Computer Physics Communications 256 (2020) 107402

Contents lists available at ScienceDirect

Computer Physics Communications

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

PANNA: Properties from Artificial Neural Network Architectures☆,☆☆

Ruggero Lot, Franco Pellegrini, Yusuf Shaidu, Emine Küçükbenli *

SISSA, Via Bonomea 265, I-34136 Trieste, Italy

ARTICLE INFO

Article history:
Received 5 July 2019
Received in revised form 23 March 2020
Accepted 14 May 2020
Available online 30 May 2020

Keywords:
Machine learning
Potential energy surface
Neural network
Force field
Molecular dynamics

ABSTRACT

Prediction of material properties from first principles is often a computationally expensive task. Recently, artificial neural networks and other machine learning approaches have been successfully employed to obtain accurate models at a low computational cost by leveraging existing example data. Here, we present a software package "Properties from Artificial Neural Network Architectures" (PANNA) that provides a comprehensive toolkit for creating neural network models for atomistic systems following the Behler–Parrinello topology. Besides the core routines for neural network training, it includes data parser, descriptor builder for Behler–Parrinello class of symmetry functions and force-field generator suitable for integration within molecular dynamics packages. PANNA offers a variety of activation and cost functions, regularization methods, as well as the possibility of using fully-connected networks with custom size for each atomic species. PANNA benefits from the optimization and hardware-flexibility of the underlying TensorFlow engine which allows it to be used on multiple CPU/GPU/TPU systems, making it possible to develop and optimize neural network models based on large datasets.

Program summary

Program title: PANNA — Properties from Artificial Neural Network Architectures
CPC Library link to program files: http://dx.doi.org/10.17632/mcryj6cnnh.1
Licensing provisions: MIT
Programming language: Python, C++
Nature of problem: A workflow for machine learning atomistic properties and interatomic potentials using neural networks.
Solution method: This package first transforms the user supplied data into pairs of precomputed input (Behler–Parrinello [1] class of symmetry functions) and target output (energy and forces) for the neural network model. The data are then packed to enable efficient reading. A user-friendly interface to TensorFlow [2] is provided to instantiate and train neural network models with varying architectures within Behler–Parrinello topology and with varying training schedules. The training can be monitored and validated with the provided tools. The derivative of the target output with respect to the input can also be used jointly in training, e.g. in the case of energy and force training. The interface with molecular dynamics codes such as LAMMPS [3] allows the neural network model to be used as an interatomic potential.
Additional comments including restrictions and unusual features: The underlying neural network training engine, TensorFlow, is a prerequisite of PANNA. While there is a special LAMMPS integration performed via a patch distributed within PANNA, the network potentials can be deposited into OpenKIM [4] database and can be used with a wide range of molecular dynamics codes.

The package allows different network architectures to be used for each atomic species, with different trainability setting for each network layer. It provides tools of exchanging weights between atomic species, and provides the option of building a Radial Basis Function network. The software is parallelized to take advantage of hardware architectures with multiple CPU/GPU/TPUs.

☆ The review of this paper was arranged by Prof. D.P. Landau.

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

* Correspondence to: John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge Massachusetts 02138, USA.
E-mail address: kucukben@g.harvard.edu (E. Küçükbenli).

https://doi.org/10.1016/j.cpc.2020.107402

References:
[1] J. Behler and M. Parrinello, Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces, Phys. Rev. Lett. 98, 146401 (2007).
[2] M. Abadi et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). URL: https://www.tensorflow.org
[3] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, J. Comp. Phys. 117 1-9 (1995). URL: http://lammps.sandia.gov
[4] E. B. Tadmor, R. S. Elliott, J. P. Sethna, R. E. Miller and C. A. Becker, The Potential of Atomistic Simulations and the Knowledgebase of Interatomic Models, JOM, 63, 17 (2011). URL: https://openkim.org

1. Introduction

In recent years machine learning has gained increasing popularity in material science and chemical physics due to several potentially high-impact applications: Electronic properties such as atomization energy, polarizability, infrared spectra and excitation energies have been calculated via machine learning with satisfactory accuracy for small organic molecules [1–3] and molecular crystals [4]. New methodologies have been developed for inorganic crystals in order to successfully predict electronic structure properties such as the density of states [5,6], Debye temperature [7], band gap of inorganic crystals [8]. In these studies, various approaches have been employed to represent atomic systems to be used within machine learning algorithms. Some examples from literature are vectors with respect to local frames [9], symmetry-based descriptors [10–12], graphs [13], matrices [14], list of bonds [15], chemical formulas [16] or molecular structures [17]. These representations are often coupled with an appropriate machine learning model such as feedforward neural networks [18], convolutional networks [19], Gaussian processes [20]. It has been demonstrated that a successful pairing of representation and machine learning model can be found to predict local properties such as atomic charges [21] or electronic density [22–24]. Forces on atoms, a local property that is trivially linked to the global total energy via gradients, have been calculated both analytically [25] and numerically as a target of the machine learnt model [26]. Fast and accurate interatomic force fields have been developed based on neural networks [27–29] and Gaussian processes [30–32].

Despite the large amount of publications that demonstrate the success of machine-learnt force fields in proof-of-concept scale, there have been fewer investigations where they have been used in production scale and found to successfully complement the existing force field database, e.g. in the case of disordered materials [33], catalytic surfaces [34], nanoclusters [35]. Although available datasets [36–38] and innovative machine learning methods [39] increase, considerations remain in adopting these methods for applications beyond the proof-of-concept scale. In particular, a high performance neural network training engine requires substantial effort to implement. Furthermore, the accuracy of a network model depends on the successful optimization of several parameters of the model, and introducing such hyperparameter changes in a neural network program may require extensive coding overhead for non-experts. For the resulting models to be used in realistic research questions, integration of these models into popular massively parallel molecular dynamics (MD) packages is a must. Hence for neural network potentials to find their place in the state-of-the art production arsenal of application scientists, tools that offer accuracy and performance for a wide range of applications are needed. Likewise, challenging real world applications can reveal the areas of improvement for obtaining better neural network models. One way forward is an open source package that gives a simple way to develop, test and use different neural network models efficiently for large amounts of datasets, integrated with MD and other material science simulation packages, without re-inventing the infrastructure necessary for atomistic machine learning training each time. To match this need we have built PANNA (Properties from Artificial Neural Network Architectures), a Python package based on TensorFlow that simplifies the process of training, testing and using neural network models in atomistic calculations.

PANNA includes tools to parse and convert ab initio simulation files into neural network (NN) inputs, the training and monitoring of NN models, conversion of the network into interatomic potential format that can be used in MD simulations. The implementation supports fully-connected network architecture, a variety of input and output formats, and it can be run on large parallel CPU/GPU/TPU systems.

PANNA complements the landscape of available software packages for generation of neural network potentials: AMP [40] has been one of the first to bring neural network potentials to the large community of condensed matter physicists and chemists thanks to its open source codebase written in Fortran, a language well-adapted for in scientific community and thanks to its ASE [41] interface. Another package written in Fortran, AEnet [42], features unique descriptor functions based on Chebyshev expansion. RuNNer code [43,44], available upon contacting the authors, is also written in a low level language (C++) and it is historically relevant as it is the code much of the methodology for neural network potentials has been originally developed with. Other packages close to PANNA in scope, written in Python, and support ASE interface for deployment instead of PANNA's LAMMPS and OpenKIM interfaces, are SchNetPack [19] and TorchANI [45]. SchNetPack and TorchANI, as well as the newly introduced KLIFF [46] use PyTorch [47] back-end for ML operations in difference to TensorFlow back-end of PANNA. Furthermore SchNetPack architecture is based on continuous filter convolutions rather than the Behler–Parrinello topology. Two other packages that do implement the BP topology with TF back-end are DeePMD [48] and SIMPLE-NN [49]. DeepMD differs in its choice of descriptors which uses a vectorial descriptor and a cutoff of interactions based on number of nearest neighbors rather than distance from a central atom. SIMPLE-NN is the closest package to PANNA in terms of implementation of core features, and as the name suggests, it is a minimalist version, a TF-based interatomic potential builder with BP descriptors and BP topology. With SIMPLE-NN the user is expected to write additional Python scripts to use the code and to visualize the output. The extended features of PANNA such as network variability between species or analysis and visualization tools embedded within Tensorboard appear out of the scope of this minimalist package. Finally, one of the largest packages for ML in chemistry is ChemML [50], which is also based on TF back-end like PANNA. While it is particularly rich in its features for prediction of chemical properties from cheminformatics descriptors, interatomic potential generation for high performance molecular dynamics simulations for solids or interfacing with electronic structure codes routinely used in

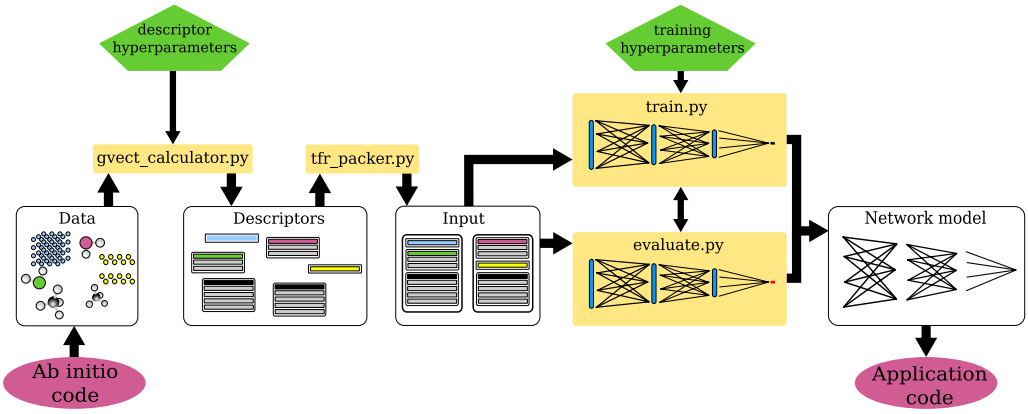

Fig. 1. PANNA workflow. Top layer corresponds to user intervention in the neural network potential generation process. Input and network hyperparameters are determined at this layer. Currently several helper programs (not shown) included in the PANNA package aim to help users in this intervention. In principle, the hyperparameters are also part of the network optimization process and can potentially become an automated part of the workflow. Yellow boxes indicate main programs in PANNA while white boxes stand for the user-owned data in different stages of processing and the network potential. The interface with the third party codes are established via parser scripts and patches (not shown) included in the package. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

condensed matter community such as Quantum ESPRESSO [51] or VASP [52], appear to be out of its current development focus.

In the following sections we describe the details of the implementation, in particular: in Section 2 we detail the neural network training workflow in PANNA; in Section 3 we provide computational details on the implementation, data format and parallelization strategies; in Section 4 we give a usage example; and finally in Section 5 we report the results on two test cases to show the capabilities of the package.

2. Neural network training workflow

Here we summarize a simple neural network training workflow to introduce the notation used hereafter (see Fig. 1). Consider a dataset of examples $D = \{s, \bar{O}\}$ where each example s_i is associated with an observable of interest $\bar{O}_i$. For neural network interatomic potentials, these examples are often made up of atomic positions and the target observables are the corresponding total energies and atomic forces obtained from accurate first principles simulations. The first step of the workflow is to choose how to represent an example as input to the network model. We describe in Section 2.1 the possible representations available in PANNA. Once a representation model is chosen, each simulation s_i is mapped to an input vector $\vec{G}_i$ of length N where N is determined based on the description model and parametrization. The neural network can then be written as a parametrized function that maps input vectors into predictions for the target observable, $O_i = F(\vec{G}_i; W_{\mathrm{NN}})$, where W_{NN} stands for the parameter set. Finding the optimum set of parameters corresponds to approximating the computationally-demanding function that is used to generate the dataset D. The parameter fitting is performed via training of the network to minimize a cost function which is often chosen proportional to the difference between observable values in the data and the network predictions for them, $C(\{\bar{O}_i - O_i\})$.

Currently PANNA implements fully-connected, deep, feed-forward neural networks. All input vectors used in single training session are requested to be of same length, N. The network is composed of L layers, each layer having n_l nodes. Considering the input vector as the zeroth layer, we can define $n_0 = N$. The computation performed at each node can be written in terms of its input, activation function, and output:

$$a_j^{l+1} = g\left(\sum_{i=1}^{n_l} a_i^l W_{ij}^{l+1} + b_j^{l+1}\right) \tag{1}$$

where a_i^l is the ith element of the output from layer l, which is the input vector for node j of layer $l+1$. The activation function for this node is specified by two parameters, weight matrix W^{l+1} and bias scalar b^{l+1}, and a non-linear function g.

The final output of the network $O_i = \vec{a}^L$ is therefore a complex nonlinear function of the input, weights and biases of each layer. If g is partially differentiable with respect to weights and biases, these can be optimized to minimize the cost function C via an optimization algorithm such as gradient descent (see Section 2.3). Once the minimization is performed and a set of parameters that best estimate the observables for the training data – as well as an independent validation data – is found, the network can be used as a predictor for other examples (see Section 2.5).

In the following, we describe the choices readily available in PANNA for each step of this workflow.

2.1. Representation

Currently, two different types of representations are implemented in PANNA: the Behler–Parrinello (BP) descriptor model [10] and a version inspired by a recent modification to it [45]. Both representations produce a fixed-size vector for each atom in the unit cell, describing its local environment up to a user-determined cutoff and with gaussian smoothing.

The cutoff function ensures that for a given atom, only neighboring atoms closer than a cutoff radius R_c have nonzero contribution to the descriptor vector and the contribution decays smoothly with distance. For a pair of atoms i, j that are R_{ij} apart, the cutoff function is:

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

In the implementation of the standard BP representation, the original functional forms introduced in Ref. [10] are used without further modification. The descriptor vectors are made up of two parts: radial and angular. Radial part depends only on the interatomic distances from the central atom i to all neighbors j within the cutoff as:

$$G_i^{\mathrm{Rad}}[s] = \sum_{j \neq i} e^{-\eta\left(R_{ij} - R_s\right)^2} f_c(R_{ij}), \tag{3}$$

where the width η and a set of centers R_s are parameters of the descriptor. For the angular part, all pairs of neighbors j, k in the

cutoff radius that form an angle θ_{ijk} with the central atom i are used to define:

$$G_i^{\mathrm{Ang}}[s] = 2^{1-\zeta_s} \sum_{j,k\neq i} \left(1+\lambda\cos\theta_{ijk}\right)^{\zeta_s} \times e^{-\eta_s\left(R_{ij}^2+R_{ik}^2+R_{jk}^2\right)} \times f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}) \quad (4)$$

where user-defined sets of η_s and ζ_s are the parameters of the descriptor. The λ parameter takes ± 1 in order to generate descriptors that display peak response at 0 and π.

In the modified BP representation, the angular descriptors are modified to contain the radial and angular resolution more explicitly, as inspired by the modification introduced in Ref. [45]. The descriptor used in that work was:

$$G_i^{\mathrm{Ang}}[s] = 2^{1-\zeta} \sum_{j,k\neq i} \left[1+\cos\left(\theta_{ijk}-\theta_s\right)\right]^{\zeta} \times e^{-\eta\left[\frac{1}{2}\left(R_{ij}+R_{ik}\right)-R_s\right]^2} \times f_c(R_{ij}) f_c(R_{ik}) \quad (5)$$

with user-defined parameters η, ζ, set of θ_s and set of R_s as the parameters of the descriptor.

The derivative of the angular descriptor in Eq. (5) is discontinuous with respect to atomic positions when three atoms are arranged collinearly, i.e. $\theta_{ijk} = 0, \pi$. Since this derivative is used in the calculation of forces on each atom via chain rule, the discontinuity may result in considerably erroneous forces, in particular for highly symmetric crystal structures. To address this issue, the angular descriptors are modified with a smoother function where the $\cos(\theta-\theta_s)$ term is replaced with $\cos(\theta)\cos(\theta_s) + \sqrt{1-\cos(\theta)^2+\epsilon}\sin(\theta_s)^2\sin(\theta_s)$ and a normalization factor of $\frac{2}{1+\sqrt{1+\epsilon}\sin(\theta_s)^2}$ is introduced, where ϵ is a small number. The resulting function for this descriptor is thus

$$G_i^{\mathrm{Ang}}[s] = \sum_{j,k\neq i} \left[1+2\frac{\cos(\theta_{ijk})\cos(\theta_s)+\sqrt{1-\cos(\theta_{ijk})^2+\epsilon}\sin(\theta_s)^2\sin(\theta_s)}{1+\sqrt{1+\epsilon}\sin(\theta_s)^2}\right]^{\zeta} \times 2^{1-\zeta} e^{-\eta\left[\frac{1}{2}\left(R_{ij}+R_{ik}\right)-R_s\right]^2} f_c(R_{ij}) f_c(R_{ik}). \quad (6)$$

In case of a system with multiple atomic species, the descriptors are resolved by species, i.e. radial descriptor sum in Eq. (3) is performed only for neighbors belonging to a single species at a time and the total radial descriptor size grows linearly with number of species n_s. Similarly, the sum in angular descriptors are repeated for each possible pair of species for a given central atom, giving rise to a growth by $n_s(n_s+1)/2$ in size of the angular descriptor.

It should be noted that a good representation is fundamental for generating a successful neural network potential. For any representation model chosen, the descriptor parameters should be carefully selected in order to achieve an optimal trade off between richness of the descriptors and the accompanying computational cost during training. PANNA provides visual inspection tools to examine the resolution of the descriptors as well as the suitability of the descriptor parameters for a given dataset (see Section 3).

2.2. Network architecture

PANNA constructs a sub-network for each atomic species such that the output of sub-networks can be combined to obtain a single prediction output for a given atomic configuration input. This is a well-established architecture and several works obtained with this architecture can be found in the literature [40,42,49,53].

In order to address the varying complexity in the environment of different species in a sample, PANNA allows each sub-network to be of different size. Moreover, to enable tighter control of the training dynamics, the option to freeze any layer is also implemented, so that only user-selected parts of the network can be trained. Lastly, a different non-linear activation function g can be chosen for each layer. Currently supported activation functions are:

- Gaussian: $g(x) = \exp(-x^2)$, as demonstrated in Ref. [45].
- ReLU (rectified linear unit): $g(x) = \max(0, x)$.
- Linear: $g(x) = x$
- Radial Basis Function (RBF) that changes the structure of the layer from the one outlined in Eq. (1) into $a_j^{l+1} = \exp\left[-\sum_{i=1}^{n_l}(a_i^l - W_{ij}^{l+1})^2\right]$.

The last layer of the network that results in the final estimate for the observable uses a linear activation in order to sum the partial predictions of each atomic species sub-network.

2.3. Training

The training of the model parameters is performed through a commonly used variant of stochastic gradient descent called Adam [54] which uses the gradients of the cost function with respect to the weights to drive the model parameters towards a local minimum. The stochasticity is a result of the practice that at each minimization step, only a randomly selected subset (or *minibatch*) of the whole training data are used for gradient calculation. The minibatch cost function C_{b} at optimization step t is

$$C_{\mathrm{b}}^{(t)}(W^{(t)}) = \sum_{i\in\mathrm{batch}^{(t)}} C^{(t)}(\bar{O}_i - O_i(W^{(t)})). \quad (7)$$

where the parameters of the model (weights and biases of all layers) at step t are represented with $W^{(t)}$ for brevity.

Adam optimizer updates each parameter $w \in W$ adaptively, i.e. depending on its individual history until optimization step t. It first calculates an estimate for the mean m_w and for the uncentered variance v_w for the current and past gradients of each parameter via an exponential running average:

$$m_w^{(t)} = (1-\beta_1)\nabla_w C^{(t)} + \beta_1 m_w^{(t-1)} \quad (8)$$

$$v_w^{(t)} = (1-\beta_2)|\nabla_w C^{(t)}|^2 + \beta_2 v_w^{(t-1)} \quad (9)$$

where $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the running average. The bigger they are, the more history dependent the optimization becomes. Adam also implements an effective correction in order to avoid the bias due to initial values where there is no history:

$$\hat{m}_w^{(t)} = m_w^{(t)}/(1-(\beta_1)^t) \quad (10)$$

$$\hat{v}_w^{(t)} = v_w^{(t)}/(1-(\beta_2)^t) \quad (11)$$

Then it updates the parameters of the model proportionally with respect to the mean and inversely proportionally to the square root of the variance of their gradient so that parameters whose gradients show large variation in the last few steps are updated more slowly in the coming step and vice versa:

$$w^{(t+1)} = w^{(t)} - \alpha\frac{\hat{m}_w^{(t)}}{\sqrt{\hat{v}_w^{(t)}}+\epsilon} \quad (12)$$

where α is the learning rate and ϵ is a small number for stability. The most important parameters of the Adam optimizer for tuning of the training of a neural network model are α, β_1 and β_2. Note that TensorFlow implementation of Adam differs slightly as in

Eq. (13); resulting in scaling of ϵ. For further details about the Adam algorithm see Ref. [54].

$$w^{(t+1)} = w^{(t)} - \alpha \frac{(1-(\beta_2)^t)}{(1-(\beta_1)^t)} \frac{m_w^{(t)}}{\sqrt{v_w^{(t)}} + \epsilon'}. \tag{13}$$

In addition to the adaptive stepsize nature of Adam, PANNA allows to gradually decrease the learning rate α during training to mimic annealing in the parameter space. The exponential decrease of the learning rate is given by:

$$\alpha^{(t)} = \alpha^{(t=0)} r^{t/\tau}, \tag{14}$$

where $\alpha^{(t)}$ is the decayed learning rate at step t. The decay rate r and the decay step τ are used to determine the decay behavior.

The cost function is an important ingredient of neural network training. In PANNA besides the standard quadratic loss on target observable values (Eq. (15)), it is possible to use an exponential loss function, which weights more strongly the outliers within a batch. To avoid numerical instability at the initial steps of the training when the gradients can be expected to be large, the exponential loss is smoothly clipped to a constant value through the application of a hyperbolic tangent (see Eq. (16)). If desired, cost per atom can be considered rather than cost per data point.

$$C_{\mathrm{b}}^{\mathrm{Q}}(W) = \sum_{i \in \mathrm{batch}} \left(\bar{O}_i - O_i(W)\right)^2. \tag{15}$$

$$C_{\mathrm{b}}^{\mathrm{E}}(W) = \exp\left[a \tanh\left(\frac{1}{a} \sum_{i \in \mathrm{batch}} \left(\frac{\bar{O}_i - O_i(W)}{N_i^{\mathrm{atoms}}}\right)^2\right)\right]. \tag{16}$$

Adding further constraint or information to cost function, i.e. regularization, can help to prevent overfitting. Two commonly used norm-based weight regularizations are supported in PANNA: regularization on the sum of the absolute value (L1 norm) and on the square (L2 norm) of the weights. The relative coefficient for each norm can be set independently. Resulting penalty is added to the total cost in the computation of gradients (Eq. (17)).

$$C_{\mathrm{b}}^{\mathrm{NR}}(W) = c_1 \|W\|_1 + c_2 \|W\|_2 \tag{17}$$

On occasions where parameter space have steep cliffs, gradients can reach large numbers and following them may result in overshooting the low cost region. To prevent this behavior, the absolute value of each gradient can be capped at a user-defined constant before being processed to update the parameters. It should be noted that such gradient clipping based on absolute value corresponds to change of optimization direction in the parameter space.

2.4. Training with forces

When both the target function and its derivatives are known and used in the training, the prediction power of a neural network model can dramatically increase. Since training already requires differentiation with respect to network parameters, additional cost of training for the derivatives can be mitigated with the chain rule. In atomic simulations, this scenario can be realized when the cost function includes both energy and forces as their analytical derivative with respect to atomic positions.

In PANNA, force prediction of the network model is computed analytically as in the following:

$$\vec{F}_i = \sum_{jk} \frac{\partial E_j}{\partial G_{jk}} \frac{\partial G_{jk}}{\partial \vec{x}_i}, \tag{18}$$

where G_{jk} is the kth element of the descriptor for atom j. The first term is already required during training for the update of network parameters (see Eq. (8)), while the second term of partial derivatives with respect to the position of atom i can be pre-computed together with the descriptors (see Section 2.1).

In PANNA, when reference forces are requested to be used in training, an extra term is added to the cost function:

$$C_{\mathrm{b}}^{F} = c_F \sum_{i \in \mathrm{batch}} \sum_{j=1}^{N_{\mathrm{atoms}}} \sum_{k \in \{x,y,z\}} (\bar{F}_{ij}^k - F_{ij}^k)^2, \tag{19}$$

where c_F is a user-defined parameter that allows to tune the weight of the force-based loss in the total cost function.

2.5. Prediction

Once the network is trained, prediction on a new data point can be computed through a single forward evaluation of the network. Unlike training, where powerful libraries are necessary for efficient optimization of parameters and data-handling of large sets of data, the evaluation task only requires matrix multiplication and basic mathematical algebra. This enables PANNA to provide a simple program for the evaluation task, which can be easily coupled with other codes, e.g. MD software that simulates movement of the atoms as directed by the interatomic neural network potential. Currently PANNA supports such interface directly with LAMMPS code [55,56] and indirectly through KIM API [57–59].

3. Implementation details

3.1. PANNA programs

PANNA core is written in Python and is based on the TensorFlow (TF) [60] framework. It is organized as a package of several main programs, each characterized by its own configuration input file (.ini) and parameters. In Section 4 we offer a usage and input example for each of these programs:

- `gvect_calculator.py`
  This code first reads information on the simulation data such as atomic positions and cell parameters, written in simple PANNA-example format, stored as JSON standard. It then computes the descriptor for each atom, with the user-specified hyperparameters, and stores all the descriptors of all atoms in the simulation in a single binary data file.
- `tfr_packer.py`
  This code collects a large number of descriptor binary files and converts them into TFRecord (TFR) format which is ready to be efficiently processed by TF. The resulting files, called TFData within PANNA, reduce the I/O overhead, simplify and speed up the dataset management during training.
- `train.py`
  This is the main routine that performs the training: it reads the TFData files, handles the queue management to supply parallel processing of minibatches and drives the training procedure with the appropriate TF calls. Information required to restart the calculation is stored as "checkpoints" during training in TF format at user-defined intervals. Additionally, the summary of each training is stored in TF "event" files that can be visualized in TensorBoard [60] (see Section 3.4).
- `evaluate.py`
  Predictions are made using this code, which can parse the checkpoints, access the parameters of the network and evaluate the network prediction for a given input configuration. It can operate on single binary descriptor files, such as the outputs of `gvect_calculator.py` or bundled TFData files

such as the outputs of `tfr_packer.py`. It can operate on a single file or on all files of a directory as specified by user input.

A number of tools are also provided to handle data processing and visualization. Just to name a few: parsing of Quantum ESPRESSO output XML files [51], of extended xyz format as in Ref. [46], or of ANI [45] dataset are done by `qe_parser.py`, `exyz_parser.py`, `ani_parser.py` respectively. These tools convert the data into PANNA-example files in JSON standard. Multiple such example files can be converted into an XCrySDen [61] animation by `json2axsf.py` for visual inspection of the data. The `gvect_param_check.py` code plots the characteristic length scales of a descriptor to help tune the radial and angular resolution parameters. Given a set of PANNA-example files and descriptor hyperparameters, `gvect_writer.py` calculates the average descriptor for each species, which may prove useful in recognizing the optimum parameter range for a specific set of data. Once the training is over and a network model is decided to be used in further applications, `extract_weights.py` program converts the network architecture and weights from tensors in TF to PANNA, LAMMPS or KIM compatible format.

3.2. TensorFlow

TensorFlow (TF), the underlying engine for data management and neural network training used in PANNA, is an open source machine learning library released by Google [60]. In TF, a computation is described by a directed graph that represents values flowing between nodes. Each node represents an instantiation of an operation, such as matrix multiplication, and has zero or more inputs and outputs. Some nodes are allowed to maintain and update their persistent states, to enable branching or looping as needed to represent the computation. In dataflow programming, movement of data is emphasized: a node operation runs as soon as the inputs become valid, making this programming paradigm suitable for distributed parallel execution. In TF all data are treated as n-dimensional arrays, i.e. tensors.

Due to its dataflow programming basis, TF is particularly well optimized for management of large datasets and training of large networks. Its stateful queue operations support advanced form of coordination of access to data. It allows a different access for pre-fetching and pre-processing of training examples, for shuffling them, and for consuming them during training, allowing user to fine tune the dataflow in their application. Several such performance related TF variables are exposed in PANNA through input keywords such as `shuffle_buffer_size_multiplier` which sets how many minibatches of data are to be accessed for shuffling. Another of such examples is the `dataset_cache` boolean that is used to determine whether to cache the TF input dataset during training for increased performance, noting that TF dataset is a complex collection of elements including the data but also possible operation objects that act on the data such as iterators.

3.3. Parallelization and hardware

TensorFlow and PANNA performance can be controlled by two main variables: `intra_op_parallelism_threads` and `inter_op_parallelism_threads`. The `intra_op` keyword sets the parallelism inside an operation, e.g. when a matrix multiplication is performed; while `inter_op` controls the parallelism of operations that can be carried out concurrently. PANNA exposes these variables with the same input keywords as TF. While the optimum number of threads for intra operation parallelization can be expected to be in the same order as other codes that perform matrix manipulation – and users can benefit from benchmarks based on `$OMP_NUM_THREAD` as the parallelization variable – the inter operation parallelism is largely affected by the network size and topology, so that its optimization per training scenario is advised.

Thanks to the underlying TF engine, PANNA can run on CPU, GPU and TPU systems. An example is provided in the documentation to demonstrate how to run PANNA with intra- and inter-node parallelization on multiple CPU nodes of a High Performance Computing (HPC) system.

Each operation of TF resides on a particular device and task, e.g. `Stitch` operation that reassembles partial results may be required for a *Parameter Server* task that contains one CPU device, or `Add` operation for tensors of a *Worker* task that contains two devices, one of which is the same of the parameter server task before, and one GPU device. TF first places the operations in a graph to devices, then partitions all operations of a device into subgraphs that can be cached, and manages communication between devices via `Send` and `Recv` operations that are specialized and optimized for several device-type pairs. PANNA modifies the device-based parallelization of TF such that every worker task executes the same neural network training graph independently. During training, workers read the network parameters and copy them internally, process a minibatch of data and calculate the gradients required for optimization, then send the gradients to the parameter server that updates the single shared instance of the network parameters independently. Therefore the weights and biases of the network are updated asynchronously, reading and updating of different workers happen concurrently, making asynchronous training free from read–write lock, enabling high scalability across nodes. It should be noted that asynchronous update introduces further stochasticity to the training procedure since at any moment network parameters are allowed to have different time stamps [62].

3.4. Visualization

As the NN training involves a stochastic process with many parameters and figures of merit, tools that are easy to use for monitoring training dynamics are highly desirable.

TensorBoard (TB), a TF-native visualization tool, accesses log files created at training time and visualizes them through a browser interface. In PANNA, a number of quantities that are relevant for NN training are saved in the log files such as total/per atom/root-mean-square of cost function per minibatch, loss due to L1 and L2 penalties, species-resolved atomic contribution to network prediction, the distribution of weights and biases for each layer, and the learning rate. See Fig. 2 for a sample view of elements from TB dashboard during training.

TB also includes dimensionality reduction tools such as Principal Component Analysis [63] and t-Stochastic Neighbor Embedding [64] on the network parameters and allows the results to be visually analyzed.

4. Usage example

In this section we describe a simple but complete use case of PANNA, from the atomistic simulation information to obtaining an NN interatomic potential ready to be employed in MD simulations. Simple input scripts are provided. Further details can be found in the tutorials distributed within the package.

4.1. Data preparation

Let us assume to have ab initio simulation results for water molecule obtained by Quantum ESPRESSO [51]. This package

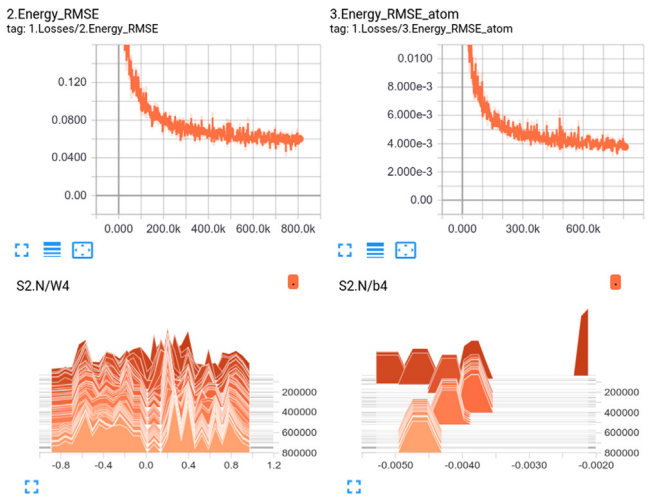

Fig. 2. Example of visualization in TensorBoard. **Top:** Scalars such as loss per atom, regularization loss, or learning rate can be tracked as the training continues. **Bottom:** Distribution of weights and biases of each layer and each atomic species as a function of optimization step can be observed. Such observations can give insight on when to try training strategies such as (un)freezing layers.

outputs in XML format. A relevant snippet of the XML file is as follows:

```
{
 <control_variables>
    <prefix>H2O_3e45a0e332</prefix>
    ...
 </control_variables>
 ...
 <atomic_structure nat="3" alat="3.779452265771e1"
 bravais_index="1">
    <atomic_positions>
     <atom name="O" index="1">-3.103910095932729e-18
     1.972216586331875e-3 1.829760905993973e-1</atom>
     <atom name="H" index="2">3.739048037144415e-17
     1.613226163461667e0 -5.580297760571020e-1</atom>
     <atom name="H" index="3">1.187082814066741e-17
     -1.644526707907135e0 -5.824461493348307e-1</atom>
    </atomic_positions>
    <cell>
     <a1>3.779452265771287e1 0.000000000000000e0
     0.000000000000000e0</a1>
     <a2>0.000000000000000e0 3.779452265771287e1
     0.000000000000000e0</a2>
     <a3>0.000000000000000e0 0.000000000000000e0
     3.779452265771287e1</a3>
    </cell>
  </atomic_structure>
  ...
  <total_energy>
   <etot>-2.200687664809398e1</etot>
   ...
  </total_energy>
}
```

Using `qe_parser.py`, the relevant information, i.e. atomic positions, cell parameters and total energy, is parsed and stored in a JSON standard for neural network training:

```
python3 panna/tools/qe_parser.py
          -i /path/to/QE_XML_FILES
          -o /path/to/PANNA_JSON_FILES
          --addhash
```

where input and output directories are used to process multiple files at once. The resulting JSON file is as follows:

```
{
 "atomic_position_unit": "cartesian",
 "lattice_vectors":
  [[37.79452265771287, 0.0, 0.0],
  [0.0, 37.79452265771287, 0.0],
  [0.0, 0.0, 37.79452265771287]],
 "energy": [-22.00687664809398, "Ha"],
 "atoms":
  [[1, "O",
   [-3.103910095932729e-18, 0.001972216586331875,
   0.1829760905993973],
   [-1.811290045808234e-06, -0.02306295358423087,
   0.05599605952219937]],
  [2, "H",
   [3.739048037144415e-17, 1.613226163461667,
   -0.558029776057102],
   [9.212452357485834e-07, 0.008691667402484345,
   -0.033636613232433991]],
  [3, "H",
   [1.187082814066741e-17, -1.644526707907135,
   -0.5824461493348307],
   [8.900448100596506e-07, 0.01437128618174652,
   -0.02235944628976539]]],
 "key": "H2O_3e45a0e332",
 "unit_of_length": "bohr"
}
```

When processing many *ab initio* simulation at once, JSON files should be named carefully to avoid overwriting. The `--addhash` argument ensures that even when ab initio filenames are the same, e.g. the QE default `prefix.xml`, JSON files are uniquely named using a hash function. Without this argument, the names of the XML files are kept for JSON files.

4.2. Computation of descriptors

Computation of the descriptors for each example can be achieved as the following:

```
python3 panna/gvect_calculator.py --config gvect_input.ini
```

where a sample configuration file might look like the following:

```
[IO_INFORMATION]
input_json_dir = ./simulations
output_gvect_dir = ./gvectors
log_dir = .

[SYMMETRY_FUNCTION]
type = mBP
species = H, O

[PARALLELIZATION]
number_of_process = 4

[GVECT_PARAMETERS]
gvect_parameters_unit = angstrom
eta_rad = 16
Rc_rad = 4.6
Rs0_rad = 0.5
```

```
RsN_rad = 16
eta_ang = 6.0
zeta = 50.0
Rc_ang = 3.1
Rs0_ang = 0.5
RsN_ang = 4
ThetasN = 8
```

Here, the first section indicates the location of input and output directories, the second specifies the type of descriptors and atomic species, the third how many parallel processes to use and the fourth the hyperparameters used to create the descriptors.

In this case, modified BP descriptors will be created according to Eqs. (3) and (6). The radial descriptor is made up of `RsN_rad` Gaussian centers equally spaced between `Rs0_rad` and `Rc_rad`. A similar relationship holds for the radial binning of the angular descriptor. For the angular binning, the centers are equally spaced in $[0, \pi]$ window, shifted to align with the midpoint of the interval. Considering the 2 species, this sample setting amounts to 16×2 radial components and $(4 \times 8) \times 3$ angular ones, i.e. a descriptor array of length 128 for each atom.

The descriptor of each atom of the system is then concatenated to produce the descriptor of the simulation and written in binary format alongside the reference energy.

4.3. Packing of data

The large number of binary files produced in the previous step is packed into a small number of large files to increase I/O efficiency. This is achieved by the following command which yields binary files with .tfr extension.

```
python3 panna/tfr_packer.py --config tfr_sample.ini
```

with a sample input as in the following:

```
[IO_INFORMATION]
input_dir = ./gvectors
output_dir = ./tfr
elements_per_file = 1000
prefix = train

[CONTENT_INFORMATION]
n_species = 2
```

4.4. Training

A common practice is to divide the data into training and validation sets. Since in the previous step, we have packed all the data into self-contained .tfr files, this division can be performed simply by moving files into different directories. When the training set is created as such, the training process can begin. This can be achieved by the following program:

```
python3 panna/train.py --config train.ini
```

A minimum required input file for this program is as follows:

```
[IO_INFORMATION]
data_dir = ./tfr_train
train_dir = ./train
log_frequency = 10
save_checkpoint_steps = 500

[DATA_INFORMATION]
atomic_sequence = H, O
output_offset = -13.62, -2041.84

[TRAINING_PARAMETERS]
batch_size = 50
learning_rate = 0.01
max_steps = 5000

[DEFAULT_NETWORK]
g_size = 128
architecture = 64:32:1
trainable = 1:1:1

[H]
architecture = 32:1
trainable = 1:1
activations = ReLU:Linear
```

In the first section the necessary information on I/O paths and logging frequency is specified. In the second, the atomic species sequence used to build the descriptors as well as a reference energy for each species is given. The reference energy listed here will be subtracted from the total energy of any data point before the data are presented to the neural network. Even though this operation only corresponds to a trivial shift of bias, hence the network can be expected to learn its value during training; because the values of the network parameters are initialized around zero, explicitly applying the shift is found to speed up the training process. The third section contains information about the specifics of the training: the size of the minibatch, the learning rate to use (constant in this case) and how many training steps to perform before stopping.

In the following section a default network model is defined. This is the model that would be assumed by default for each species unless further information is provided. The size of the input descriptor `g_size` $= 128$ and the architecture (e.g., two hidden and one final output layer) are specified. The activation function is by default Gaussian for all the hidden layers, and all layers will be allowed to change during training. Finally the last section allows the user to define a different network model for a species of his/her choosing. This feature can be particularly useful for fine tuning the network in order to achieve a compact, low cost model for several species.

At any time during or after the training, we could inspect the evolution of the observables by using TB:

```
tensorboard --logdir=./train
```

4.5. Validation of the model

The performance of a neural network model in terms of accuracy can be estimated through the evaluation of the final model on data that has not been included in the training set. This can be achieved with the following program:

```
python3 panna/evaluate.py --config validation.ini
```

and sample input file:

```
[IO_INFORMATION]
data_dir = ./tfr_validate
train_dir = ./train
eval_dir = ./validate

[TFR_STRUCTURE]
g_size = 128

[VALIDATION_OPTIONS]
single_step = True
```

Table 1
The size of datasets obtained from Ref. [65] used for training and validation. As the dataset is constructed by sampling the normal modes of each molecule, alongside a scaling factor to reduce the bias towards bigger molecules, it contains a different amount of data for each molecule type, e.g. 480 examples for N_2 and 17 280 for C_4H_{10} (butane) and 340 for C_8H_{18}(octane). The final models are benchmarked against 10 347 configurations from normal mode sampling of 138 molecules from the GDB-11 [67] with 10 heavy atoms, also included in Ref. [65]..

Label	Max # of heavy atoms	# of elements in training set [$\times 10^6$]	# of elements in validation set [$\times 10^6$]
DSmax4	4	0.656	0.134
DSmax6	6	3.432	0.427
DSmax8	8	17.476	2.182

The locations of the validation data, train files, i.e. weights and biases, and validation output are given in the first section. The second section specifies the network-related properties of the validation data, which is expected to be coherent with the specifications used during training.

This program produces a simple text file with reference and predicted output for all the simulations in the validation dataset, and can be used to assess the quality of the model.

5. Results

In this section we detail two example studies that demonstrate the usability of PANNA for periodic and aperiodic systems, with varying amount and quality of data. Finally we also demonstrate the energy conservation of the final network model in an MD scenario.

5.1. Molecules

Here we report training of the network architecture previously used in Ref. [45]. With half a million parameters, this is a larger network than the average architecture employed in the literature. By reproducing the results of Ref. [45], we demonstrate that the training and testing modules of PANNA can answer the high performance demand scenarios in machine learning of interatomic potentials.

The dataset [65] contains 57 462 small organic molecules consisting of H, C, N and O atoms, with up to 8 heavy atoms and corresponding total energy calculated via Density Functional Theory (DFT) [66]. Training is performed for 3 different datasets, labeled as DSmax4, DSmax6 and DSmax8, each including data from molecules with up to 4, 6, or 8 heavy atoms respectively. Table 1 shows the size of datasets used for training and validation.

The modified Behler–Parrinello symmetry functions described in Section 2.1 are used as in the original reference, with 32 Gaussian centers for the radial part, and 8 angular and 8 radial centers for the angular part. Considering the 4 species in the dataset, this results in descriptors of size 768 for each atom. The atomic network architecture consists of 3 hidden layers of sizes 128, 128 and 64, all with Gaussian activation function, followed by a linear activation layer.

While the originally proposed cost function is proportional to the exponential of the square loss, it is found to yield very large gradients in the early stages of the training causing numerical instability. The original reference addresses this by weights norm clipping. In this study a simple quadratic loss in combination with the capped exponential loss described in Eq. (16) is found to alleviate the problem while still preserving increased gradients on the outliers. An initial learning rate of 0.001 with a decay rate $r = 0.98$ and decay step $\tau = 3200$ is used following Eq. (14). In the case of training with DSmax8, learning rate decay step is increased to $\tau = 16000$, leading to a slower decay. A fixed batch size of 1024 examples is used.

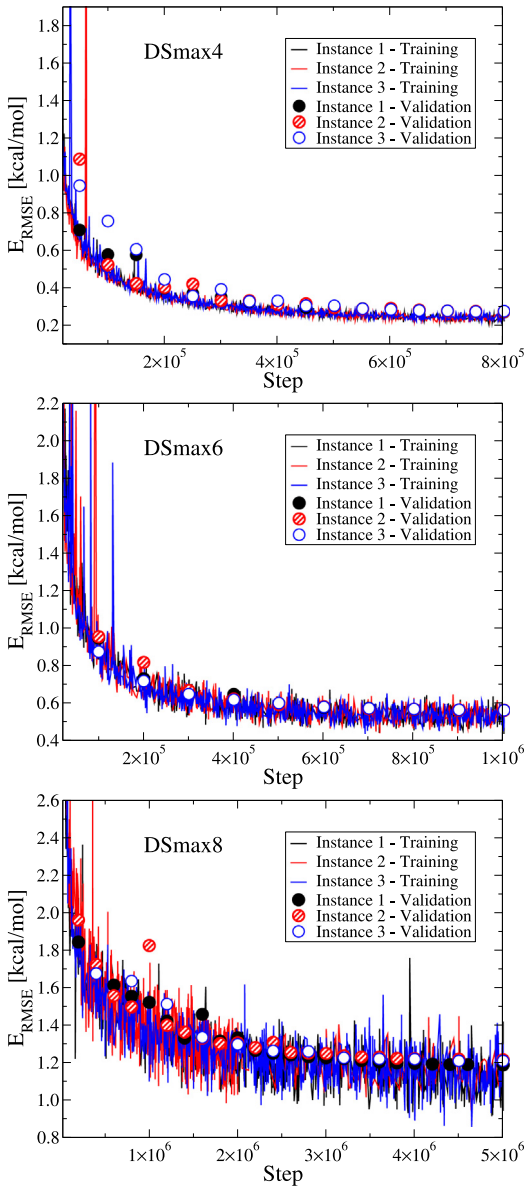

Fig. 3. The energy RMSE during training and on validation set as a function of optimization steps for trainings with DSmax4 (top), DSmax6 (middle) and DSmax8 datasets (bottom). For each dataset, three instances of training are performed starting from different random initial parameters. The RMSE calculated at the final step for training (validation) set are 0.24 (0.27), 0.24 (0.28) and 0.24 (0.28) ; 0.54 (0.57), 0.54 (0.57) and 0.53 (0.57); and 1.12 (1.19), 1.18 (1.22) and 1.14 (1.21) in kcal/mol for trainings with DSmax4, DSmax6 and DSmax8 respectively. For comparison, the results from Ref. [45] are 1.16 (1.28) kcal/mol training (validation) RMSE for training with DSmax8.

Fig. 3 shows the evolution of root mean square error (RMSE) on training and validation sets during training. For each dataset, three networks with identical architecture but different random seeds are trained. It can be seen that the proposed training schedule yields quantitatively reproducible results, which are consistently in good agreement with those reported in Ref. [45].

As it may be expected, the bigger the training dataset gets in variability, going from DSmax4 to DSmax8, the harder it gets to solve the regression problem successfully in few optimization steps. Hence the training and validation errors increase steadily from approximately 0.3 to 1.2 kcal/mol per example, while the

Table 2
RMSE of energy prediction in kcal/mol for networks trained and tested with datasets of various molecular complexities. In the first five columns the RMSE is calculated for configurations where the total energy is within $E_{cut} = 275$ kcal/mol of the lowest energy configuration for each molecule. $DS\langle N\rangle$ stands for dataset with molecules having $\langle N\rangle$ heavy atoms. Additionally for the 10 heavy atom set, the RMSE for all configurations independent of their energies, and the relative RMSE for configurations within the lowest $E_{cut} = 300$ kcal/mol window is also given respectively. The last two columns contain the directly comparable values.

Training set	Test set							
	DS5	DS6	DS7	DS8	DS10	DS10 all	DS10 300	Ref. [45]
DSmax4	17.1	23.2	28.3	30.0	24.5	139.2	21.0	26.0
DSmax6	0.5	0.7	13.5	15.5	14.5	138.5	15.5	17.7
DSmax8	0.6	0.7	1.2	1.4	2.1	87.7	2.0	1.8

optimization steps required go from approximately one to five million steps.

The arithmetic average of different instances of trainings can be used to make committee predictions for each data point. The validation set RMSE resulting from such committees are 0.25, 0.52, 1.14 for DSmax4, DSmax6, DSmax8 respectively. Note that these values are very close to best individual network prediction errors 0.27, 0.57, 1.19 respectively (see also Fig. 3), hinting that energy prediction error of different networks for each example may be highly correlated. Single network predictions will be reported in the rest of this section.

While the above analysis is based on training and validation sets of similar complexity, the value of neural network potentials can be better judged based on their transferability. To assess this property, we test networks on test sets of varying complexity. For example, a network trained on DSmax4 is tested on molecules with 5, 6, 7, 8 and 10 heavy atoms. In order to compare our results with Ref. [45] in the case of 10 heavy atom test set, the RMSE is also calculated with respect to the lowest energy structure for each molecule. RMSE calculated this way is referred as relative RMSE. The summary of results is reported in Table 2. It can be seen from the energy-capped vs uncapped results that transferability is increased when considering only the low energy structures, and the overall network performance reduces as the training and test sets become more dissimilar.

Comparing the prediction of a network trained on DS8max with the DFT results per individual simulations shows that the error is larger for higher energy configurations (see Fig. 4). The distribution of error shows exponential decay for small error region with a visibly fat tail. Further investigations also show that majority of the outlier configurations belong to a single molecule, hinting that careful error analysis beyond RMSE may be required for judging quality of network potentials.

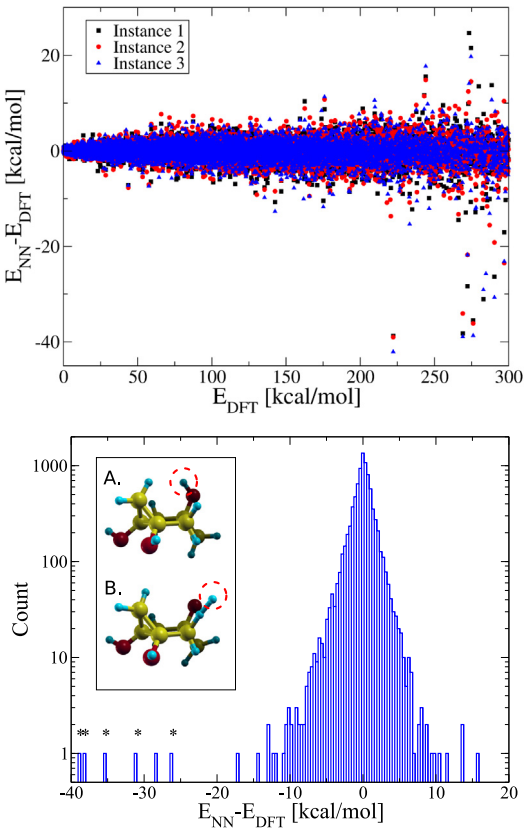

Fig. 4. Top: Neural network prediction error compared to DFT energy on GDB-11 set with 10 heavy atoms where total energy is shifted so that the lowest energy structure of each molecule in the dataset corresponds to 0 kcal/mol. The networks are trained on DSmax8 set (see Fig. 3, bottom panel). **Bottom:** Prediction error distribution in log scale for Instance 2. It is noteworthy that all the marked outliers correspond to configurations of a single molecule shown in the inset where the H atom bound to O in the ground state configuration (A) is displaced far away, and may even form a bond with another H atom (B). Without the marked outliers, the RMSE reduces to 1.8 kcal/mol from 2.0 kcal/mol.

5.2. Silicon

In this example a neural network potential is trained to reproduce the energies and forces of Silicon in solid and liquid phases. While in the previous example the target output is obtained with DFT, here energy and forces are obtained using empirical Stillinger–Weber (SW) potential with the original parametrization [68]. Hence the target energy function that neural network is trained to approximate is indeed a simple function of atomic positions up to three body interactions.

The dataset consists of MD simulation snapshots at every 76.2 fs after the equilibration period, in microcanonical (NVE) ensemble, for 216 Si atoms in cubic box. The 90% of the data is obtained with simulations at constant volume V_0, at a density corresponding to the liquid phase of Silicon at melting temperature, where lattice parameter is 16.053 Å. To sample the solid phase, atomic positions are initialized in cubic diamond phase and velocity of each atom is randomly chosen from a normal distribution that corresponds to a given initial instantaneous temperature. Approximately 100 independent MD simulations with equilibrium temperatures between 1 K and 2500 K are performed this way. To efficiently sample atomic environments corresponding to the liquid phase at different thermodynamic temperatures, first a set of atomic positions corresponding to the liquid radial distribution function is established via melting. This configuration is then used as the initial atomic configuration for approximately 100 MD simulations with equilibrium temperatures ranging between 1000 K and 5000 K.

To sample the effect of lattice parameter on energy, additional molecular dynamics simulations at volumes equal to $\pm 10\%$ and $\pm 5\%$ of the previously fixed cell volume are performed with average equilibrium kinetic energies compatible with temperatures of 3 K, 300 K and 3000 K. Configurations from these MD simulations with various cell dimensions make up the remaining 10% of the dataset. The final dataset gathered in this fashion is composed of 10000 configurations and is split in 80% and 20% parts for training and validation purposes respectively.

Modified Behler–Parrinello type descriptors with radial and angular windows of $0.5 - 4.6$ Å and $1.5 - 4.6$ Å are generated. This cutoff radius is chosen conservatively larger than the SW interaction cutoff 3.8 Å, since the descriptor is only sensitive to the average bond length in an angle (see Eq. (6)). The descriptors included 15 Gaussian centers for the radial part, and 4 radial,

8 angular centers for the angular part, resulting in descriptors of size 48 for each atom. For the remaining parameters in Eqs. (3) and (6), the following choice is made: $\eta_{\text{rad}} = 16.0$ Å^{-2}, $\eta_{\text{ang}} = 6.0$ Å^{-2}, $\zeta = 50$. In order to obtain the predicted forces analytically, derivative of the descriptors with respect to each atomic position is also calculated and stored.

An all-to-all connected network with two hidden layers of sizes 32 and 16, both with Gaussian activation, is constructed. This network structure has 2113 parameters which are optimized with Adam algorithm in order to minimize the sum of quadratic loss functions, Eqs. (15) and (19), using energy and forces respectively.

First, the network parameters are trained using a loss function based on energy predictions alone, by setting the coefficient of the loss on forces, c_F, to zero. A batch size of 256 simulations is used and the learning rate is decayed exponentially starting from 0.01 with a decay rated of 5000 steps, for 57 000 steps. The energy RMSE on the validation set is 2.9 meV/atom with about 9% of predictions having over 5 meV/atom error, and with a maximum absolute error of 16.1 meV/atom, in the same range of accuracy with results obtained modeling similar problems [10,53]. The force prediction of this network however is not accurate enough to perform reliable MD simulations, as the RMSE error on each force component calculated for the validations set is 0.295 eV/Å. The evolution of prediction error during training is summarized in Fig. 5.

In order to better approximate the potential energy surface and obtain more accurate forces, training is performed with finite c_F. It is found that for the system size considered, values for $c_F \approx 1$ result in substantial reduction of error on force components. The result for $c_F = 1.3$ is reported in the rest of this study. Since the inclusion of forces increases the required computation and memory for each step, a small batch size of 64 simulations is chosen. The learning rate decay rate is increased to 22 000 steps to compensate for the smaller batch size and the additional loss.

Using forces in guiding the parameter optimization dramatically increases the information about the target function the network is trained to approximate, hence lifts off possible degeneracy of network parameters. Yet, it can be seen in Fig. 5 that the accuracy on energy can be retained despite the penalty due to derivative matching, i.e. accurate energies and derivatives can be achieved simultaneously. The RMSE error in energy prediction calculated for the validation set is 2.7 meV/atom with about 9% of predictions having over 5 meV/atom error, and with a maximum absolute error of 12.9 meV/atom. The RMSE for prediction of force components instead is 0.134 eV/Å, with only 4% of the validation dataset predicted with error beyond 0.3 eV/Å, and with a maximum absolute error of 10.7 eV/Å. These prediction errors are similar to what is achieved in recent machine learning studies where forces were used in optimization of the machine learning parameters [69,70].

Accurate forces that correspond to derivatives of energy enable conservation of total energy during MD simulations. An NVE simulation with average kinetic energy corresponding to approximately 500 K is performed with time step 0.381 fs for 38.1 ps. It can be seen in Fig. 6 that the conservation of energy is observed for the neural network potential to the same order of an equivalent SW simulation. When the average kinetic energy is increased to explore the liquid phase during the MD simulation, it is seen that the overall prediction error increases yet the conservation of energy is still retained.

The pair correlation function observed during an NVE MD simulation is compared for solid (500 K) and liquid phases (2700 K) (see Fig. 7). The prediction of the neural network potential matches the SW result well even for high temperature.

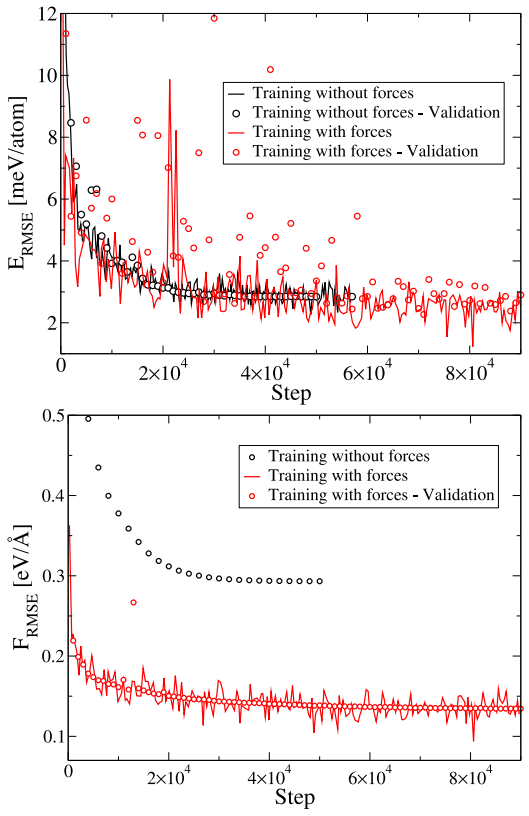

Fig. 5. Top: RMSE of energy prediction on training batch (solid lines) and validation set (points). Training without loss from force predictions, $c_F = 0$, uses a bigger batch size and quicker learning rate decay, resulting in a smoother training convergence behavior (black). The training converges more slowly when loss due to force prediction is considered with $c_F = 1.3$ (red). The smaller batch size and slower decay contributes to slower convergence with increased stochasticity. **Bottom:** RMSE of force prediction for each cartesian component on training batch (solid lines) and validation set (points). Note that early in the training, the total cost is dominated by the loss due to error on forces.

Table 3
Elastic constants in GPa for Silicon in diamond phase at 0 K calculated in this work for neural network potential (NN) and Stillinger–Weber potential (SW), and compared to experimental results from the literature. Note that network is trained to approximate the SW potential, which predicts C_{12} and C_{44} with limited accuracy with respect to the experiment.

	NN	SW	Experiment (4.2 K) [71]
C_{11}	148.4	151.4	167.5
C_{12}	75.5	76.4	64.9
C_{44}	56.8	56.4	80.2

Lastly, we assess the accuracy of predicted mechanical properties. Equation of state predicted with neural network potential shows good agreement with the one of SW for the volume range included in the training set, and it also performs reasonably well for the extended ranges (see Fig. 8). The bulk modulus obtained via Birch–Murnaghan equation of state fit is 105.5 GPa and 101.4 GPa in the case of neural network potential and SW respectively. Further cell deformations lead to the estimate of the other elastic moduli, as reported in Table 3. The neural network potential reproduces the SW results within 2% accuracy.

6. Conclusion

PANNA package offers a complete pipeline for the training of neural network potentials to be used in material science applications. It tackles preprocessing and management of external data

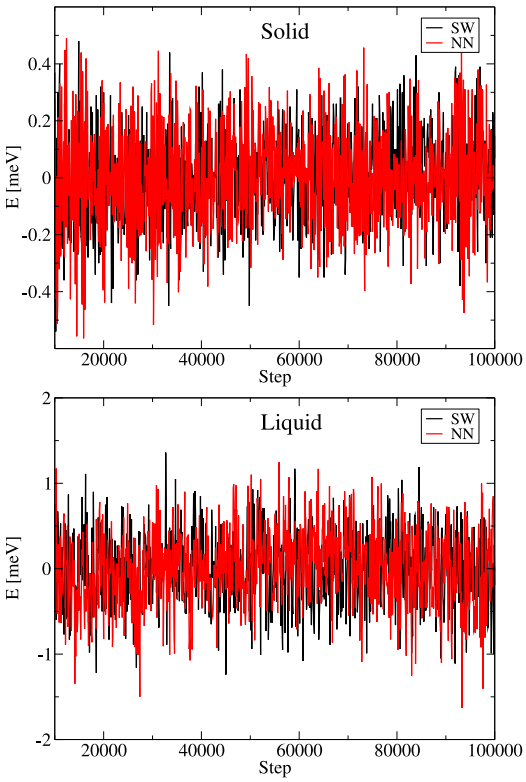

Fig. 6. Fluctuations in the total energy of the simulation of 216 Si atoms as a function of step during an NVE MD with a time step of 0.381 fs. **Top:** The diamond phase at an average temperature of approximately 500 K. Average energy difference of 77 meV between NN and SW potentials, ($E_{NN} > E_{SW}$), is removed for clarity. **Bottom:** Liquid phase at an average temperature of 1800 K. Average energy difference of 954 meV ($E_{SW} > E_{NN}$) is removed for clarity. This difference is considerably higher for liquid phase compared to the solid, indicating the increased discrepancy between the neural network prediction and SW potential for the high temperature phase. Despite this trend, conservation of energy is preserved due to analytically calculated forces.

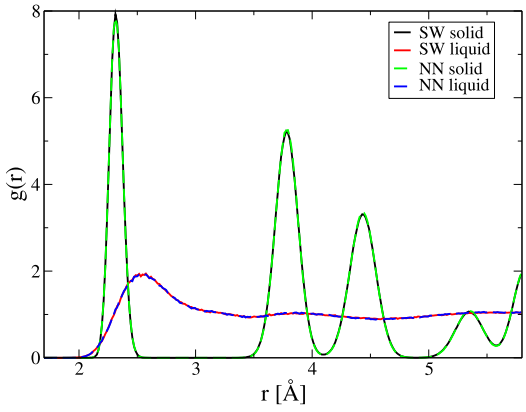

Fig. 7. Solid and liquid radial distribution function at 500 K and 2700 K, respectively, computed during an NVE MD, for SW potential and the neural network potential. Although the accuracy in neural network energy prediction is lower for the high temperature phase as seen in Fig. 6, the radial distribution function is well reproduced.

from diverse sources, training and validation of neural network models based on them, and ultimately conversion of these models to a format compatible with multiple molecular dynamics codes.

PANNA offers flexibility in the definition of model architecture in terms of number of layers and their sizes, activation functions and regularizations. Considering the wide range of materials and

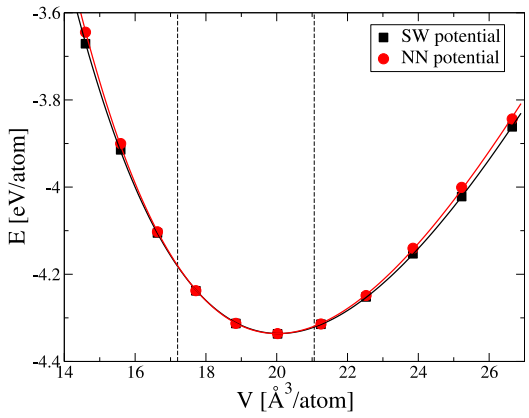

Fig. 8. Energy as a function of volume as obtained with the SW and the NN potentials. The lines are the Birch-Murnaghan equation of state fit to the data for each potential. The range of volume of the training data for the NN is shown with the dashed lines. The performance of the NN outside this range indicates its ability to extrapolate beyond the training set.

structures considered in material science, such flexibility can be desirable. PANNA also offers the possibility to include the derivatives in the training, e.g. total energy and forces. In the simple example of neural network approximation of Stillinger–Weber potential, it is observed that aiming for the correct forces greatly improves the quality of the network model, while retaining the accuracy of its prediction for energy.

Thanks to the TensorFlow back-end, PANNA can handle large networks and datasets, as demonstrated in the reproduction of state of the art training results from literature. At the same time, the simple input file interface to TensorFlow provides easy access to the features of this powerful neural network engine.

Machine learning interatomic potentials is a rapidly growing field. Therefore, in time, addition of new methodologies, network architectures and descriptors are expected to take place as a part of maintenance of the package. This potential growth of the field makes developer-friendliness of a package as important as the one for users. PANNA implementation is based on the idea that BP topology, as some more novel architectures including autoencoders [72] and convolutional networks [73], are a directed acyclic graph with layers. Hence, a network of any of these types of architectures can be built as a scaffold of layers upon defining the input and output rules of each layer separately. PANNA adapts this approach by using the Scaffold class of TensorFlow so that each novel architecture can be implemented independently with minimal change required to the existing methods of the package, reducing the innovation barrier for developers.

As the user-base of the package increases, the input parsing and potential generating routines are planned to include a larger number of codes. In hosting an open source package, and enabling deposition of each model to an open online database, PANNA aims to provide a platform for users and developers that support the reproducibility efforts in computational material science.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors are thankful to Alexie Kolpak, Asegun Henry and Spencer Wyant for an inspirational introduction to the universe

of interatomic potentials and machine learning; to Stefano de Gironcoli for fruitful technical discussions, in particular for the smooth angular descriptor form that prevents discontinuous force derivative during dynamics; and to Qingjie Li and Ju Li for their feedback as the early adopters of PANNA. E.K. is grateful for the financial support by European Union's Horizon 2020 research and innovation program under grant agreement No. 676531 (project E-CAM) and for DOE BES Award No. de-sc0019300. E.K. and F.P. acknowledge the MIT-FVG funds awarded in the 2017 MISTI (MIT International Science and Technology Initiatives) call by Regione Friuli Venezia Giulia that accelerated this work considerably. High performance calculations were carried out thanks to resources provided by CINECA and SISSA. This work also used computational resources at the Extreme Science and Engineering Discovery Environment (XSEDE) [74], specifically Stampede2 at TACC through allocation TG-DMR120073.

References

[1] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, O.A. von Lilienfeld, New J. Phys. 15 (9) (2013) 095003, http://dx.doi.org/10.1088/1367-2630/15/9/095003.
[2] M. Hirn, S. Mallat, N. Poilvert, Multiscale Model. Simul. 15 (2) (2017) 827–863, http://dx.doi.org/10.1137/16M1075454, arXiv:https://doi.org/10.1137/16M1075454.
[3] K. Yao, J.E. Herr, D. Toth, R. Mckintyre, J. Parkhill, Chem. Sci. 9 (2018) 2261–2269, http://dx.doi.org/10.1039/C7SC04934J.
[4] F. Musil, S. De, J. Yang, J.E. Campbell, G.M. Day, M. Ceriotti, Chem. Sci. 9 (2018) 1289–1300, http://dx.doi.org/10.1039/C7SC04665K.
[5] A. Chandrasekaran, D. Kamal, R. Batra, C. Kim, L. Chen, R. Ramprasad, Nano Lett. 5 (1) (2019) 22, http://dx.doi.org/10.1038/s41524-019-0162-7.
[6] K.T. Schütt, H. Glawe, F. Brockherde, A. Sanna, K.R. Müller, E.K.U. Gross, Phys. Rev. B 89 (2014) 205118, http://dx.doi.org/10.1103/PhysRevB.89.205118.
[7] O. Isayev, C. Oses, C. Toher, E. Gossett, S. Curtarolo, A. Tropsha, Nature Commun. 8 (2017) 15679.
[8] A.C. Rajan, A. Mishra, S. Satsangi, R. Vaish, H. Mizuseki, K.-R. Lee, A.K. Singh, Chem. Mater. 30 (12) (2018) 4031–4038, http://dx.doi.org/10.1021/acs.chemmater.8b00686.
[9] H. Wang, L. Zhang, J. Han, W. E, Comput. Phys. Comm. 228 (2018) 178–184.
[10] J. Behler, M. Parrinello, Phy. Rev. Lett. 98 (2007) 146401.
[11] A.P. Bartók, R. Kondor, G. Csányi, Phys. Rev. B 87 (2013) 184115, http://dx.doi.org/10.1103/PhysRevB.87.184115.
[12] M. Gastegger, L. Schwiedrzik, M. Bittermann, F. Berzsenyi, P. Marquetand, J. Chem. Phys. 148 (24) (2018) 241709, http://dx.doi.org/10.1063/1.5019667.
[13] T. Xie, J.C. Grossman, Phys. Rev. Lett. 120 (2018) 145301, http://dx.doi.org/10.1103/PhysRevLett.120.145301.
[14] M. Rupp, A. Tkatchenko, K.-R. Müller, O.A. von Lilienfeld, Phys. Rev. Lett. 108 (2012) 058301, http://dx.doi.org/10.1103/PhysRevLett.108.058301.
[15] K. Hansen, F. Biegler, R. Ramakrishnan, W. Pronobis, O.A. von Lilienfeld, K.-R. Müller, A. Tkatchenko, J. Phys. Chem. Lett. 6 (12) (2015) 2326–2331, http://dx.doi.org/10.1021/acs.jpclett.5b00831.
[16] F. Legrain, J. Carrete, A. van Roekeghem, S. Curtarolo, N. Mingo, Chem. Mater. 29 (15) (2017) 6220–6227, http://dx.doi.org/10.1021/acs.chemmater.7b00789.
[17] M. Tsubaki, T. Mizoguchi, J. Phys. Chem. Lett. 9 (19) (2018) 5733–5741, http://dx.doi.org/10.1021/acs.jpclett.8b01837, PMID: 30081630.
[18] 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.
[19] K.T. Schütt, H.E. Sauceda, P.-J. Kindermans, A. Tkatchenko, K.-R. Müller, J. Chem. Phys. 148 (24) (2018) 241722, http://dx.doi.org/10.1063/1.5019779.
[20] A.P. Bartók, M.C. Payne, R. Kondor, G. Csányi, Phys. Rev. Lett. 104 (2010) 136403, http://dx.doi.org/10.1103/PhysRevLett.104.136403.
[21] B. Nebgen, N. Lubbers, J.S. Smith, A.E. Sifain, A. Lokhov, O. Isayev, A.E. Roitberg, K. Barros, S. Tretiak, J. Chem. Theory Comput. 14 (9) (2018) 4687–4698, http://dx.doi.org/10.1021/acs.jctc.8b00524, PMID: 30064217.
[22] F. Brockherde, L. Vogt, L. Li, M.E. Tuckerman, K. Burke, K.-R. Müller, Nature Commun. 8 (2017) 872, http://dx.doi.org/10.1038/s41467-017-00839-3.
[23] A. Grisafi, A. Fabrizio, B. Meyer, D.M. Wilkins, C. Corminboeuf, M. Ceriotti, ACS Cent. Sci. 5 (1) (2019) 57–64, http://dx.doi.org/10.1021/acscentsci.8b00551.
[24] A.V. Sinitskiy, V.S. Pande, Deep neural network computes electron densities and energies of a large set of organic molecules faster than density functional theory (DFT), 2018, arXiv:arXiv:1809.02723.
[25] O.T. Unke, M. Meuwly, J. Chem. Theory Comput. 15 (6) (2019) 3678–3693, http://dx.doi.org/10.1021/acs.jctc.9b00181, PMID: 31042390.
[26] S. Chmiela, H.E. Sauceda, I. Poltavsky, K.-R. Müller, A. Tkatchenko, Comput. Phys. Comm. 240 (2019) 38–45, http://dx.doi.org/10.1016/j.cpc.2019.02.007.
[27] J. Behler, J. Chem. Phys. 134 (7) (2011) 074106, http://dx.doi.org/10.1063/1.3553717.
[28] L. Zhang, J. Han, H. Wang, R. Car, W. E, Phys. Rev. Lett. 120 (2018) 143001, http://dx.doi.org/10.1103/PhysRevLett.120.143001.
[29] Y. Huang, J. Kang, W.A. Goddard, L.-W. Wang, Phys. Rev. B 99 (2019) 064103, http://dx.doi.org/10.1103/PhysRevB.99.064103.
[30] Z. Li, J.R. Kermode, A. De Vita, Phys. Rev. Lett. 114 (2015) 096405, http://dx.doi.org/10.1103/PhysRevLett.114.096405.
[31] A. Glielmo, P. Sollich, A. De Vita, Phys. Rev. B 95 (2017) 214302, http://dx.doi.org/10.1103/PhysRevB.95.214302.
[32] P. Rowe, G. Csányi, D. Alfè, A. Michaelides, Phys. Rev. B 97 (2018) 054303, http://dx.doi.org/10.1103/PhysRevB.97.054303.
[33] N. Artrith, A.M. Kolpak, Nano Lett. 14 (5) (2014) 2670–2676, http://dx.doi.org/10.1021/nl5005674.
[34] E.L. Kolsbjerg, A.A. Peterson, B. Hammer, Phys. Rev. B 97 (2018) 195424, http://dx.doi.org/10.1103/PhysRevB.97.195424.
[35] C. Zeni, K. Rossi, A. Glielmo, A. Fekete, N. Gaston, F. Baletto, A. De Vita, J. Chem. Phys. 148 (24) (2018) 241739, http://dx.doi.org/10.1063/1.5024558.
[36] N. Mounet, M. Gibertini, P. Schwaller, D. Campi, A. Merkys, A. Marrazzo, T. Sohier, I.E. Castelli, A. Cepellotti, G. Pizzi, N. Marzari, Nature Nanotechnol. 13 (3) (2018) 246–252, http://dx.doi.org/10.1038/s41565-017-0035-5.
[37] A. Jain, K.A. Persson, G. Ceder, APL Mater. 4 (5) (2016) 053102, http://dx.doi.org/10.1063/1.4944683.
[38] E. Gossett, C. Toher, C. Oses, O. Isayev, F. Legrain, F. Rose, E. Zurek, J. Carrete, N. Mingo, A. Tropsha, S. Curtarolo, Comput. Mater. Sci. 152 (2018) 134–145, http://dx.doi.org/10.1016/j.commatsci.2018.03.075.
[39] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Proceedings of the 34th International Conference on Machine Learning - Volume 70, in: ICML'17, JMLR.org, 2017, pp. 1263–1272.
[40] A. Khorshidi, A.A. Peterson, Comput. Phys. Comm. 207 (2016) 310–324, http://dx.doi.org/10.1016/j.cpc.2016.05.010.
[41] T. Iwasa, K. Nobusada, J. Phys. Chem. C 111 (1) (2007) 45–49, http://dx.doi.org/10.1021/jp063532w.
[42] N. Artrith, A. Urban, Comput. Mater. Sci. 114 (2016) 135–150, http://dx.doi.org/10.1016/j.commatsci.2015.11.047.
[43] J. Behler, RuNNer A Neural Network Code for HighDimensional Potential-Energy Surfaces, Universität Göttingen, 2018, URL http://www.uni-goettingen.de/de/560580.html.
[44] A. Singraber, J. Behler, C. Dellago, J. Chem. Theory Comput. 15 (3) (2019) 1827–1840, http://dx.doi.org/10.1021/acs.jctc.8b00770.
[45] J.S. Smith, O. Isayev, A.E. Roitberg, Chem. Sci. 8 (2017) 3192–3203, http://dx.doi.org/10.1039/C6SC05720A.
[46] M. Wen, R.S. Elliott, E.B. Tadmor, KLIFF: Kim-based learning-integrated fitting framework, URL https://github.com/mjwen/kliff.
[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, pp. 8024–8035.
[48] 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.
[49] K. Lee, D. Yoo, W. Jeong, S. Han, Comput. Phys. Comm. 242 (2019) 95–103, http://dx.doi.org/10.1016/j.cpc.2019.04.014.
[50] M. Haghighatlari, G. Vishwakarma, D. Altarawy, R. Subramanian, B.U. Kota, A. Sonpal, S. Setlur, J. Hachmann, ChemRxiv (2019) 8323271, http://dx.doi.org/10.26434/chemrxiv.8323271.v1.
[51] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M.B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A.D. Corso, S. de Gironcoli, P. Delugas, R.A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N.L. Nguyen, H.-V. Nguyen, A.O. de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A.P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, J. Phys.: Condens. Matter 29 (46) (2017) 465901, http://dx.doi.org/10.1088/1361-648x/aa8f79.
[52] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (1996) 11169–11186, http://dx.doi.org/10.1103/PhysRevB.54.11169.
[53] B. Onat, E.D. Cubuk, B.D. Malone, E. Kaxiras, Phys. Rev. B 97 (2018) 094106, http://dx.doi.org/10.1103/PhysRevB.97.094106.
[54] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv:1412.6980.

[55] S. Plimpton, J. Comput. Phys. 117 (1995) 1–19.
[56] Currently, a patch for LAMMPS is distributed within PANNA package at https://gitlab.com/PANNAdevs/panna. It defines a new pair style that can read the weights computed with PANNA and use a NN force field for relaxations or molecular dynamics.
[57] E.B. Tadmor, R.S. Elliott, J.P. Sethna, R.E. Miller, C.A. Becker, JOM 63 (7) (2011) 17, http://dx.doi.org/10.1007/s11837-011-0102-6.
[58] R.S. Elliott, E.B. Tadmor, Knowledgebase of interatomic models (KIM) application programming interface (API), 2011, http://dx.doi.org/10.25950/ff8f563a https://openkim.org/kim-api (Online; accessed: 2019-07-05).
[59] E. Kucukbenli, F. Pellegrini, R.S. Elliott, Driver for panna v1.0.0 neural network potentials v000, 2019, OpenKIM, https://doi.org/10.25950/cf495c5a.
[60] 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, 2015, URL https://www.tensorflow.org/.
[61] A. Kokalj, J. Mol. Graph. Model. 17 (3) (1999) 176–179, http://dx.doi.org/10.1016/S1093-3263(99)00028-5.
[62] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q.V. Le, A.Y. Ng, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 25, Curran Associates, Inc., 2012, pp. 1223–1231.
[63] I.T. Jolliffe, J. Cadima, Philos. Trans.. Ser. A Math. Phys. Eng. Sci. 374 (2016) 20150202.
[64] L. van der Maaten, G. Hinton, J. Mach. Learn. Res. 9 (2018) 2579–2605.
[65] J.S. Smith, O. Isayev, A.E. Roitberg, Sci. Data 4 (2017) 170193 EP –, Data Descriptor.
[66] P. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864–B871, http://dx.doi.org/10.1103/PhysRev.136.B864.
[67] T. Fink, J.-L. Reymond, J. Chem. Inf. Model. 47 (2) (2007) 342–353, http://dx.doi.org/10.1021/ci600423u, PMID: 17260980.
[68] F.H. Stillinger, T.A. Weber, Phys. Rev. B 31 (8) (1985) 5262–5271, http://dx.doi.org/10.1103/physrevb.31.5262.
[69] A.P. Bartók, J. Kermode, N. Bernstein, G. Csányi, Phys. Rev. X 8 (4) (2018) 041048, http://dx.doi.org/10.1103/physrevx.8.041048.
[70] L. Bonati, M. Parrinello, Phys. Rev. Lett. 121 (26) (2018) http://dx.doi.org/10.1103/physrevlett.121.265701.
[71] J.J. Hall, Phys. Rev. 161 (3) (1967) 756–761, http://dx.doi.org/10.1103/physrev.161.756.
[72] W. Wang, R. Gómez-Bombarelli, npj Comput. Mater. 5 (1) (2019) 125, http://dx.doi.org/10.1038/s41524-019-0261-5.
[73] T. Xie, A. France-Lanord, Y. Wang, Y. Shao-Horn, J.C. Grossman, Nature Commun. 10 (1) (2019) 2667, http://dx.doi.org/10.1038/s41467-019-10663-6.
[74] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J.R. Scott, N. Wilkins-Diehr, Comput. Sci. Eng. 16 (5) (2014) 62–74, http://dx.doi.org/10.1109/MCSE.2014.80.