Computer Physics Communications 207 (2016) 310–324

Contents lists available at ScienceDirect

Computer Physics Communications

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

Amp: A modular approach to machine learning in atomistic simulations[☆]

Alireza Khorshidi, Andrew A. Peterson[*]

School of Engineering, Brown University, Providence, RI, 02912, United States

ARTICLE INFO

Article history:
Received 4 March 2016
Received in revised form
4 May 2016
Accepted 8 May 2016
Available online 2 June 2016

Keywords:
Potential energy surface
Neural networks
Atomic Simulation Environment (ASE)
Density functional theory
Zernike polynomials

ABSTRACT

Electronic structure calculations, such as those employing Kohn–Sham density functional theory or *ab initio* wavefunction theories, have allowed for atomistic-level understandings of a wide variety of phenomena and properties of matter at small scales. However, the computational cost of electronic structure methods drastically increases with length and time scales, which makes these methods difficult for long time-scale molecular dynamics simulations or large-sized systems. Machine-learning techniques can provide accurate potentials that can match the quality of electronic structure calculations, provided sufficient training data. These potentials can then be used to rapidly simulate large and long time-scale phenomena at similar quality to the parent electronic structure approach. Machine-learning potentials usually take a bias-free mathematical form and can be readily developed for a wide variety of systems. Electronic structure calculations have favorable properties – namely that they are noiseless and targeted training data can be produced on-demand – that make them particularly well-suited for machine learning. This paper discusses our modular approach to atomistic machine learning through the development of the open-source Atomistic Machine-learning Package (*Amp*), which allows for representations of both the total and atom-centered potential energy surface, in both periodic and non-periodic systems. Potentials developed through the atom-centered approach are simultaneously applicable for systems with various sizes. Interpolation can be enhanced by introducing custom descriptors of the local environment. We demonstrate this in the current work for Gaussian-type, bispectrum, and Zernike-type descriptors. *Amp* has an intuitive and modular structure with an interface through the python scripting language yet has parallelizable fortran components for demanding tasks; it is designed to integrate closely with the widely used Atomic Simulation Environment (ASE), which makes it compatible with a wide variety of commercial and open-source electronic structure codes. We finally demonstrate that the neural network model inside *Amp* can accurately interpolate electronic structure energies as well as forces of thousands of multi-species atomic systems.

Program summary

Program title: Amp

Catalogue identifier: AFAK_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AFAK_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: yes

No. of lines in distributed program, including test data, etc.: 21239

No. of bytes in distributed program, including test data, etc.: 1412975

Distribution format: tar.gz

Programming language: Python, Fortran.

Computer: PC, Mac.

Operating system: Linux, Mac, Windows.

Has the code been vectorized or parallelized?: Yes

[☆] 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. Tel.: +1 401 863 2153.
E-mail address: andrew_peterson@brown.edu (A.A. Peterson).

http://dx.doi.org/10.1016/j.cpc.2016.05.010

RAM: Variable, depending on the number and size of atomic systems.

Classification: 16.1, 2.1.

External routines: ASE, NumPy, SciPy, f2py, matplotlib

Nature of problem:
Atomic interactions within many-body systems typically have complicated functional forms, difficult to represent in simple pre-decided closed-forms.

Solution method:
Machine learning provides flexible functional forms that can be improved as new situations are encountered. Typically, interatomic potentials yield from machine learning simultaneously apply to different system sizes.

Unusual features:
Amp is as modular as possible, providing a framework for the user to create atomic environment descriptor and regression model at will. Moreover, it has Atomic Simulation Environment (ASE) interface, facilitating interactive collaboration with other electronic structure calculators within ASE.

Running time:
Variable, depending on the number and size of atomic systems.

1. Introduction

Atomistic calculations have become routine tasks to study complicated systems at atomic, molecular, or larger length scales. These calculations are often centered around the fundamental concept of the potential energy surface (PES). Within the framework of Born–Oppenheimer approximation [1], the PES describes the potential energy of a system as a function of the positions of the atomic nuclei. A graphical interpretation of the PES gives an intuitive insight into a variety of concepts of chemistry: Local minima on the PES are concave up in all local directions, and correspond to stable molecules or phases; these regions can be found by geometry optimization [2]. On the other hand, a first-order saddle point on the PES is concave down in just one local direction but concave up in all other local directions; the characteristics of this saddle-point region can provide insight of the rate of chemical reaction or phase transition, such as via transition-state theory [3]. In normal-mode analysis, the curvature of the potential well at the local minimum approximates the vibrational entropy of the corresponding stable configuration [4]. In *ab initio* molecular dynamics, the system evolves over time in a classical way according to the PES calculated within the context of quantum mechanics [5]. Therefore, an accurate representation of the PES is a pre-requisite for the rigorous study of the dynamics of systems. In fact, the validity of studies such as reaction dynamics, vibrational analysis, and molecular dynamics depends completely on the reliability of the PES.

The exact analytical solution to the many-body Schrödinger equation – which provides energy as a function of atomic positions – is considered possible for only a few molecular systems, such as a single hydrogen atom (H) or a hydrogen molecular ion (H_2^+) [6]. However, over the years, many valuable theories have been developed [7] to provide accurate approximations to the electronic structure of complicated many-body systems, such as perturbation theory, Hartree–Fock theory, and Kohn–Sham density functional theory (hereafter abbreviated as DFT). In particular, DFT has been extensively used to complement experimental observations, as well as to provide insights and mechanisms not accessible by experiment in a wide range of scientific research areas [8]. However, the computational cost of electronic structure methods tends to grow drastically with the system size, and typically scales nonlinearly with the number of electrons in the system. Moreover, the number of possible geometrical configurations and compositions increases exponentially with the number and identities of atoms in the system. These issues make electronic structure methods typically impractical for systems larger than a few hundreds of atoms; however, many noteworthy phenomena (*e.g.* ligand interaction with protein macromolecules [9] or mechanical and electro-mechanical field effects) happen in much larger length scales. Moreover, an *ab initio* molecular dynamics simulation over a period of a few nanoseconds would require millions of quantum mechanical calculations, and would take years to finish with current computer facilities. Such simulations could give precise insights of phenomena in different fields, *e.g.* structural evolution of proteins in biochemistry [10], hydrochemical processes within soils in geotechnics [11], or response of nanostructures to loadings in materials science [12,13]. This bottleneck in computational power can be expected to persist in the foreseeable future, leading to a need for improved computational methodologies.

To counter the above-mentioned difficulties, a great amount of effort has been devoted to develop models that mimic the PES, learning from quantum mechanical calculations. This approach of fitting analytical expressions to the PES, with a fixed number of atoms and chemical species, has a rich history [14–21], including moving Shepard interpolation [17], reproducing kernel Hilbert space [18], interpolating moving least-squares [19], and neural networks fed by transformed atomic coordinates [20,21] as such. In particular, the neural-network model has been further suggested to be fed by a number of other types of inputs such as permutationally invariant polynomials [22] or cluster expansion coefficients [23]. For a further review of these methods, the reader is referred to Ref. [24].

The constraint of fixed number of atoms can be relaxed by assuming the potential energy of an atomic system is an extensive variable of atomic energy contributions. The potentials yielded from this approach are size-extensible, meaning that they can simultaneously describe energetics of different system sizes. Commonly each atomic energy contribution is further assumed to depend on the local chemical environment only. These assumptions indeed make it possible for a model to learn from the energetics of a number of local environments, then predict the energies of unseen systems (with arbitrary sizes) having similar local environments. In practice, atomic systems typically have lots of similarity in local environments between images or even atoms within one image. As a simplified yet illustrative example, Fig. 1 shows a periodic system of hydrogen and carbon monoxide adsorbates

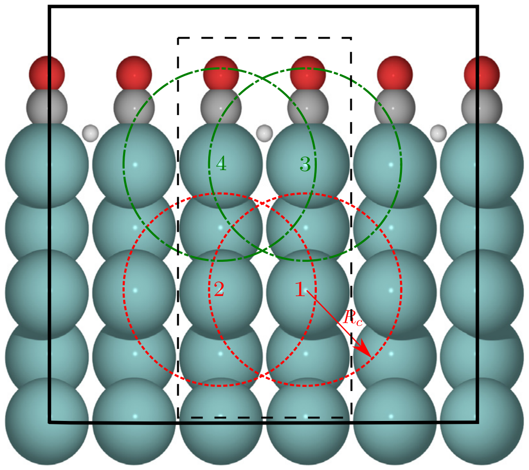

Fig. 1. A sample periodic atomic system consisting of hydrogen and carbon monoxide adsorbates on a molybdenum surface. Local environments are shown by dashed circles with cutoff radius R_c. Atoms with indices 1 and 2 inside the main unit cell experience the same local environments (red dotted circles). Similarly, local environments (green dash-dotted circles) of atoms with indices 3 and 4 are identical down to a reflection. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

on a molybdenum surface. The local environments (shown by red dotted circles with cutoff radius R_c) of atoms with indices 1 and 2 are identical. Similarly, atoms with indices 3 and 4 experience the same local environments (shown by green dash-dotted circles) down to a reflection. The bond-order potentials (*e.g.* ReaxFF [25]) accomplish size-extensibility by taking the locality approach discussed above. Force fields resulting from bond-order potentials have been used for large-scale molecular dynamics simulations (*e.g.* [26]).

As an alternative to system-dependent physically-inspired models, regression models based on machine learning can be systematically formulated to provide a general, size-extensible framework to interpolate the PES of any atomic system of interest (regardless of chemical species, periodicity, *etc.*). Although the types of machine-learning models vary, many have the property of a relatively agnostic functional form that is capable of reproducing arbitrary functions to user-specified accuracy, provided sufficient high-quality training data is available. The data produced by a variety of quantum mechanics theories [7], spanning from functional-based methods (*e.g.* DFT) to Slater-determinant-based methods (such as Møller-Plesset perturbation theory or multi-configuration self-consistent field theory), are very well-suited to machine-learning representations in the sense that they are noiseless – that is, perfectly precise – and further training data can be produced on demand. A size-extensible machine-learning approach was taken by Behler and Parrinello [27], where the authors suggested a new neural network representation of the PES tailored to be size-extensible. While the original formulation was only for systems containing a single type of element (*e.g.* all Na atoms), this was extended to allow multiple types of elements in the same model by Artrith et al. [28]. Separately, Bartók et al. [29] have used Gaussian processes with a bispectrum derived from a four-dimensional spherical harmonics representation of the atomic environment to produce a new form of size-extensible interpolating potentials. We shall return to the Behler–Parrinello and also Bartók et al. approaches later. When using a systematic regression model, the number of adjustable parameters can be easily changed, and an efficient number (which represents a trade-off between model size and accuracy and can depend on the size of the reference data set) can be selected. On the other hand, as the size of the reference data grows, the number of parameters can also be increased simultaneously, leading to a more sophisticated representation of the PES.

With the rise of routine quantum-mechanical calculations in the research community, many researchers have amassed large amounts of data on personal machines and clusters. This suggests that typical users could benefit from their existing data, by being able to utilize machine learning to predict force fields on unseen systems, greatly accelerating their day-to-day tasks. This suggests the need for a straightforward, open-source machine-learning code for atomistics that is flexible enough to handle various data sources and fitting models. We have developed the open-source *Amp* (Atomistic Machine-learning Package), publicly available at [30]; it aims to fill this need by using relatively fast machine-learning models to interpolate the potential energies and atomic forces of a reference data set (referred to as the "training" data set) prepared by a parent electronic structure calculator. Boes et al. [31] have recently used this code to carry out a benchmark comparative study between a well-trained neural network potential inside *Amp* and a ReaxFF potential for gold properties. *Amp* is designed to integrate closely with the Atomic Simulation Environment (ASE) [32], and therefore, any calculator available in ASE (*e.g.* EMT [33], GPAW [34], Dacapo [35], VASP [36], NWChem [37], Quantum ESPRESSO [38], ...) may be used as the parent calculator to prepare the training data set; so long as the calculator produces a unique value of energy for a given atomic configuration, *Amp* is agnostic to the level of theory employed in the calculator. *Amp* is as intuitive and modular as possible, and can readily take user-defined atomic environment descriptors and regression models. It is built to be extensible to a variety of descriptors (or transformations of the geometric information) and machine-learning regression models, as they arise in future research. To facilitate such easy modifications, *Amp* has a pure python interface, although several computationally heavy parts of the underlying code have parallel and fortran versions to speed up calculations.

This paper discusses both the theoretical foundations as well as coding philosophies of *Amp*. We provide the general properties, as well as several examples of descriptors and models in the machine-learning approach. In the last section, for demonstration purposes, the PES of a quaternary system is interpolated using *Amp*.

2. Theory

Within the framework of the Born–Oppenheimer approximation, the ground-state potential energy E of an atomic configuration is a unique function of the positions of the nuclei; that is, $E = E(\tilde{\mathbf{R}})$ where $\tilde{\mathbf{R}} = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_N\}$, N is the number of atoms in the system, and $\mathbf{R}_i$ is the position of atom i. In general, $E(\tilde{\mathbf{R}})$ is a very complicated function, and individual data points of the PES are often found (or actually, approximated) most efficiently by computationally-intensive, variational techniques such as DFT. Here, we will refer to the quantum mechanics PES as any repeatable calculation of $E(\tilde{\mathbf{R}})$, such as arises from DFT, noting that more sophisticated electronic structure methods can give results that more closely match the true PES. As this quantum mechanics PES is the parent data set used to train machine-learning representations, for the purposes of this paper it is the standard which machine-learning methods will mimic.

Machine-learning regression algorithms excel at fitting approximate functional forms to arbitrary, unknown functions; with this, the idea is to approximate $E(\tilde{\mathbf{R}})$ with the machine-learned function $\hat{E}(\tilde{\mathbf{R}})$ using a regression model:

$$\left\{(\tilde{\mathbf{R}}_1, E_1), (\tilde{\mathbf{R}}_2, E_2), \ldots\right\} \xrightarrow{\text{regression}} E = \hat{E}(\tilde{\mathbf{R}}), \quad (1)$$

where $(\tilde{\mathbf{R}}_i, E_i)$ denotes the training data consisting of individual points (atomic images) on the quantum mechanics PES. That is, a method such as DFT can be used to calculate many distinct images on the PES, and then machine-learning regression – such as artificial neural networks or kernel ridge regression – can develop a closed-form, analytical expression to approximate $E(\tilde{\mathbf{R}})$. Note that the potential energy E of the system is typically invariant with respect to arbitrary translation and rotation of the whole system, in the absence of external fields or constraints. Translation and rotation invariance can be taken into account by defining the potential energy E as a function of a set of "internal coordinates", $\tilde{\mathbf{R}}^{(\mathrm{int})}$, *e.g.* bond lengths, valence angles, or dihedral angles. Thus, a modified formulation of Eq. (1) with rotation and translation invariance could be considered, as given below:

$$\left\{(\tilde{\mathbf{R}}_1^{(\mathrm{int})}, E_1), (\tilde{\mathbf{R}}_2^{(\mathrm{int})}, E_2), \ldots\right\} \xrightarrow{\text{regression}} E = \hat{E}(\tilde{\mathbf{R}}^{(\mathrm{int})}). \tag{2}$$

The number of independent internal coordinates is $3N - 6$ in a system of N atoms, (*e.g.* for a molecular system where three translation and three rotation degrees of freedom have been removed). A more efficient interpolation can be achieved if internal coordinates are first mapped into a feature space F, which may reduce the dimensionality or otherwise make the problem more suitable to a machine-learning representation [39]. Data points in the feature space F along with energies of the atomic systems can then be used for regression:

$$\begin{aligned}&\left\{(\tilde{\mathbf{R}}_1^{(\mathrm{int})}, E_1), (\tilde{\mathbf{R}}_2^{(\mathrm{int})}, E_2), \ldots\right\}\\ &\xrightarrow{\text{feature map}} \left\{(\mathbf{G}(\tilde{\mathbf{R}}_1^{(\mathrm{int})}), E_1), (\mathbf{G}(\tilde{\mathbf{R}}_2^{(\mathrm{int})}), E_2), \ldots\right\}\\ &\xrightarrow{\text{regression}} E = \hat{E}(\mathbf{G}(\tilde{\mathbf{R}}^{(\mathrm{int})})),\end{aligned} \tag{3}$$

where $\mathbf{G} \in F$ is the feature vector of image. Regression models typically cannot capture discontinuity in the input space, and so, it is crucial that $\mathbf{G}\left(\mathbf{R}^{(\mathrm{int})}\right)$ in Eq. (3) is a continuous function of $\mathbf{R}^{(\mathrm{int})}$.

The above equations represent what we will refer to as a general "image-centered" framework for interpolating the PES—that is, for each image in the training set, the coordinates are arranged into a single vector which is then fed into a machine-learning model. This vector may be as simple as a single-column list of the Cartesian positions of all atoms in the image. For a unique representation of atomic positions, the input vector to the regression model should contain at least $3N - 6$ components corresponding to independent internal coordinates, in either the case of using pure internal coordinates (Eq. (2)) or the case of using a function of them (Eq. (3)). Other than translation- and rotation-invariance discussed above, to match the true physics, the calculated potential value should also be permutation-invariant; *i.e.* invariant with respect to relabeling of atoms with the same chemical identities. However, these properties of translation, rotation, and permutation invariance are not guaranteed in an image-centered scheme, and the image-centered scheme is not, in general, size extensible; that is, the number and chemical identity of atoms per image is typically fixed.

Next, we discuss a general "atom-centered" framework for interpolating the PES, which can systematically address some of the limitations of the image-centered scheme discussed above. A common approach for developing empirical potentials is to break up the total energy of the system into atomic energy contributions. Using the locality approximation, each atomic energy contribution E_i can be taken to depend only on the local environment of atom i as

$$E = \sum_{i=1}^{N} E_i(\tilde{\mathbf{R}}^{(\mathrm{loc})}). \tag{4}$$

As a result, an "atom-centered" framework of learning E becomes possible. In this framework, instead of directly modeling the total image energy $E(\tilde{\mathbf{R}})$, the atomic energy contributions E_i are modeled, defined in the much dimensionally reduced space of local environments $\tilde{\mathbf{R}}^{(\mathrm{loc})}$. This local environment is typically defined to be a sphere of radius R_c (the "cutoff radius") surrounding the position of atom i; however, longer-range interactions can still be included via techniques such as Ewald summations [28,40]. Similar to Eq. (3), a proper mapping from the space of now local environments into a feature space F representing the functional dependence of local energetics can further improve interpolation, leading to

$$\begin{aligned}&\left\{(\tilde{\mathbf{R}}_1, E_1), (\tilde{\mathbf{R}}_2, E_2), \ldots\right\}\\ &\xrightarrow{\text{feature map and regression}} E = \sum_{i=1}^{N} \hat{E}_i(\mathbf{G}_i(\tilde{\mathbf{R}}^{(\mathrm{loc})})),\end{aligned} \tag{5}$$

where $\mathbf{G}_i(\tilde{\mathbf{R}}^{(\mathrm{loc})}) \in F$ in the feature vector of atom i, and $\hat{E}_i$ is the machine-learned function that approximates E_i. In the remainder of this paper, we will sometimes refer to the feature vector as the "fingerprint" of an atom's local environment, after the convention of Behler et al. [41].

Modularity in *Amp*. To allow for flexibility in both the transformation of the coordinates as well as the choice of machine-learning model, we have built *Amp* with two key components as shown in Fig. 2: The input atomic structure is transformed, for example to describe the local environment about each atom, and then the transformed structure is fed into a model, which is fit from training data to give the potential energy (and/or forces). We abstract the code to allow the user to individually choose or create both parts of the scheme independently, through *descriptor* and *model* keywords, which are then regressed through methods in *Amp*.

A schematic of how *Amp* works in the image-centered mode is shown in Fig. 3(a). This mode makes no locality assumption. The simplest choice of descriptor in this mode is to leave the coordinates in Cartesian form untransformed. In this case, the scheme based on Eq. (1) is adopted, with $\tilde{\mathbf{R}}$ being pure Cartesian coordinates of all atoms in the atomic configuration. This choice works only if the number and identities of atoms in the systems under study do not change; only the atomic positions can change. Also, all images should be homogeneous with respect to periodicity of atomic configurations. Because no feature mapping is made to the coordinates, this choice can in principle always represent the PES to arbitrary accuracy, provided the machine-learning model has enough parameters (and enough training data are provided). A transformation on the Cartesian coordinates of the atomic configuration might be further performed leading to a single feature vector $\mathbf{G}$ for the whole image as in Eq. (3).

Fig. 3(b) shows how *Amp* behaves when it adopts the atom-centered scheme represented by Eq. (5). In this scheme, permutation invariance is achieved by assuming that the same machine-learning model parameters (for example weights and biases for neural networks) apply to atoms of the same chemical species. This scheme can always be used for atomic images with different numbers of atoms (system sizes). Also, either or both the training and test data sets can be inhomogeneous with respect to periodicity of atomic configurations. Importantly, the atom-centered scheme relies on a locality assumption, so is not mathematically guaranteed to be capable of replicating the quantum mechanics PES, particularly when long-range interactions are present or the fingerprint is not complete enough to describe all local interactions.

The following sections give more background on the descriptor and model, and a discussion of some popular methodologies of each.

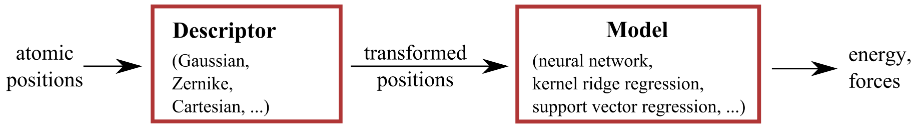
atomic positions
Descriptor
(Gaussian, Zernike, Cartesian, ...)
transformed positions
Model
(neural network, kernel ridge regression, support vector regression, ...)
energy, forces

Fig. 2. The modular nature of *Amp* is designed to allow splitting of the representation into two components, a descriptor and a model, shown schematically with listed examples.

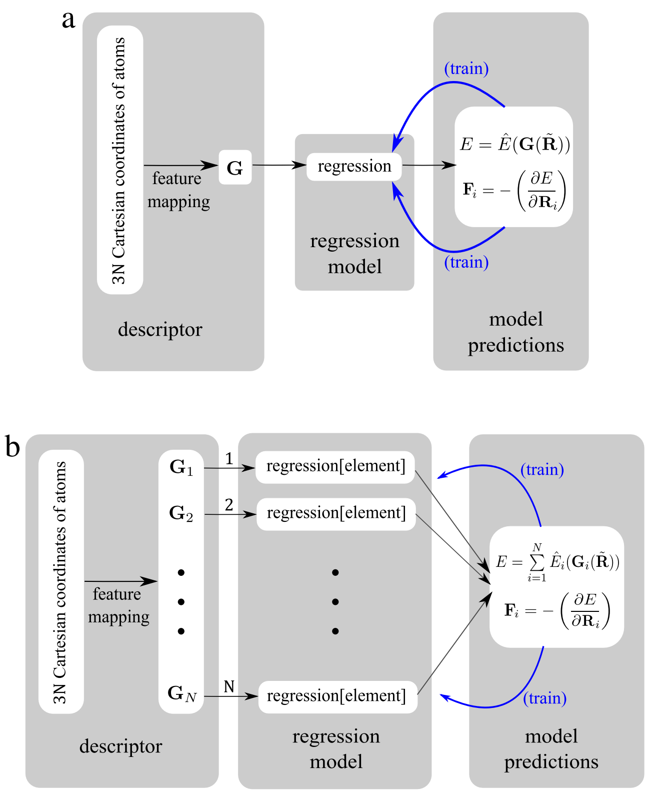
a
3N Cartesian coordinates of atoms
feature mapping
G
regression
regression model
(train)
$E = \hat{E}(\mathbf{G}(\tilde{\mathbf{R}}))$
$\mathbf{F}_i = -\left(\frac{\partial E}{\partial \mathbf{R}_i}\right)$
(train)
descriptor
model predictions
b
3N Cartesian coordinates of atoms
feature mapping
$\mathbf{G}_1$
$\mathbf{G}_2$
$\mathbf{G}_N$
1
2
N
regression[element]
regression[element]
regression[element]
(train)
$E = \sum_{i=1}^{N} \hat{E}_i(\mathbf{G}_i(\tilde{\mathbf{R}}))$
$\mathbf{F}_i = -\left(\frac{\partial E}{\partial \mathbf{R}_i}\right)$
(train)
descriptor
regression model
model predictions

Fig. 3. Schematics of how *Amp* calculator works in (a) image-centered mode and (b) atom-centered mode.

2.1. Descriptor

In our scheme, the *descriptor* is a transformation of the atomic positions that typically produces feature vectors (fingerprints) describing the atomic system, which are suitable to be fed into various machine-learning models. We have designed *Amp* to accommodate the two major frameworks of fingerprinting schemes described in the previous section: either image-centered in which a single feature vector is produced for the entire image, or atom-centered in which one feature vector is generated for each atom in an image. In the following sections, three possible approaches for describing the local chemical environment of atoms using Gaussian functions, a Zernike descriptor, and a bispectrum descriptor are discussed; all representations are available as built-in modules inside *Amp*, and illustrate how future descriptors can be incorporated into the *Amp* framework. In Section 4, we will demonstrate the use of these three descriptors.

2.1.1. Gaussian

Describing the local chemical environment around atoms by Gaussian functions has been suggested by Behler and Parrinello, who employed such a scheme in tandem with element-specific neural networks [27,42]. According to their approach, the feature vector $\mathbf{G}_i$ of Eq. (5) for atom i is partitioned into two separate subvectors $\mathbf{G}_i^{\mathrm{I}}$ and $\mathbf{G}_i^{\mathrm{II}}$ ($\mathbf{G}_i^{\mathrm{I}} \cup \mathbf{G}_i^{\mathrm{II}} = \mathbf{G}_i$) representing the interactions of atom i with neighboring single atoms and pairs of atoms, respectively. The components of the subvector $\mathbf{G}_i^{\mathrm{I}}$ are comprised of radial functions, denoted f_i^{I},

$$f_i^{\mathrm{I}} = \sum_{\substack{\text{atoms } j \text{ within } R_c \\ \text{distance of atom } i \\ j \neq i}} e^{-\eta (R_{ij}-R_s)^2/R_c^2} f_c\left(R_{ij}\right), \tag{6}$$

which capture pairwise interactions of atom i with all neighboring atoms j based upon the distance between the two atoms, $R_{ij} = \|\mathbf{R}_{ij}\| = \|\mathbf{R}_j - \mathbf{R}_i\|$. The feature subvector $\mathbf{G}_i^{\mathrm{I}}$ for the local environment of atom i is formed such that different components of $\mathbf{G}_i^{\mathrm{I}}$ are calculated from f_i^{I} with different values for the width and center parameters, η and R_s. f_c is a cutoff function, which provides a smooth approach to no contribution for atoms at the cutoff radius; this will be fully defined below.

Triple-atom interactions are captured by incorporating angular dependence into the f_i^{II} functions which make up the values in the $\mathbf{G}_i^{\mathrm{II}}$ subvector. For each atom i, the function is constructed by summing over the cosine values of the angles $\theta_{ijk} = \cos^{-1}\left(\mathbf{R}_{ij}.\mathbf{R}_{ik}/\left(R_{ij}R_{ik}\right)\right)$, for all possible triplets of atoms i, j, and k centered at atom i, according to

$$f_i^{\mathrm{II}} = 2^{1-\zeta} \sum_{\substack{\text{atoms } j,\,k \text{ within } R_c \\ \text{distance of atom } i \\ j,\,k \neq i \\ (j \neq k)}} \left(1 + \lambda \cos \theta_{ijk}\right)^{\zeta} e^{-\eta\left(R_{ij}^2+R_{ik}^2+R_{jk}^2\right)/R_c^2} \times f_c\left(R_{ij}\right) f_c\left(R_{ik}\right) f_c\left(R_{jk}\right). \tag{7}$$

Similar to $\mathbf{G}_i^{\mathrm{I}}$, the various components of $\mathbf{G}_i^{\mathrm{II}}$ are calculated from f_i^{II} with different values for parameters λ, η, and ζ. Fig. 4 shows how components of Gaussian feature subvectors $\mathbf{G}_i^{\mathrm{I}}$ and $\mathbf{G}_i^{\mathrm{II}}$ change with, respectively, distance R_{ij} between pair atoms i and j and valence angle θ_{ijk} between triplet of atoms i, j, and k with central atom i.

Since the atomic interactions that go into the feature subvectors $\mathbf{G}_i^{\mathrm{I}}$ and $\mathbf{G}_i^{\mathrm{II}}$ are only counted within a sphere defined by a cutoff radius (R_c), a cutoff function $f_c\left(r\right)$ is employed in Eqs. (6) and (7) to smoothly decay the values to zero as $r \to R_c$. That is, f_c must be zero at $r = R_c$. Without such a function, a discontinuity would arise in the fingerprints as neighbors moved differentially across the cutoff boundary; this could lead to singularities in calculated forces. In order to have a continuous force-field, the cutoff function $f_c\left(r\right)$ as well as its first derivative should be continuous in $r \in [0, \infty)$, as discontinuities in the derivative can lead to discontinuities in the model-predicted forces. The expression employed by Behler and colleagues [42] for such a function is given below:

$$f_c\left(r\right) = \begin{cases} 0.5\left(1 + \cos\left(\pi r/R_c\right)\right) & \text{if } r \leq R_c, \\ 0 & \text{if } r > R_c. \end{cases} \tag{8}$$

This has the desirable features of being zero at $r = R_c$, one at $r = 0$, and a smooth function in between. However, we note that multiplying the cutoff function (8) three times as in the expression of Eq. (7) proposed by the Ref. [42], makes f_i^{II} fall off very rapidly with pair-atom distances, which in turn may result in a poor description of interactions between triple atoms relatively far apart, but within local environment sphere. This suggests that a cutoff function that maintains a larger value through the cutoff sphere may be preferable. We can construct a general polynomial form with the same limiting behavior at $r = 0$ and $r = R_c$, but with a parameter γ that

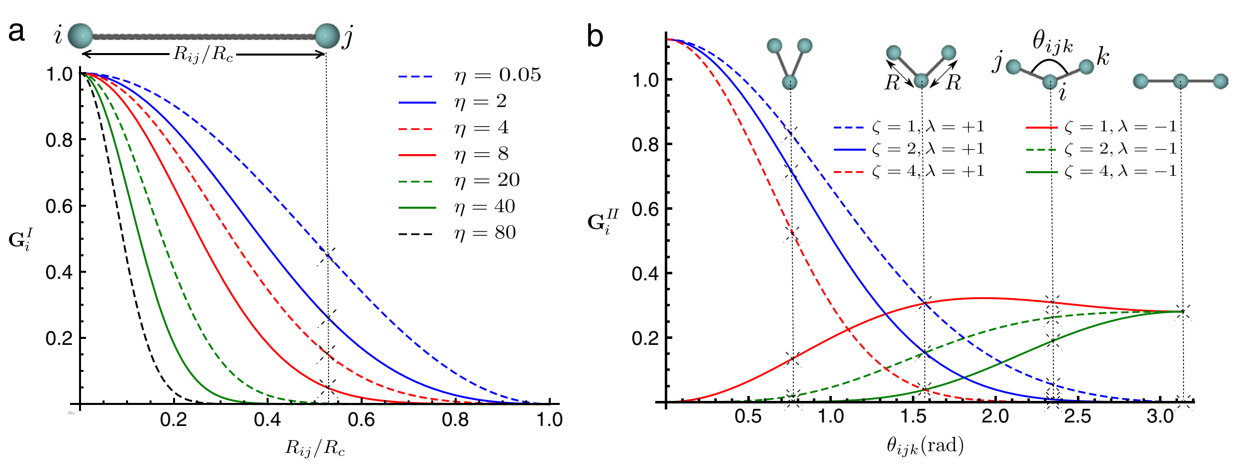

Fig. 4. Variations of components of Gaussian feature subvectors $\mathbf{G}_i^{\mathrm{I}}$ and $\mathbf{G}_i^{\mathrm{II}}$ of atom i with cosine cutoff function (described later, Eq. (8)) versus pair-atom distance R_{ij} and valence angle θ_{ijk}, respectively; (a) components of $\mathbf{G}_i^{\mathrm{I}}$ as given by Eq. (6) for $R_s = 0$, (b) components of $\mathbf{G}_i^{\mathrm{II}}$ as given by Eq. (7) for $\eta = 0.005$ and $R = R_c/3$.

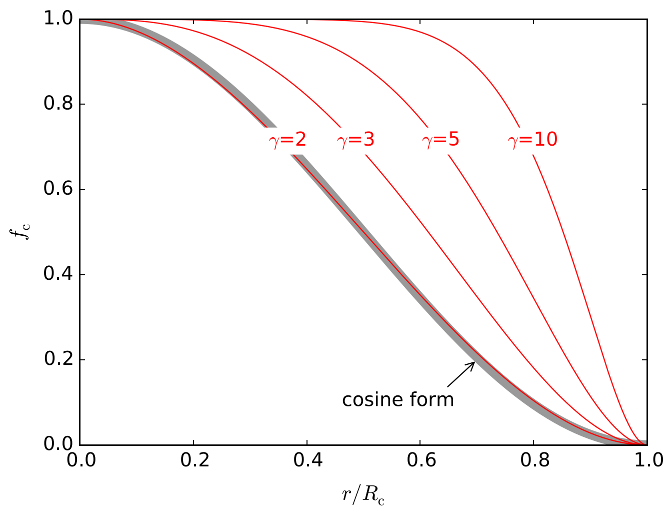

Fig. 5. The cutoff function. In gray is the cosine-based function suggested by Behler and colleagues. In the foreground, for several different values of γ, is the polynomial-based version proposed in this paper in Eq. (9). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

can be adjusted to delay the decay, as below:

$$f_c(r) = \begin{cases} 1 + \gamma \cdot (r/R_c)^{\gamma+1} - (\gamma + 1)(r/R_c)^{\gamma} & \text{if } r \le R_c, \\ 0 & \text{if } r > R_c. \end{cases} \quad (9)$$

The behavior of this cutoff function is compared to that of the cosine-based function in Fig. 5. Note that the polynomial function (9) approximates the cosine function's behavior if γ is set to 2; as γ is increased the decay is delayed, but the limiting behaviors are intact. As γ is increased to very large values, it approaches the step function, which would be equivalent to not employing a cutoff function. Thus, the polynomial form can take on a range of cutoff function behaviors; this has also been implemented in *Amp* to provide more user-control over cutoff function behavior.

2.1.2. Zernike

The Zernike descriptors have been used for three-dimensional (3-D) shape retrieval in the machine-learning community [43,44]. Here we show the use of these descriptors as an example of extending *Amp* through its modularity, and highlight that they have some desirable properties which may make them advantageous for describing local chemical environments. Among the desirable properties:

1. They form an orthogonal basis set for describing 3-D functions within a set volume.
2. The basis set is in principle complete as its size is (unboundedly) increased; this means that the size of the fingerprint can be straightforwardly optimized to trade-off efficiency and accuracy.
3. They automatically allow for N-body interactions, where N is unlimited, as opposed to the Gaussian descriptors which have been implemented only for two- and three-body interactions.
4. They naturally allow multiple-element interactions without increasing fingerprint sizes. This also allows elements to be added to a training set "on-the-fly", as opposed to the Gaussian approach.
5. They are reasonably computationally efficient, as compared with the other two descriptors of this work: Gaussian (discussed before) and bispectrum (will be discussed later on). Since they can be explicitly expressed in terms of monomials, calculation of derivatives can be computationally efficient, accelerating training and force-call evaluation.

The general procedure that we employ in using the Zernike descriptor as a fingerprint vector will first be described, before the mathematics are detailed. First, the atomic environment within a cutoff radius is expressed as a function comprised of Dirac delta distributions at the locations of the neighboring atoms; these Dirac deltas bear coefficients that label the individual elements. Then, this function is represented in terms of the Zernike basis set; the coefficients of this representation are then condensed to become rotationally invariant, and used to construct a fingerprint that describes the atom's local environment. The procedure of fitting basis-set coefficients to a distribution of deltas is inspired by the approach developed by Bartók et al. [29] for the bispectrum approach, which is summarized later in Section 2.1.3.

We begin with briefly reviewing the theoretical backgrounds of Zernike moments and the Zernike descriptor; please see Ref. [44] for an excellent detailed discussion. For the purpose of analysis of the shape of an object, moments are helpful tools which are defined as the projection of the function describing the object onto a particular set of basis functions. The basis set can be either orthogonal or non-orthogonal; Zernike moments fall under the former category. Orthogonality of the basis set is a favorable property, since it yields that the computed features of the object be mutually independent. Consequently it allows for a compact and efficient representation of the object without redundant and correlated information. Another favorable property of a basis set is its completeness, which implies that any square-integrable function defining an object can be approximated as a linear combination of moments of the basis set within arbitrary accuracy. Essentially, this means that the reconstructed object converges to the original object as the number of utilized moments (unboundedly) increases. In shape retrieval, the Zernike basis has been shown to outperform its ancestors (*e.g.* geometric and complex moments based on real and complex monomials, respectively) because of its orthogonality and completeness properties [45].

Here, we mathematically define the Zernike basis. As the upcoming discussion is heavily tied to the concept of the unit sphere and for the sake of clarity, we first define the 2-sphere (S^2) of unit radius as the set of points $\mathbf{x} \in \mathbb{R}^3$, with $\|\mathbf{x}\|^2 = \sum_{i=1}^{3} x_i{}^2 = 1$. That is, the conventional spherical coordinates $(\tilde{r}, \theta, \phi)$ correspond to points inside S^2 for $0 \leq \tilde{r} < 1$, and to points on the surface of S^2 for $\tilde{r} = 1$. This sphere will serve as the local neighborhood of the central atom in which neighbor interactions are accounted; that is, it defines the space inside of the cutoff radius with $\tilde{r} \equiv r/R_c$. Conceptually, a 3-D Zernike basis set (Z) is formed by a tensor product between the Zernike polynomials basis set (R), which is complete and orthogonal in the domain $0 \leq \tilde{r} < 1$, and the spherical harmonics basis set (Y), which is complete and orthogonal on the surface of S^2. In the explicit form, the 3-D Zernike basis functions Z_{nl}^m, which make up Z, are defined both inside and on the surface of S^2 as

$$Z_{nl}^m(\tilde{r}, \theta, \phi) = R_n^l(\tilde{r}) Y_l^m(\theta, \phi), \tag{10}$$

where the index $n \geq 0$ is an integer, l is restricted to even $n - l \geq 0$, and m is an integer such that $|m| \leq l$. $R_n^l(\tilde{r})$ are the 3-D Zernike polynomials, given by

$$R_n^l(\tilde{r}) = \sqrt{2n+3} \sum_{s=0}^{\frac{n-l}{2}} (-1)^s \binom{\frac{n-l}{2}}{s} \binom{n - s + \frac{1}{2}}{\frac{n-l}{2}} \tilde{r}^{n-2s}, \tag{11}$$

for even $n - l \geq 0$, else identical to zero. $Y_l^m(\theta, \phi)$, on the other hand, are spherical harmonics explicitly represented in terms of associated Legendre polynomials P_l^m as

$$Y_l^m(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta)\, e^{\imath m \phi}, \tag{12}$$

for $\imath = \sqrt{-1}$, integer degree $l \geq 0$, and integer order m such that $|m| \leq l$. The orthogonality relation between Zernike functions takes on the form

$$\left\langle Z_{n_1 l_1}^{m_1}(\tilde{r}, \theta, \phi), Z_{n_2 l_2}^{m_2}(\tilde{r}, \theta, \phi) \right\rangle = \delta_{n_1 n_2} \delta_{l_1 l_2} \delta_{m_1 m_2}, \tag{13}$$

with

$$\langle f(\tilde{r}, \theta, \phi), g(\tilde{r}, \theta, \phi) \rangle$$
$$= \int_0^1 d\tilde{r}\, \tilde{r}^2 \int_0^{\pi} d\theta\, \sin\theta \int_0^{2\pi} d\phi\, f^*\left(\tilde{r}, \theta, \phi\right) g\left(\tilde{r}, \theta, \phi\right), \tag{14}$$

where the superscript $*$ stands for complex conjugate. In Eq. (13), δ is the Kronecker delta function. We can use the 3-D Zernike basis functions to represent a function $f(\tilde{r}, \theta, \phi)$:

$$f(\tilde{r}, \theta, \phi) = \sum_{n=0}^{\infty} \sum_{l} \sum_{m=-l}^{l} c_{nl}^m Z_{nl}^m(\tilde{r}, \theta, \phi), \tag{15}$$

with sum over l running restricted to even $n - l \geq 0$, and $c_{nl}^m = \langle Z_{nl}^m(\tilde{r}, \theta, \phi), f(\tilde{r}, \theta, \phi)\rangle$. To be computationally tractable, the sum running over n is truncated at a maximum value $n_{\max}$; this value can be increased to provide an increasingly accurate representation of the underlying function. Also, for a real-valued function $f\left(\tilde{r}, \theta, \phi\right)$, the symmetry relation $c_{nl}^{-m} = (-1)^m c_{nl}^{m*}$ holds between Zernike moments, which implies that only moments corresponding to $m \geq 0$ need to be computed.

Canterakis [43] has shown that the subset $\left\{Z_{nl}^{-l}, Z_{nl}^{-l+1}, \ldots, Z_{nl}^{l}\right\}$ forms a $(2l+1)$-dimensional irreducible subspace, with the property that the norm $\|\mathbf{c}_{nl}\|$ of the vector $\mathbf{c}_{nl} = \left(c_{nl}^{-l}, c_{nl}^{-l+1}, \ldots, c_{nl}^{l}\right)^T$ of moments is invariant under the rotation of the corresponding function $f\left(\tilde{r}, \theta, \phi\right)$ about the origin (superscript T stands for transpose). Therefore, the 3-D Zernike descriptor defined in the form of a vector whose components are $\|\mathbf{c}_{nl}\|$ for different values of n and l, has the favorable property of rotation invariance.

The local chemical environment of atom i is taken within a sphere centered at atom i with a specified cutoff radius R_c, similar to the Gaussian feature vector discussed in Section 2.1.1. Translation invariance is again achieved by setting the origin (*i.e.* the center of the sphere) to be at atom i's nuclear position. To compute components of the 3-D Zernike descriptor associated with the local environment of atom i, we define an atomic density distribution similar to that proposed for the Gaussian approximation potentials [29]

$$\rho_i(\mathbf{r}) = \sum_{j \neq i}^{\substack{\text{atoms } j \text{ within } R_c \\ \text{distance of atom } i}} \eta_j \delta\left(\mathbf{r} - \mathbf{R}_{ij}\right) f_c\left(\|\mathbf{R}_{ij}\|\right), \tag{16}$$

at any point $\mathbf{r} = (r, \theta, \phi)$, where δ is the Dirac delta distribution, $\mathbf{R}_{ij} = \mathbf{R}_j - \mathbf{R}_i$ is the position of atom j with respect to atom i. In order to gain permutation invariance, the constant, user-specified values η_j are considered the same for atoms j with identical chemical identities; for example, the atomic number of the element in question can be used for η_j. The atomic density distribution (16) is then scaled to fit into the unit sphere S^2, and the resulting scaled atomic density $\rho_i(\tilde{r}, \theta, \phi)$ is subsequently used as the expanded function $f(\tilde{r}, \theta, \phi)$ in Eq. (15) for computing Zernike moments (c_{nl}^m) in order to calculate components ($\|\mathbf{c}_{nl}\|$) of our Zernike fingerprint, $\mathbf{G}_i$. This fingerprint of atom i is input to the atom-centered scheme of Eq. (5). Fig. 6 plots variations of a number of components of the Zernike-type fingerprint vector $\mathbf{G}_i$ as functions of pair-atom distance, triple-atom valence angle, and quadruple dihedral angle. We emphasize that this descriptor is sensitive to dihedral angles – that is, quadruple-atom interactions – while the Gaussian descriptor is insensitive to this feature. (The bispectrum descriptor, described next, also has this feature.) However, a nice feature of Zernike descriptor which may make it preferable over the bispectrum is that Zernike moments can be explicitly represented in terms of monomials, which makes calculation of both moments themselves as well as their derivatives computationally much cheaper; note that calculation of derivatives of descriptor components is an essential step for evaluating atomic forces.

2.1.3. Bispectrum

In their development of the Gaussian Approximation Potentials (GAP), Bartók et al. [29,46] have used a bispectrum derived from four-dimensional (4-D) spherical harmonics as the translation- and rotation-invariant feature vector, in combination with Gaussian processes as the regression model. Here, we briefly describe this bispectrum as a general descriptor which can be paired with arbitrary learning models; that is, as a feature vector $\mathbf{G}_i$ of Eq. (5). (For a more detailed discussion of the development of this descriptor in relation to GAP, please see their original paper [29].) To begin, let us first extend the notion of the 2-sphere to 3-sphere (S^3) of unit radius as the set of points $\mathbf{x} \in \mathbb{R}^4$, with $\|\mathbf{x}\|^2 = \sum_{i=1}^{4} x_i{}^2 = 1$. Alternatively, points on S^3 can be described by hyper-spherical coordinates $\tilde{r} = 1$, $0 \leq \psi, \theta \leq \pi$, and $0 \leq \phi <$

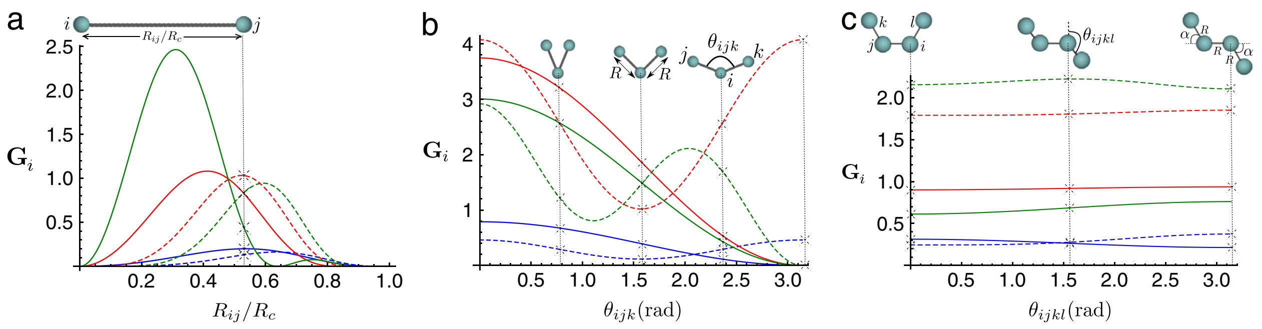

Fig. 6. Variations of several components of Zernike feature vector $\mathbf{G}_i$ of atom i with polynomial cutoff function (9) and $\gamma = 4$; $(n, l) = (1, 1)$ for blue solid curve, (2, 2) for blue dashed curve, (3, 1) for red solid curve, (4, 2) for red dashed curve, (5, 1) for green solid curve, and (5, 3) for green dashed curve; (a) versus pair-atom distance R_{ij}, (b) versus valence angle θ_{ijk} for $R = R_c/2$, and (c) versus dihedral angle θ_{ijkl} for $R = R_c/2$ and $\alpha = \pi/3$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2π which can be mapped to $\mathbf{x}$ coordinates as below:

$$\begin{aligned} x_1 &= \cos\psi, \\ x_2 &= \sin\psi\cos\theta, \\ x_3 &= \sin\psi\sin\theta\cos\phi, \\ x_4 &= \sin\psi\sin\theta\sin\phi. \end{aligned} \tag{17}$$

A point $\mathbf{r} = (x, y, z)$ in 3-D Euclidean space within the local environment ($\|\mathbf{r}\| < R_c$) can be mapped onto the surface of S^3 according to [47]

$$x_1 = \sqrt{1-(\|\mathbf{r}\|/R_c)^2}, \qquad x_2 = z/R_c, \\ x_3 = x/R_c, \qquad x_4 = y/R_c, \tag{18}$$

or

$$\psi = \arcsin\left(\|\mathbf{r}\|/R_c\right), \\ \theta = \arccos\left(z/\|\mathbf{r}\|\right), \\ \phi = \begin{cases} \arctan(y/x) & \text{if } x, y \geq 0, \\ \pi + \arctan(y/x) & \text{if } x < 0, \\ 2\pi + \arctan(y/x) & \text{if } x \geq 0,\ y < 0. \end{cases} \tag{19}$$

Consequently, rotation of the local atomic environment in 3-D Euclidean space corresponds to rotation on the surface of S^3. In this space, the inner product of two functions $f(\psi, \theta, \phi)$ and $g(\psi, \theta, \phi)$ can be defined as below:

$$\langle f(\psi,\theta,\phi), g(\psi,\theta,\phi)\rangle = \int_0^{\pi} d\psi \sin^2\psi \int_0^{\pi} d\theta \sin\theta \\ \times \int_0^{2\pi} d\phi f^*(\psi,\theta,\phi)\, g(\psi,\theta,\phi). \tag{20}$$

Then, 4-D spherical harmonics, $U^j_{m'm}(\psi, \theta, \phi)$ are the simultaneous eigenfunctions of quantum mechanical angular momentum operators $\hat{\mathbf{L}}^2$ and $\hat{\mathbf{L}}_z$, and form an orthogonal complete basis set for the Hilbert space Ω of square-integrable (L^2) functions defined on the surface of S^3 [48]:

$$\left\langle U^{j_1}_{m_1'm_1}(\psi,\theta,\phi), U^{j_2}_{m_2'm_2}(\psi,\theta,\phi)\right\rangle = \delta_{j_1j_2}\delta_{m_1'm_2'}\delta_{m_1m_2}. \tag{21}$$

Consequently, any member $f(\psi, \theta, \phi)$ of space Ω of L^2 functions can be expanded in terms of spherical harmonics as

$$f(\psi,\theta,\phi) = \sum_{j,m',m} c^j_{m'm} U^j_{m'm}(\psi,\theta,\phi), \tag{22}$$

with

$$c^j_{m'm} = \left\langle U^j_{m'm}(\psi,\theta,\phi), f(\psi,\theta,\phi)\right\rangle. \tag{23}$$

Bartók et al. [29,46] have shown that components of the bispectrum given by

$$B_{jj_1j_2} = \sum_{m_1',m_1=-j_1}^{j_1} c^{j_1}_{m_1'm_1} \sum_{m_2',m_2=-j_2}^{j_2} c^{j_2}_{m_2'm_2} \\ \times \sum_{m',m=-j}^{j} C^{jj_1j_2}_{mm_1m_2} C^{jj_1j_2}_{m'm_1'm_2'} \left(c^j_{m'm}\right)^*, \tag{24}$$

are invariant under rotation of $f(\psi, \theta, \phi)$ on the surface of S^3, where the C's are Clebsch–Gordan coefficients.

Now, if the atomic density distribution $\rho_i(\psi, \theta, \phi)$ ($\rho_i(\mathbf{r})$ of Eq. (16) mapped onto S^3 according to the transformations (19)) is used as the expanded function $f(\psi, \theta, \phi)$ of Eq. (22), then the corresponding bispectrum components (24) are invariant under rotation of $\rho_i(\mathbf{r})$ in 3-D Euclidean space. For a user-specified maximum value ($j_{\max}$) of j, components of the bispectrum feature vector $\mathbf{G}_i$ of *Amp* are computed from Eq. (24) with the following values for j_1, j_2, and j: $j_1 = j_2 = 0, 1/2, 1, \ldots, j_{\max}$ and $j = 0, 1, \ldots, \min(j_1 + j_2, j_{\max})$.

2.2. Machine-learning model

In our scheme, the *model* is a functional form containing adjustable parameters that can be regressed to link the fingerprints with the desired output properties, namely the energy and/or forces. The model can take many different forms – generally from the field of machine learning – such as Gaussian processes, kernel ridge regression, support vectors, or an artificial neural network [49]. In particular, the feed-forward class of artificial neural networks has seen the greatest application in computational chemistry, especially in the case of large training data sets; it will be discussed in the next section.

To be useful, the model needs to be trained to some data of interest. Training is basically adjusting the model parameters to fit the model to the inputs and outputs of the training data, which may come from experiments, computations, or any other source of data, by calibrating model parameters. In our case, electronic structure calculation energies (and forces, described below) provide the reference values, which have the qualities of being noiseless and can be produced on-demand (as opposed to experiment data for example). The process of fitting is often carried out by minimizing a loss function; a common choice is the sum of square residuals, where the atom-normalized residuals of the machine-learned energies $\hat{E}_j$ are compared to the training (electronic structure) energies E_j,

$$\Gamma = \frac{1}{2}\sum_{j=1}^{M}\left(\frac{\hat{E}_j}{N_j} - \frac{E_j}{N_j}\right)^2, \tag{25}$$

where M is the number of energy data points or atomic configurations (one energy data point per atomic configuration) and N_j is the number of atoms in the atomic configuration j. The squared-error loss function (25) can be minimized by adjusting model parameters. In the procedure of fitting the model, it is usually preferable to use as much data as is provided by quantum mechanical calculations. Arguably, the forces – which are gradients of the PES – provide much richer information than the energy points themselves; for example, a 30-atom image will have 90 unique force values which can be associated with the individual atoms, whereas it will have only a single potential energy. Therefore, atomic forces may also be incorporated into the loss function (25), as shown for example below:

$$\Gamma = \frac{1}{2}\sum_{j=1}^{M}\left\{\left(\frac{\hat{E}_j}{N_j}-\frac{E_j}{N_j}\right)^2+\frac{\alpha}{3N_j}\sum_{k=1}^{3}\sum_{i=1}^{N_j}\left(\hat{F}_{ik}-F_{ik}\right)^2\right\}, \qquad (26)$$

where F_{ik} and $\hat{F}_{ik}$ are the quantum mechanics and machine-learned forces on atom i in the direction k, and α is a constant with the dimension of length. The value of α determines the path of convergence, and ideally should be chosen such that both energy convergence and force convergence happen together. Half multipliers in Eqs. (25) and (26) are just for math simplification.

For the neural network model, the backpropagation method [49] can be used to efficiently calculate the gradient of the loss function with respect to the parameters; in conjunction with a variety of gradient-based optimization algorithms, this can be used to find optimal model parameters by minimizing the loss function (25) or (26). We note that the loss function for a neural network model is not convex and has multiple local minima, so the gradient-descent procedure is dependent upon the initial guess of parameters and will find a local minimum. Among these gradient-descent algorithms are steepest descent, conjugate gradient, Levenberg–Marquardt, extended Kalman filter, and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, the last of which is the default optimizer in *Amp*. As *Amp* calls upon the open-source software library *SciPy* [50] for the loss-function minimization, the user is free to specify any optimizer within the SciPy package, including global schemes such as basin hopping [51]. The user can also write or wrap their own custom optimizer, provided it follows the API of *SciPy* optimizers.

The final purpose of interpolating potentials is to predict the energetics of atomic configurations for which the potential has *not* been trained. Consequently, the question arises: How reliable are the predictions of the developed potential? To test the performance of the regression model against unseen atomic configurations, usually a new data set (called the "test data set") independent of the training data set is generated with known electronic structure energy and force values. The question is addressed with a single-dimension example in Fig. 7, where data points from an unknown function are divided into five training and one test point. The two curves are the interpolations after training the model where sum of squared deviations has been reduced significantly. It is seen that the red dashed curve "overfits", and does not generalize well to the test data as compared to the blue dotted curve. This shows that the value of the loss function on the training data set by itself is not enough to indicate how representative the fitted curve is. Basically, the non-smoothness of the regression model can increase in two ways, either by enlarging the norm of the vector of model parameters while keeping the number of them fixed, or by increasing the number of model parameters itself (*i.e.* model size). However, a highly non-smooth model is at risk of over-fitting; that is, of giving poor predictions of the test data set. To avoid the over-fitting phenomenon, the first approach [49] is to add a regularization penalty term $\beta\|\boldsymbol{\omega}\|^2$

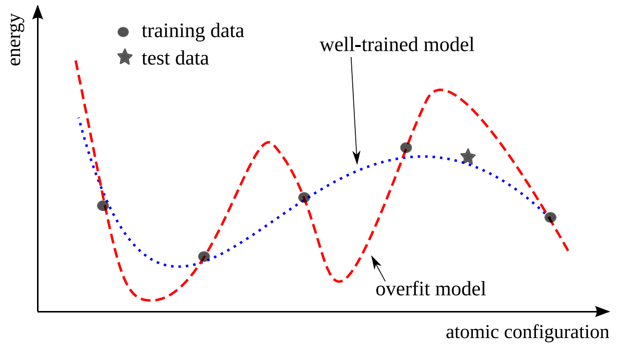

Fig. 7. Illustration of overfitting. The blue dotted line is a reasonable fit to the training data as well as a test data point which was not used for training. The red dashed line has been overfit, which can be observed via the large residual in the prediction of the test data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

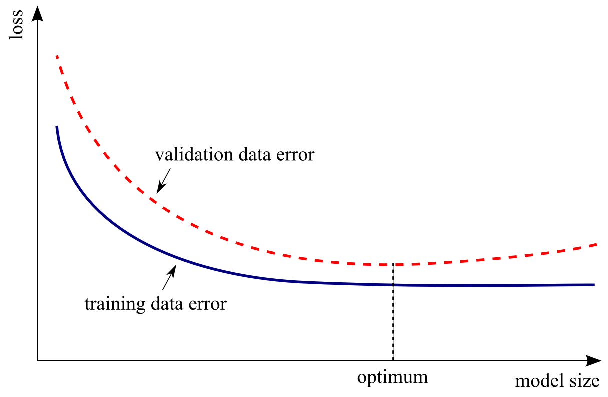

Fig. 8. Variations of the error on training and validation data sets with model size.

to the loss function ((25) or (26)) with $\boldsymbol{\omega}$ being the vector of model parameters, encouraging smoothness of interpolation. The regularization coefficient β is in general a positive number. The choice of a proper value for β is subtle, depends on the training data set, and can be decided with cross-validation; a small value may not cure the over-fitting issue, whereas a large value may cause over-smoothness. The second approach is to choose an optimum size for the model; large enough to make the PES flexible, satisfactorily fitting the quantum mechanics PES, while at the same time small enough to avoid unwanted ridges on the machine-learned PES in between training points of the quantum mechanics PES. In practice, this can be done by reserving a subset of the training data set (called "validation data set") for the purpose of validation of the model size. Fig. 8 shows variations of the loss function value corresponding to both training and validation data sets with the model size, *i.e.* number of adjustable parameters of the model. It illustrates that there exists a subtle optimum size of the model which gives the smallest value of error on the validation data.

Moreover, most electronic structure codes provide not just the energies, but also the forces on each atom (by the Hellmann–Feynman theorem [52]) for each atomic image calculated. These forces are derivatives of the potential energy function, and if they are also included in the training scheme as suggested in Eq. (26), can remarkably improve fitting of the machine-learning PES. This is illustrated for a simple example in Fig. 9, where the DFT PES of a three-atom gold cluster in the shape of an isosceles triangle with vertex angle $\theta = 1.3$ rad, is calculated with GPAW [34] and shown as a function of leg length l with gray solid curve. Fitting the already accessible quantum mechanics forces as well as quantum mechanics energies at the two points picked on the sides of the

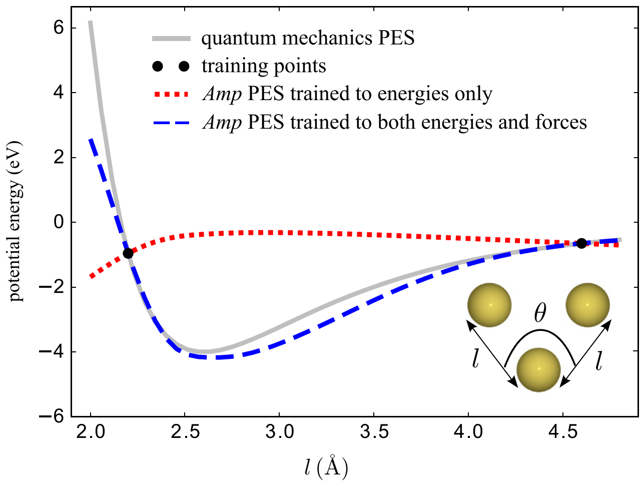

Fig. 9. Quantum mechanics PES corresponding to a three-atom gold cluster ($\theta = 1.3$ rad) calculated by GPAW [34] is plotted by gray solid curve, whereas *Amp* PES is plotted by red dotted curve (trained to only energies of two points) and blue dashed curve (trained to both energies and forces of the two points). Force training can substantially improve the *Amp* PES. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

curve, results in a much better interpolation (blue dashed curve of Fig. 9) than fitting only quantum mechanics energies as the same points (red dotted curve of Fig. 9).

2.2.1. Neural networks

A common machine-learning model suitable for large data sets is the artificial neural network (referred to here just as neural networks). Historically, the neural network was considered a very simple model of how the nervous system processes information. The first mathematical model was developed in 1943 by McCulloch and Pitts [53] for classification purposes; this considered biological neurons to either send or not send a signal to the neighboring neuron. The model was soon extended to do linear and nonlinear regression, by replacing the binary activation function with a continuous function. A schematic of a typical feed-forward neural network is shown in Fig. 10, where the basic functional units of the neural network are represented by nodes. A number of parallel nodes constitute a layer. A feed-forward neural network consists of at least an input layer plus an output layer. When approximating the PES, the output layer has just one node representing the potential energy. For a more robust interpolation, a number of "hidden layers", each having a number of nodes, may exist in the neural network as well; hidden refers to the fact that these layers have no physical meaning. In each node, a number of inputs are multiplied by the corresponding weights and summed up with a constant bias. An activation function then acts upon the summation and an output is generated. For example at node $N_{2,3}$ of Fig. 10, output $o_3^{(2)}$ is related to the inputs $o_i^{(1)}$ according to

$$o_3^{(2)} = f\left(\sum_{i=1}^{3} \omega_{i3}^{(2)} o_i^{(1)} + \omega_{43}^{(2)}\right), \tag{27}$$

where f is the activation function, $\omega_{43}^{(2)}$ is the bias, and other $\omega^{(2)}$'s are weights. The output is finally sent to the neighboring node in the next layer. Typically used activation functions are hyperbolic tangent, sigmoid, Gaussian, or a linear function. The unbounded linear activation function is particularly useful in the output node to scale neural network outputs to the range of reference values. For our purposes, the output of the neural network represents the energy of atomic system. The derivative of the neural network output with respect to inputs can be obtained by chain-rule differentiation [54]; in atomic systems, this is related to the atomic forces $\hat{F}_{ik}$, which can therefore be readily calculated with analytical expressions. Calculating derivatives of the neural network output with respect to adjustable parameters, *i.e.* weights and biases, is also straight-forward by chain-rule of calculus, and these derivatives can be used to accelerate gradient-based methods of loss-function minimization. Expressions for these derivatives are given in the supplementary material (see Appendix A).

It has been proven that a neural network with an infinite number of adjustable parameters can in principle approximate any function to arbitrary accuracy [55]. The fitting capability is of course limited in a finite-sized model, but can systematically be improved by increasing the size of the hidden-layer structure. A challenge of neural networks is that the loss function (25) or (26)

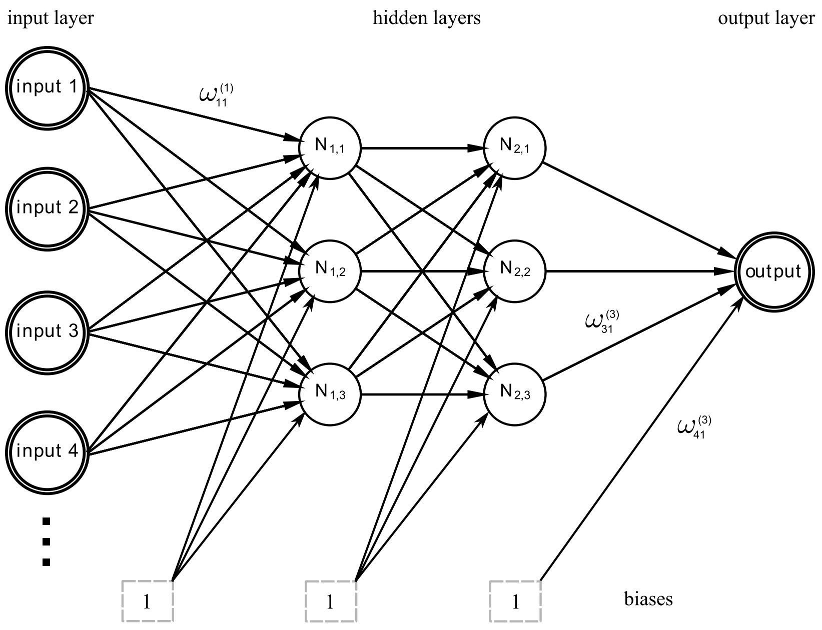

Fig. 10. Schematic of a feed-forward neural network with two hidden layers each having three nodes.

takes a non-convex functional form having multiple local minima. Subsequently, when starting with a random set of initial model parameters, the gradient-based optimizer might get trapped in a local minimum which is not deep enough to yield a satisfactorily small value of the loss function. Indeed, a global search scheme such as simulated annealing or genetic algorithm can be primarily overlaid to skip from the current local basin to a nearby deeper basin, which has been implemented as an option in *Amp*. It has been also suggested that pre-conditioning the inputs to the neural network can significantly accelerate its training [42,56].

To end up with an optimum number of hidden nodes (which depends on the training data set), a possible approach available inside *Amp* is to start with a small number of nodes in each hidden layer and train the network as accurately as possible. Further nodes can then be added sequentially to make the loss function value smaller in a continuous way. Ideally, this would result in an optimal-sized neural network; of course, the issue with local minima means this property is not guaranteed. In the context of neural networks, another approach to evaluate overfitting and possibly improve predictions is to constitute a "committee" [54] of neural networks and train all of them to the training data set independently. The committee can then decide whether to accept a prediction or reject it and improve neural network predictions. If the predictions of different neural networks are close enough, an average value can be accepted as the energies and forces of the first-seen data. On the other hand, if the predictions are too far from each other, then all are rejected and the first-seen data is added to the training data set for further fitting.

3. The *Amp* code

This section discusses generalities of working with *Amp*, as well as our coding philosophies deployed in its construction. Of course, detailed instructions exist and are kept updated in the package documentation at [57]. Generally, we have designed the *Amp* calculator to be as flexible and modular as possible by having the majority of the code and user interface in python, while enhancing speed for the computationally intensive portions via fortran modules that can be parallelized across nodes and cores. We have designed *Amp* such that the descriptor scheme and the regression scheme are separate from one another and thus can be specified independently, thus making it straightforward for the user to develop and implement new approaches. Additionally, all portions of the code can be run in pure python, facilitating rapid development, debugging, and testing. However, computationally intensive portions of the code have duplicate routines written in fortran, achieving a much higher computational throughput than their python counterparts. The fortran routines are automatically tested against their python counterparts in nightly builds, ensuring the output of the fast calculations matches those of the original calculator. To the author's knowledge, this open-source package represents the first publicly available machine-learning software package for atomistics. The project began under the name *Neural* [58] to replicate the pioneering methods of Behler, Parrinello, Artrith, and others; the name was subsequently switched to *Amp* [59,60] starting with version 0.3 in order to reflect the broader, modular nature of our approach that is not restricted only to neural networks or the Gaussian feature vectors. In the current version, we have developed a self-contained implementation of descriptors and the neural-network model which helps to minimize dependences and also provides a benchmark implementation, but since the package is modular we expect that it is naturally suited to future, more efficient implementations based on tensor libraries, for example.

We built the *Amp* calculator to be an ASE [32] calculator object; it thus contains all the normal methods inherent to ASE calculators, such as "get_potential_energy" and "get_forces". With the basic "get_forces" and "get_potential_energy" methods, this trained calculator can then be used just like any ASE electronic structure calculator (*e.g.* Dacapo, VASP, Gaussian, Quantum ESPRESSO, ...) to perform atomistic simulations such as molecular dynamics, optimizations, transition-state searches, *etc.*, but typically at much faster computational time than the parent calculator. Because of the flexible nature of ASE, this approach makes it simple to quickly switch between the parent electronic structure and *Amp* calculators in identical simulations, facilitating rapid assessment of the performance of the machine-learning calculator.

Importantly, *Amp* has an additional method not present in general ASE calculators: "train". This method is designed to be a simple interface to facilitate training the machine-learning model to specified input data; again, the broad compatibility of ASE with a variety of electronic structure and force-field codes makes it straightforward to use training data from the user's preferred calculator. After being trained, it can be used just like any other ASE calculator, *e.g.*, if calc is a trained instance of *Amp*, then the below code will call the energies and forces from the machine-learning calculator:

```
>>> atoms = ···
>>> atoms.set_calculator(calc)
>>> energy = atoms.get_potential_energy()
>>> forces = atoms.get_forces()
```

The trained calculator can readily be saved to and loaded from disk, making it simple and portable after training. Additionally, the code hashes the training images to avoid duplicating effort in training; this also enables us to specify a database of derived image quantities, such as fingerprints and fingerprint derivatives, which avoids having to recalculate these quantities in subsequent training runs. The following paragraphs give brief examples of how to import, initialize, and train the *Amp* calculator, based on the latest development version of the code as of this writing. Instructions for detailed use and package installation are reserved for the online documentation [57].

Importing and initializing. An untrained *Amp* calculator is initialized by specifying the descriptor and model to be used. For example, the Behler–Parrinello [27,28] scheme uses the Gaussian descriptor and neural network model; this could be established using default values for the arguments by the below:

```
from amp import Amp
from amp.descriptor.gaussian import Gaussian
from amp.model.neuralnetwork import NeuralNetwork

calc = Amp(descriptor=Gaussian(),
           model=NeuralNetwork())
```

Of course, the user can specify options when instantiating the `Gaussian` and `NeuralNetwork` classes, such as the characteristics of the fingerprint or the hidden-layer structure of the neural network.

Training. A machine-learning calculator must be trained to existing data to be useful. This can be called as simply as

```
calc.train('trainingimages.traj')
```

The argument to the train method is the set of images to be used for training. In the example it is the location of an ASE trajectory file. This can also be an ASE database file ".db" (recommended for large or inhomogeneous data sets), or just a python list of images. Upon calling train, a log file records the training progress, and

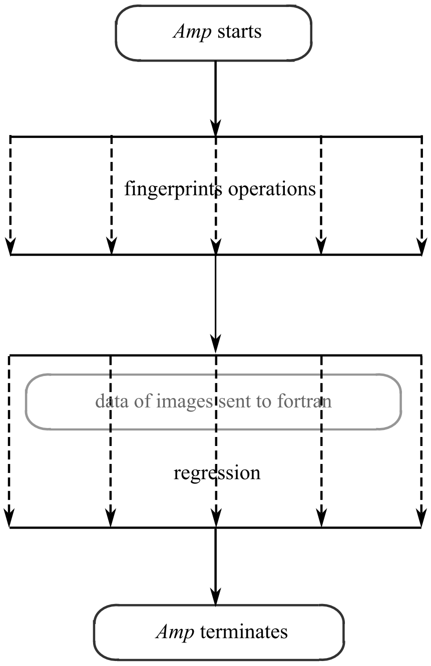

Fig. 11. Schematic of how parallel computing is designed in *Amp*.

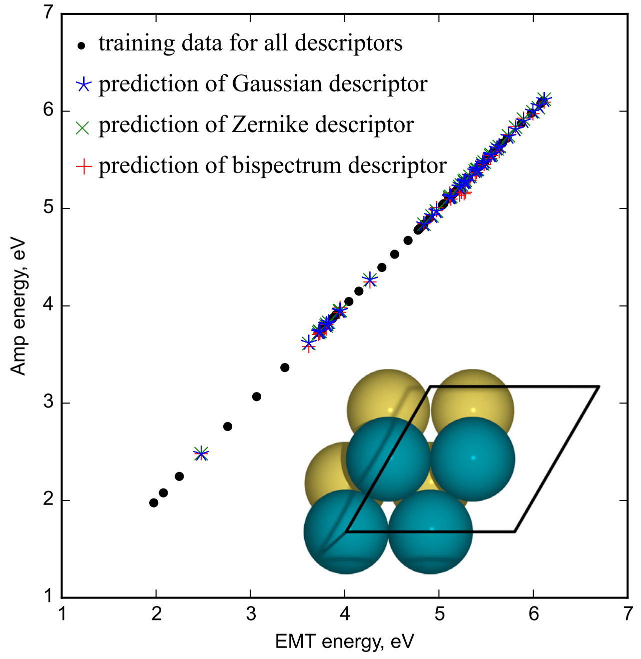

Fig. 12. Predictions of Gaussian, Zernike, and bispectrum descriptors of trained *Amp* calculators.

upon completion, the calculator parameters are saved to a user-specifiable file, by default "calc.amp". This can be re-loaded in future sessions as

```
calc = Amp.load('calc.amp')
```

High-performance and parallel computing. As the training data set grows in size, the computational requirements associated with fingerprinting images and training a calculator can become quite demanding. Fortunately, both fingerprinting and loss-function calculations can be formulated as "embarrassingly parallel" problems; that is, many cores can work independently to process their own sets of images. To accomplish parallelization, *Amp* starts with serial computing, but when arriving to calculating atomic fingerprints and their derivatives, parallel sub-tasks are carried out on each part of the atomic configurations. Generally, these intensive computational portions of the code also have fortran routines to accelerate the computations over the pure-python versions. Further when performing machine-learning regression in the train mode, parallel sub-tasks are spawned, each of which is responsible for a certain set of images; these sub-tasks then report back their components of the loss function (and its derivatives) to the master task. A schematic of how parallel computing is designed in *Amp* is shown in Fig. 11.

4. Demonstrations

To demonstrate the performance of *Amp*, two brief examples are considered in this section. The first example – utilizing images of a metal surface with a pseudomorphic overlayer – demonstrates the modularity of *Amp* by training the same data set with Gaussian, Zernike, and bispectrum descriptors. The second example – with gaseous species adsorbed on a metal surface – demonstrates training to a large data set with both energy and forces.

Example 1. A periodic system comprised of an fcc(111) gold surface with 2×2 atoms within the unit cell two layers deep is considered. The top layer is substituted by palladium to form a pseudomorphic monolayer structure. The system is given an initial Maxwell–Boltzmann distribution of momenta by setting the temperature to 6000 °K. Interactions between atoms are evaluated by the effective medium theory (EMT) [33] calculator within ASE [32], and the equations of motion are integrated over time by the velocity Verlet algorithm with time step of five fs, for the total time of 500 fs. As a result, a trajectory of 100 atomic configurations is generated. These images are then randomly divided into two bins of 60 images (training data set) and 40 images (test data set). We established *Amp* calculators with Gaussian (with default parameters η, ζ, and λ as given in Fig. 4), Zernike (with $n_{\max} = 7$), and bispectrum (with $j_{\max} = 3$) local environment descriptors. The number of Gaussian, Zernike, and bispectrum descriptor components are 20, 20, and 22, respectively. We employed the neural network model in atom-centered mode and were able to achieve a good fit to the data with just two hidden layers each having a single node, for this simple example. (We note that networks in general must be more complicated than this for practical applications.) Calculators are then trained until the residual energies were less than 0.1 meV per atom. Training data points can be seen in black color in Fig. 12. We then called the trained calculators to predict the energies of the test atomic configurations. As can be seen, the predictions of *Amp* with Gaussian, Zernike, and bispectrum descriptors agree well with the parent EMT energies, indicating the calculator could successfully learn from the parent data using a variety of feature vector schemes. To our knowledge, this is the first demonstrated use of a Zernike-type descriptor for atomistic machine-learning, and the first reported pairing of the bispectrum descriptor (developed for use with Gaussian processes [29]) with a neural-network model. Movies of the training and test data images exist in the supplementary material (see Appendix A).

Example 2. A large number of images of CO and H adsorbates on a Mo surface were prepared in planewave DFT for a previous study by Zhang et al. [61]; this is the system shown in Fig. 1. From this work, a data set of 1977 atomic configurations was extracted. See the earlier work for full electronic structure calculation details; they are briefly summarized here. Each atomic configuration is a periodic structure consisting of a 3×3 molybdenum bcc(110) slab three layers deep inside the unit cell. The number of adsorbed atoms varies between zero to one for hydrogen and zero to nine

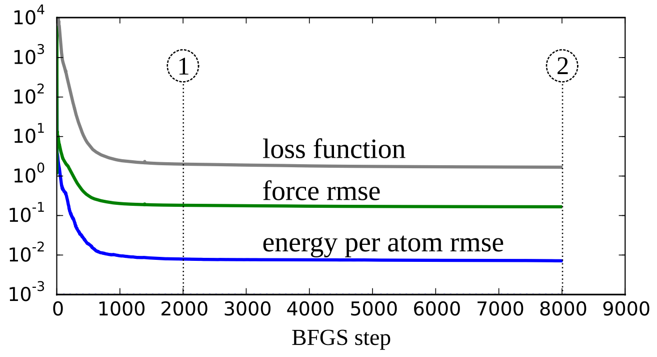

Fig. 13. Reduction of the value of the squared-error loss function as given by (26) with $\alpha = 0.04$ during BFGS optimization steps. Points ① and ② reference points discussed in the text.

for carbon monoxide. DFT energies and forces of atomic systems were calculated with the Dacapo [35] calculator and the revised Perdew–Burke–Ernzerhof (RPBE) [62] exchange–correlation functional. The data set was generated using the constrained minima hopping technique [63], which ensures a range of atomic configurations corresponding to regions around local minima and crossing between minima. We randomly divided the images into a training data set of 1497 images and a test data set of 480 images. Next, we trained *Amp* with the Gaussian descriptor, modeling energetics of the local environments of 6.5 Å radius by Eqs. (6) and (7) with default values for η, λ, and ζ, and the cosine cutoff function (8). We employed the tanh activation function and three hidden layers, each having ten nodes, in our neural networks corresponding to each chemical species. A preliminary global search of 4000 random walks in the space of neural network parameters was carried out to find best parameters minimizing the loss function (26). 8000 Steps of BFGS line search, starting from the best parameters found in the global search scheme, were then performed during the training process. Fig. 13 shows the reduction of the value of squared-error loss function (26) as well as energy per atom and force root-mean-square error (rmse) with BFGS optimization steps. With the current version of the code, the global and local parameter search along with fingerprinting operations took about 600 CPU hours with a total memory of 125 GB to bring the energy per atom rmse and force rmse down to 7.884 meV/atom and 0.182 eV/Å, respectively (point ① on the figure). A further decrease to 7.153 meV/atom and 0.166 eV/Å (point ②) took about 1000 CPU hours more suggesting a limit to the goodness-of-fit caused either by a local minimum or the model characteristics. (We are continuously working to further improve these performance metrics.) This compares favorably to the pioneering work of Artrith and Kolpak [64], which fit a four-element system to residual energies of 8.2 meV/atom and did not fit forces. To the authors knowledge, our current example is the first report on developing a machine-learning potential for a quaternary system, trained to both energies as well as forces. Blue spots of Fig. 14(a) show how well the learned potential mimics Dacapo energies and forces of the training data set. In order to validate the developed potential, Dacapo as well as *Amp*-predicted energetics for the test data images, not already seen by *Amp*, are further studied. Root-mean-squared deviations between Dacapo and *Amp*-predicted energies per atom is 7.170 meV, and root-mean-squared deviations between Dacapo and *Amp*-predicted forces is 0.208 eV/Å for the test data set; these data points are shown in Fig. 14(b) by green spots. The good agreement between quantum mechanics and machine-learning energies and forces shows that the trained parameters can be confidently used for systems having similar local environments. Movies of training and test data images for this example are included in the supplementary material as well (see Appendix A).

5. Conclusion

Electronic structure calculations are now broadly used to understand mechanisms underlying nanoscale phenomena. These calculations can be produced on-demand, and are moreover noiseless. These properties make electronic structure calculations ideal to be complemented by machine learning for accelerated throughput. Based on this idea, we have developed the open-source Atomistic Machine-learning Package (*Amp*) to be as follows:

1. As intuitive and modular as possible with independent descriptor and model classes. The user can also create these two components at will. An intelligent choice of descriptor properly modeling the total energy (in image-centered mode) or the atomic energy (in atom-centered mode) of the atomic system, can favorably lead to interpolating more training atomic images with fewer number of model parameters. We have demonstrated good performance with three descriptors that are built into *Amp*; namely, Gaussian, Zernike, and bispectrum. As for the model class of *Amp*, it can take a variety of machine-learning regression models, *e.g.* neural networks which scale linearly with the number of training images, or Gaussian processes which scale non-linearly with the number of training images but produce uncertainty intervals as well. The loss function can be specified by the user to include forces, as opposed to only energies, which can improve the fit.
2. A python-based Atomic Simulation Environment (ASE) calculator. This enables interactive collaboration between an *Amp* machine-learning calculator and any electronic structure calculator inside the widely-used ASE. As a result, the underlying potential energy surface of an atomic system can be readily approximated by *Amp* supervised learning of energies/forces of a set of images, evaluated by an ASE electronic structure calculator. Moreover, active learning of the potential energy surface is possible by interactively demanding energy and force calls from the ASE electronic structure calculator. This approach can accelerate electronic structure searches for stable clusters or transition states, for instance.

Historically, there has been a separation between the creation of interatomic potentials and their use; for example, one research group will create a new model and disseminate it to the research community. Users then apply the potential to new situations, sometimes in regions where the potential's performance may be questionable. As opposed to this pattern, the approach in *Amp* allows users to directly create their own models. A key aspect of machine learning is that a model is improved by more experiences; *Amp* facilitates this by making it simple for users to validate their model predictions; if they find regions where the model fails in its approximation of the true PES, these data points can easily be added to the training set, which will systematically improve the model in the region of need.

Acknowledgments

The authors thank Dr. Nongnuch Artrith (MIT) for an inspirational introduction to atomistic machine learning through the Behler–Parrinello scheme, and to Professor Pedro Felzenszwalb (Brown), Dr. Sobhan Naderi Parizi (Google), Habibeh Ashouri-Choshali (Sharif University of Technology), Dr. Albert Bartók-Pártay (Cambridge), Rune Christensen (DTU), Dr. Ulrik Grønbjerg Vej-Hansen (DTU), Rohan Yuri Sanspeur (Brown), Dr. Zachary Ulissi (Stanford), and others for fruitful technical discussions as we developed this project. The authors are grateful for financial support for A.Kh. from ARO MURI project Award W911NF-11-1-0353, and for A.A.P. through ONR Award N000014-15-1-2223.

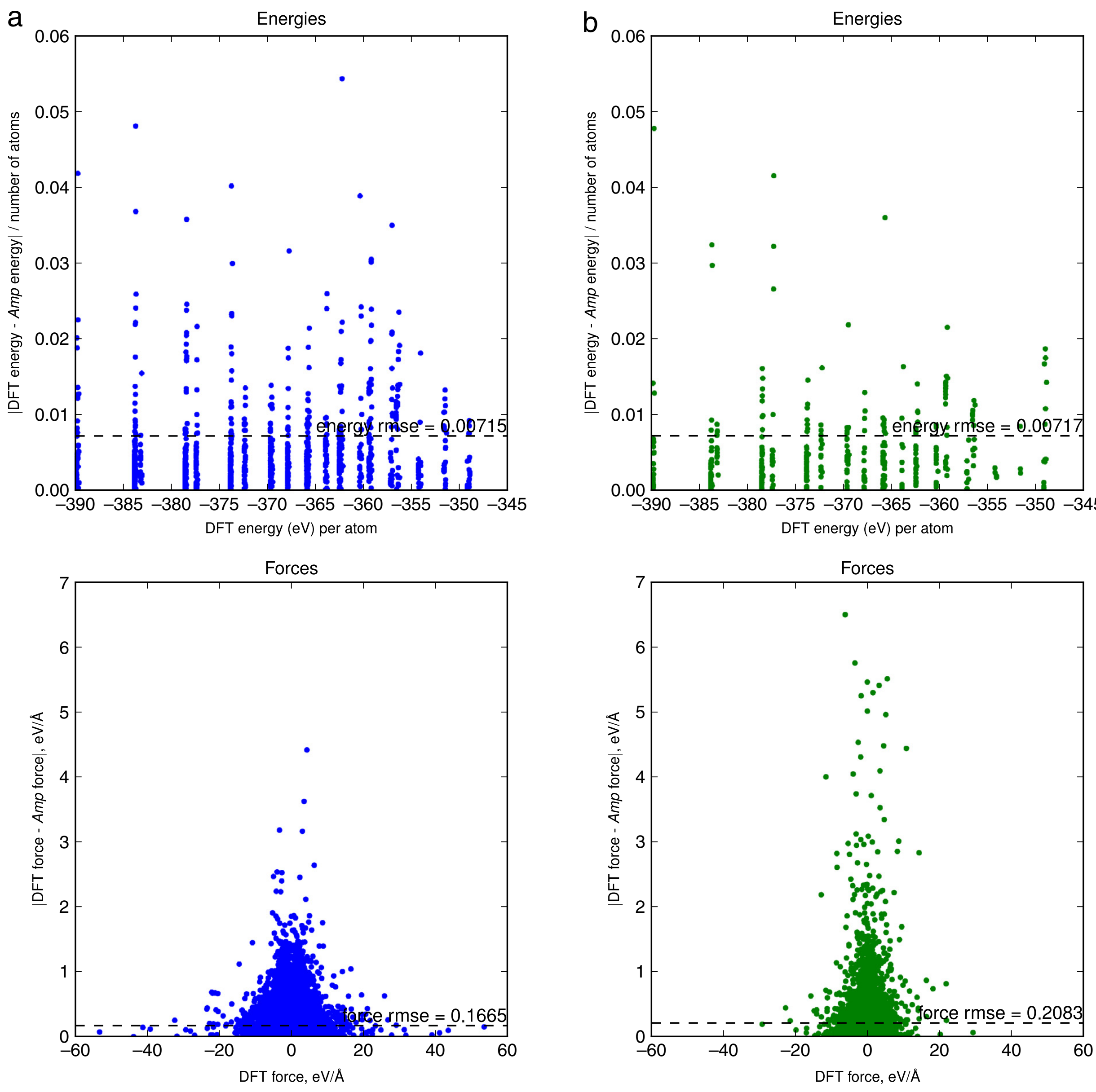

Fig. 14. Plots of discrepancies between Dacapo DFT energies and forces and machine-learning *Amp*-predicted energies and forces for (a) training data set and (b) test data set. Different clusters in the energy plots correspond to different number of adsorbates on the molybdenum slab.

High-performance computational work was carried out at Brown's Center for Computation & Visualization (CCV).

Appendix A. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2016.05.010.

References

[1] M. Born, R. Oppenheimer, Ann. Phys. 389 (1927) 457–484.
[2] H.B. Schlegel, Wiley Interdisciplinary Reviews: Computational Molecular Science 1 (2011) 790–809.
[3] S.J. Moss, C.J. Coady, J. Chem. Educ. 60 (1983) 455.
[4] T. Keyes, Normal Mode Analysis: Theory and Applications to Biological and Chemical Systems, Chapman and Hall/CRC, 2006.
[5] D. Marx, J. Hutter, Ab Initio Molecular Dynamics: Theory and Implementation. NIC. Forschungszentrum 2000.
[6] T.C. Scott, M. Aubert-Frecon, J. Grotendorst, Chem. Phys. 324 (2006) 323–338.
[7] C.J. Cramer, Essentials of Computational Chemistry: Theories and Models, Wiley, ISBN: 9780470091838, 2005.
[8] R.O. Jones, Rev. Modern Phys. 87 (2015) 897–923.
[9] Hunter Utkov, M.C. Maura Livengood, Chapter 7 - Using Density Functional Theory Methods for Modeling Induction and Dispersion Interactions in Ligand–Protein Complexes, in: Annual Reports in Computational Chemistry, vol. 6, Elsevier, 2010, pp. 96–112.
[10] A. Raval, S. Piana, M.P. Eastwood, D.E. Shaw, Protein Science 25 (2016) 19–29.
[11] N. Lu, M. Khorshidi, J. Geotech. Geoenviron. Eng. 141 (2015) 04015032.
[12] K. Yan, T.A. Maark, A. Khorshidi, V.A. Sethuraman, A.A. Peterson, P.R. Guduru, Angew. Chem., Int. Ed. 55 (2016) 6175–6181.
[13] L. Malekmotiei, A. Samadi-Dooki, G.Z. Voyiadjis, Macromolecules 48 (2015) 5348–5357.
[14] M. Krauss, F.H. Mies, J. Chem. Phys. (1965) 2703–2708.
[15] D.R. McLaughlin, D.L. Thompson, J. Chem. Phys. 59 (1973) 4393–4405.
[16] D.G. Truhlar, C.J. Horowitz, J. Chem. Phys. 68 (1978) 2466–2476.
[17] J. Ischtwan, M.A. Collins, J. Chem. Phys. 100 (1994) 8080–8088.
[18] T. Hollebeek, T.S. Ho, H. Rabitz, Annu. Rev. Phys. Chem. 50 (1999) 537–570.
[19] G.G. Maisuradze, D.L. Thompson, A.F. Wagner, M. Minkoff, J. Chem. Phys. 119 (2003) 10002–10014.
[20] S. Lorenz, A. Groß, M. Scheffler, Chem. Phys. Lett. 395 (2004) 210–215.
[21] S. Lorenz, M. Scheffler, A. Groß, Phys. Rev. B 73 (2006) 115431.
[22] B.J. Braams, J.M. Bowman, Int. Rev. Phys. Chem. 28 (2009) 577–606.
[23] M. Malshe, R. Narulkar, L.M. Raff, M. Hagan, S. Bukkapatnam, P.M. Agrawal, R. Komanduri, J. Chem. Phys. 130 (2009) 184102.
[24] L.M. Raff, R. Komanduri, M. Hagan, S.T. Bukkapatnam, Neural Networks in Chemical Reaction Dynamics, first ed., Oxford University Press, 2012.
[25] A.C.T. van Duin, S. Dasgupta, F. Lorant, W.A. Goddard III, J. Phys. Chem. A 105 (2001) 9396–9409.

[26] A. Ostadhossein, E.D. Cubuk, G.A. Tritsaris, E. Kaxiras, S. Zhang, A.C.T. van Duin, Phys. Chem. Chem. Phys. 17 (2015) 3832–3840.
[27] J. Behler, M. Parrinello, Phys. Rev. Lett. 98 (2007) 146401.
[28] N. Artrith, T. Morawietz, J. Behler, Phys. Rev. B 83 (2011) 153101.
[29] A.P. Bartók, M.C. Payne, R. Kondor, G. Csányi, Phys. Rev. Lett. 104 (2010) 136403.
[30] *Amp* lives at https://bitbucket.org/andrewpeterson/amp.
[31] J.R. Boes, M.C. Groenenboom, J.A. Keith, J.R. Kitchin, Int. J. Quantum Chem. 116 (2016) 979–987.
[32] ASE is an open-source software package available at https://wiki.fysik.dtu.dk.
[33] K.W. Jacobsen, P. Stoltze, J.K. Nørskov, Surf. Sci. 366 (1996) 394–402.
[34] J. Enkovaara, C. Rostgaard, J.J. Mortensen, J. Chen, M. Dułak, L. Ferrighi, J. Gavnholt, C. Glinsvad, V. Haikola, H.A. Hansen, H.H. Kristoffersen, M. Kuisma, A.H. Larsen, L. Lehtovaara, M. Ljungberg, O. Lopez-Acevedo, P.G. Moses, J. Ojanen, T. Olsen, V. Petzold, N.A. Romero, J. Stausholm-Møller, M. Strange, G.A. Tritsaris, M. Vanin, M. Walter, B. Hammer, H. Häkkinen, G.K.H. Madsen, R.M. Nieminen, J.K. Nørskov, M. Puska, T.T. Rantala, J. Schiøtz, K.S. Thygesen, K.W. Jacobsen, J. Phys.: Condens. Matter 22 (2010) 253202.
[35] S.R. Bahn, K.W. Jacobsen, Comput. Sci. Eng. 4 (2002) 56–66.
[36] J. Hafner, J. Comput. Chem. 29 (2008) 2044–2078.
[37] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, Comput. Phys. Comm. 181 (2010) 1477–1489.
[38] 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 (2009) 395502.
[39] M. Malshe, L.M. Raff, M. Hagan, S.T. Bukkapatnam, R. Komanduri, J. Chem. Phys. 132 (2010) 204103.
[40] A.Y. Toukmaji Jr., J.A.B, Comput. Phys. Comm. 95 (1996) 73–92.
[41] J. Behler, S. Lorenz, K. Reuter, J. Chem. Phys. (2007) 127.
[42] J. Behler, J. Chem. Phys. 134 (2011) 074106.
[43] N. Canterakis, 3D Zernike moments and Zernike affine invariants for 3D image analysis and recognition, in: 11th Scandinavian Conference on Image Analysis. pp. 85–93.
[44] M. Novotni, R. Klein, Comput. Aided Des. 36 (2004) 1047–1062. Solid Modeling Theory and Applications.
[45] C.H. Teh, R.T. Chin, IEEE Trans. Pattern Anal. Mach. Intell. 10 (1988) 496–513.
[46] A.P. Bartók, R. Kondor, G. Csányi, Phys. Rev. B 87 (2013) 184115.
[47] A.V. Meremianin, J. Phys. A: Math. Gen. 39 (2006) 3099.
[48] D.A. Varshalovich, A.N. Moskalev, V.K. Khersonskii, Quantum Theory of Angular Momentum, World Scientific, Singapore, 1988.
[49] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[50] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python 2001–. [Online; accessed 2016-02-16].
[51] D.J. Wales, J.P.K. Doye, J. Phys. Chem. A 101 (1997) 5111–5116.
[52] R.P. Feynman, Phys. Rev. 56 (1939) 340–343.
[53] W.S. McCulloch, W.H. Pitts, Bull. Math. Biophys. 5 (1943) 115–133.
[54] Y.H. Hu, Handbook of Neural Network Signal Processing, first ed., CRC Press, Inc., Boca Raton, FL, USA, ISBN: 0849323592, 2000.
[55] K. Hornik, M. Stinchcombe, H. White, Neural Netw. 2 (1989) 359–366.
[56] M.J. Perez-Ilzarbe, International Joint Conference on Neural Networks, vol. 2, 1999, pp. 1384–1388.
[57] *Amp* documentation lives at http://amp.readthedocs.io.
[58] A. Khorshidi, A. Peterson, Zenodo (2014). http://dx.doi.org/10.5281/zenodo.12665.
[59] A. Khorshidi, A. Peterson, Zenodo (2015). http://dx.doi.org/10.5281/zenodo.20636.
[60] A. Khorshidi, A. Peterson, Zenodo (2016). http://dx.doi.org/10.5281/zenodo.46737.
[61] Y.J. Zhang, V. Sethuraman, R. Michalsky, A.A. Peterson, ACS Catal. 4 (2014) 3742–3748.
[62] B. Hammer, L.B. Hansen, J.K. Nørskov, Phys. Rev. B 59 (1999) 7413–7421.
[63] A.A. Peterson, Top. Catalysis 57 (2014) 40–53.
[64] N. Artrith, A.M. Kolpak, Nano Lett. 14 (2014) 2670–2676.