Li, Y. Ando, E. Minamitani, S. Watanabe, J. Chem. Phys. 147 (21) (2017) 214106, http://dx.doi.org/10.1063/1.4997242.
[20] M. Hellström, J. Behler, J. Phys. Chem. Lett. 7 (17) (2016) 3302–3306, http://dx.doi.org/10.1021/acs.jpclett.6b01448.
[21] S.K. Natarajan, J. Behler, Phys. Chem. Chem. Phys. 18 (41) (2016) 28704–28725, http://dx.doi.org/10.1039/C6CP05711J.
[22] V. Quaranta, M. Hellström, J. Behler, J. Phys. Chem. Lett. 8 (7) (2017) 1476–1483, http://dx.doi.org/10.1021/acs.jpclett.7b00358.
[23] M. Hellström, M. Ceriotti, J. Behler, J. Phys. Chem. B 122 (44) (2018) 10158–10171, http://dx.doi.org/10.1021/acs.jpcb.8b06433.
[24] B. Onat, E.D. Cubuk, B.D. Malone, E. Kaxiras, Phys. Rev. B 97 (9) (2018) 094106, http://dx.doi.org/10.1103/PhysRevB.97.094106.
[25] S. Gabardi, E. Baldi, E. Bosoni, D. Campi, S. Caravati, G.C. Sosso, J. Behler, M. Bernasconi, J. Phys. Chem. C 121 (42) (2017) 23827–23838, http://dx.doi.org/10.1021/acs.jpcc.7b09862.
[26] Z. Li, J.R. Kermode, A. De Vita, Phys. Rev. Lett. 114 (9) (2015) 096405, http://dx.doi.org/10.1103/PhysRevLett.114.096405.
[27] A. Kamath, R.A. Vargas-Hernández, R.V. Krems, T. Carrington, S. Manzhos, J. Chem. Phys. 148 (24) (2018) 241702, http://dx.doi.org/10.1063/1.5003074.
[28] D. Dragoni, T.D. Daff, G. Csányi, N. Marzari, Phys. Rev. Mater. 2 (2018) 013808, http://dx.doi.org/10.1103/PhysRevMaterials.2.013808.
[29] A.P. Bartók, J. Kermode, N. Bernstein, G. Csányi, Phys. Rev. X 8 (2018) 041048, http://dx.doi.org/10.1103/PhysRevX.8.041048.
[30] M.W. Mahoney, P. Drineas, Proc. Natl. Acad. Sci. 106 (3) (2009) 697–702, http://dx.doi.org/10.1073/pnas.0803205106, arXiv:https://www.pnas.org/content/106/3/697.full.pdf.
[31] A. Khorshidi, A. Peterson, Comput. Phys. Comm. 207 (2016) 1–15, http://dx.doi.org/10.1016/j.cpc.2016.05.010.
[32] N. Artrith, A. Urban, Comput. Mater. Sci. 114 (2016) 135–150, http://dx.doi.org/10.1016/j.commatsci.2015.11.047.
[33] H. Wang, L. Zhang, J. Han, W. E, Comput. Phys. Comm. 228 (2018) 178–184, http://dx.doi.org/10.1016/j.cpc.2018.03.016.
[34] J.S. Smith, O. Isayev, A.E. Roitberg, Chem. Sci. 8 (4) (2017) 3192–3203, http://dx.doi.org/10.1039/C6SC05720A.
[35] B. Kolb, L.C. Lentz, A.M. Kolpak, Sci. Rep. 7 (1) (2017) 1192, http://dx.doi.org/10.1038/s41598-017-01251-z.
[36] J. Behler, J. Chem. Phys. 134 (7) (2011) 074106, http://dx.doi.org/10.1063/1.3553717.
[37] A.P. Bartók, R. Kondor, G. Csányi, Phys. Rev. B 87 (18) (2013) 184115, http://dx.doi.org/10.1103/PhysRevB.87.184115.
[38] M. Rupp, A. Tkatchenko, K.-R. Müller, O.A. von Lilienfeld, Phys. Rev. Lett. 108 (5) (2012) 058301, http://dx.doi.org/10.1103/PhysRevLett.108.058301.
[39] W. Jeong, K. Lee, D. Yoo, D. Lee, S. Han, J. Phys. Chem. C 122 (39) (2018) 22790–22795, http://dx.doi.org/10.1021/acs.jpcc.8b08063.
[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, Software available from tensorflow.org, 2015.

[41] S. Plimpton, J. Comput. Phys. 117 (1) (1995) 1–19, http://dx.doi.org/10.1006/jcph.1995.1039.

[42] A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I.E. Castelli, R. Christensen, M. Dułak, J. Friis, M.N. Groves, B. Hammer, C. Hargus, E.D. Hermes, P.C. Jennings, P. Bjerre Jensen, J. Kermode, J.R. Kitchin, E. Leonhard Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. Bergmann Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K.S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, K.W. Jacobsen, J. Phys.: Condens. Matter 29 (27) (2017) 273002, http://dx.doi.org/10.1088/1361-648X/aa680e.

[43] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (16) (1996) 11169–11186, http://dx.doi.org/10.1103/PhysRevB.54.11169, arXiv:0927-0256(96)00008.

[44] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. De Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A.P. Seitsonen, A. Smogunov, P. Umari, R.M. Wentzcovitch, J. Phys. Condens. Matter 21 (39) (2009) http://dx.doi.org/10.1088/0953-8984/21/39/395502, arXiv:0906.2569.

[45] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, G. Scalmani, V. Barone, G.A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A.V. Marenich, J. Bloino, B.G. Janesko, R. Gomperts, B. Mennucci, H.P. Hratchian, J.V. Ortiz, A.F. Izmaylov, J.L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V.G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J.A. Montgomery, J.E. Peralta, F. Ogliaro, M.J. Bearpark, J.J. Heyd, E.N. Brothers, K.N. Kudin, V.N. Staroverov, T.A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A.P. Rendell, J.C. Burant, S.S. Iyengar, J. Tomasi, M. Cossi, J.M. Millam, M. Klene, C. Adamo, R. Cammi, J.W. Ochterski, R.L. Martin, K. Morokuma, O. Farkas, J.B. Foresman, D.J. Fox, Gaussian16 Revision B.01, Gaussian Inc., Wallingford CT, 2016.

[46] SIMPLE-NN documentation can be found at http://mtcg.snu.ac.kr/doc/index.html.

[47] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868, http://dx.doi.org/10.1103/PhysRevLett.77.3865.

[48] Y. Youn, Y. Kang, S. Han, Comput. Mater. Sci. 95 (2014) 256–262, http://dx.doi.org/10.1016/j.commatsci.2014.07.053.