Neural Networks 161 (2023) 154–164

Contents lists available at ScienceDirect

Neural Networks

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

A feedforward unitary equivariant neural network

Pui-Wai Ma [a,*], T.-H. Hubert Chan [b]

[a] United Kingdom Atomic Energy Authority, Culham Science Centre, Abingdon, OX14 3DB, United Kingdom
[b] Department of Computer Science, The University of Hong Kong, Hong Kong

ARTICLE INFO

Article history:
Received 9 August 2022
Received in revised form 23 January 2023
Accepted 27 January 2023
Available online 1 February 2023

Keywords:
Equivariant neural network
Feedforward neural network
Unitary equivariant
Rotational equivariant

ABSTRACT

We devise a new type of feedforward neural network. It is equivariant with respect to the unitary group $U(n)$. The input and output can be vectors in $\mathbb{C}^n$ with arbitrary dimension n. No convolution layer is required in our implementation. We avoid errors due to truncated higher order terms in Fourier-like transformation. The implementation of each layer can be done efficiently using simple calculations. As a proof of concept, we have given empirical results on the prediction of the dynamics of atomic motion to demonstrate the practicality of our approach.

1. Introduction

Neural Networks (NN) have gained popularity in many different areas because of its universal approximator property (Sonoda & Murata, 2017). In recent years, equivariant NN (ENN) in different architectures have been applied in various areas, such as 3D object recognition (Esteves et al., 2020; Thomas et al., 2018), molecule classification (Weiler et al., 2018), interatomic potential development (Batzner et al., 2022; Kondor, 2018), and medical images diagnosis (Müller et al., 2021; Winkels & Cohen, 2018; Worrall & Brostow, 2018).

When NNs are employed to model some physical phenomena, they should obey certain physical symmetry rules. For example, if an NN is intended to return some potential function between particles, the output should be *invariant* with respect to rotation of input particles' coordinates. On the other hand, for an NN predicting particle movements, the output should be *equivariant* with respect to rotation, i.e., if a rotation operator is applied to the input particles' coordinates, the effect is the same as applying the same rotation operator to the output.

In some works (Brandstetter et al., 2022), equivariance is achieved through data augmentation, i.e., additional training data are created by transforming existing training data (e.g., create additional copies by rotation). However, if equivariance is implemented in an NN, one can avoid the need of data augmentation, which reduces the demand on storage and improves sampling efficiency. This is especially important if one is working on data in continuous space. For example, if input data are points in Euclidean space and the output data are translational and/or rotational equivariance or invariance, it is not practical to create too many copies of data.

Previous works have achieved equivariance via higher order representations for intermediate network layers. For example, the implementation of spherical symmetry, such as S^2 or $SO(3)$, can be achieved through a layer with kernel performing a 3D convolution with spherical harmonics or Wigner D-matrices (Gerken et al., 2021; Thomas et al., 2018). This is analogous to Fourier transforms in linear space. However, these kinds of implementation are computationally expensive (Cobb et al., 2021).

In physical systems, although they can in principle be described by physical rules, analytical methods are not always feasible when the analytical form (such as the Hamiltonian) is unknown. On the other hand, an NN consists of many computationally simple components that can operate in parallel, and hence, they are suitable for large scale complicated simulations, as long as there is enough training data.

Our contributions. We have designed a new framework for feedforward neural networks. Specifically, it has the following properties.

1. The inputs and outputs are vectors in $\mathbb{C}^n$ with a complex number in each coordinate. Our neural networks are equivariant with respect to the unitary group $U(n)$.
2. In each layer, in addition to a linear combination of vectors from the previous layer, we have an extra term that is a linear combination of the normalized vectors as well. This extra term acts like the bias term in an affine transformation.
3. Each layer has an activation function that acts on vectors in $\mathbb{C}^n$ and is also equivariant with respect to unitary operators.

* Corresponding author.
E-mail addresses: Leo.Ma@ukaea.uk (P.-W. Ma), Hubert@cs.hku.hk (T.-H.H. Chan).

https://doi.org/10.1016/j.neunet.2023.01.042
0893-6080/

4. Equivariance is achieved in a feedforward neural network without any convolution layer.

Moreover, in Section 4, we have performed numerical experiments on the simulation of a physical system using our ENN framework in the scenario when the rules governing the system might be unknown.

2. Related works

We compare our framework with previous approaches on equivariant neural networks.

Kondor and Trivedi (2018) proved analytically that convolutional structure is a necessary and sufficient condition for equivariance to the action of a compact group. Therefore, many works designed the architecture of their NN based on this theorem, where convolution layer is introduced. Cohen and Welling (2016) introduced group equivariant convolution network. They used features map functions on discrete group, and so it only works with respect to finite symmetry groups.

Cohen et al. (2018) considered convolution NN for spherical images through Fourier analysis using Wigner D-matrices. Kondor et al. (2018) improved the implementation using Clebsch–Gordan decomposition, where the NN is operated in Fourier space only. It avoids the need of switching back and forth between Fourier and real spaces.

Thomas et al. (2018) shows if the input and output of each layer is a finite set of points in $\mathbb{R}^3$ and a vector in a representation of $SO(3)$, one can decompose this into irreducible representation through convolution with spherical harmonics and Wigner D-matrices. Esteves et al. (2020) implements exact convolutions on the sphere using spherical harmonics. It maps spherical features of a layer to the spherical features of another layers.

Convolution using spherical harmonics is analogous to Fourier transform in signal processing. In practice, it only preserves the most significant coefficients. Error is inherently introduced due to truncated higher order terms. It is also computation demanding due to the need of performing integration or summation.

Our newly designed feedforward neural network guarantees equivariance without any convolution layer. We should note our ENN has a structure in vector form which is different from the conventional NN structure in scalar form that was considered by Kondor and Trivedi (2018). Besides, our implementation is much simpler than previous works.

Satorras et al. (2021) devised an equivariant graph NN (GNN) with respect to $E(n)$ operators (that include rotation, reflection and translation). Similar to our approach, it does not contain convolution layer. The input spatial coordinates are vectors. Due to the construction of a GNN, their spatial coordinates are not filtered by activation functions. Their spatial coordinates are updated through averaging with respect to neighbors. The number of nodes in each layer is restricted to be the same, where our approach is general enough to allow different numbers of nodes in different layers. In addition to the spatial coordinates on which the operators act, their GNN contains feature vectors which do not fall under the equivariant aspect. We will also discuss how to add these extra features in our approach.

3. Our framework for ENNs

3.1. Equivariance with respect to unitary operators

In general, given a function $\phi : \mathcal{X} \to \mathcal{Y}$ (where the domain $\mathcal{X}$ and the co-domain $\mathcal{Y}$ might be different) and a group G, we assume that each element $g \in G$ induces *group actions* $T_g : \mathcal{X} \to \mathcal{X}$ and $\widehat{T}_g : \mathcal{Y} \to \mathcal{Y}$ on $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then, the function ϕ is *equivariant* under G if for all $g \in G$ and $x \in \mathcal{X}$, the following holds:

$$\phi(T_g(x)) = \widehat{T}_g(\phi(x)). \tag{1}$$

Formally, a group action needs to satisfy $T_{g_1 g_2} = T_{g_1} \circ T_{g_2}$ for all $g_1, g_2 \in G$.

Invariant is the special case when for all $g \in G$, the group action $\widehat{T}_g$ is the identity function on $\mathcal{Y}$.

In this paper, we consider domains of the form $\mathbb{C}^{n\times M}$, which we interpret as M points in $\mathbb{C}^n$. $\mathbb{C}$ is the set of complex number. We consider the unitary group $U(n)$, where each element corresponds to a unitary operator $\mathcal{U}$ on $\mathbb{C}^n$. An $n \times n$ matrix $\mathcal{U}$ is said to be unitary if its column vectors form an orthonormal set in $\mathbb{C}^n$. The unitary group contains the orthogonal group $O(n)$ (that corresponds to rotations and reflections) and $SO(n)$ (that corresponds to rotations only).

Given a unitary operator $\mathcal{U} : \mathbb{C}^n \to \mathbb{C}^n$, the group action on M points are defined by

$$(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}) \mapsto (\mathcal{U}\mathbf{x}^{(1)}, \mathcal{U}\mathbf{x}^{(2)}, \ldots, \mathcal{U}\mathbf{x}^{(M)}). \tag{2}$$

3.2. Structure of our ENN

We construct a feedforward neural network with $L-1$ hidden layers. The input layer is labeled as the 0th layer, and the output layer is the Lth layer. For the $(k+1)$th layer, its input is from the kth layer:

$$\mathbf{x}_k \in \mathbb{C}^{n\times M_k}, \tag{3}$$

where M_k is the number of vector elements of

$$\mathbf{x}_k = \{\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}, \ldots, \mathbf{x}_k^{(M_k)}\}. \tag{4}$$

Each vector element $\mathbf{x}_k^{(\alpha)} \in \mathbb{C}^n$ is an n-dimensional vector. Similarly, we have the output

$$\mathbf{x}_{k+1} \in \mathbb{C}^{n\times M_{k+1}}. \tag{5}$$

We define a variable

$$\mathbf{y}_k = \mathbf{x}_k\mathbf{W}_k + \mathbf{e}_k\mathbf{b}_k. \tag{6}$$

The weight and bias parameters matrices

$$\mathbf{W}_k \in \mathbb{C}^{M_k\times M_{k+1}} \tag{7}$$

$$\mathbf{b}_k \in \mathbb{C}^{M_k\times M_{k+1}} \tag{8}$$

and

$$\mathbf{e}_k = \left\{ \frac{\mathbf{x}_k^{(1)}}{\|\mathbf{x}_k^{(1)}\|}, \frac{\mathbf{x}_k^{(2)}}{\|\mathbf{x}_k^{(2)}\|}, \ldots, \frac{\mathbf{x}_k^{(M_k)}}{\|\mathbf{x}_k^{(M_k)}\|} \right\}, \tag{9}$$

where $\|.\|$ is the norm of an n-dimensional vector.

This definition is different from conventional feedforward NN. First, the $\mathbf{x}_k$ is a matrix and is put on the left-hand side of the weight parameter. Second, a new matrix variable $\mathbf{e}_k$ is introduced. These two changes are crucial steps to avoid the need to perform convolution. It allows the unitary operator getting out naturally from the left-hand side of $\mathbf{x}_k$ and $\mathbf{e}_k$. This is critical because matrix multiplication is in general non-commuting. If $\mathbf{x}_k$ is put on the right-hand side of $\mathbf{W}_k$, since $\mathbf{W}_k\mathcal{U}\mathbf{x}_k \neq \mathcal{U}\mathbf{W}_k\mathbf{x}_k$, our proof below becomes invalid.

Observe that for any unitary operator $\mathcal{U}$, it holds that

$$\|\mathbf{x}_k^{(\alpha)}\| = \|\mathcal{U}\mathbf{x}_k^{(\alpha)}\|. \tag{10}$$

For all $\alpha \in \{1, 2, \ldots, M_k\}$, we can obtain

$$\mathbf{y}_k(\mathcal{U}\mathbf{x}_k) = \mathcal{U}\mathbf{x}_k\mathbf{W}_k + \mathcal{U}\mathbf{e}_k\mathbf{b}_k = \mathcal{U}\mathbf{y}_k(\mathbf{x}_k), \tag{11}$$

where $\mathcal{U}$ is applied element-wise on each $\mathbf{x}_k^{(\alpha)}$.

Then, we define an activation function

$$\boldsymbol{\sigma}_{k+1}(\mathbf{y}_k) = \mathbf{x}_{k+1}. \tag{12}$$

for the $(k+1)$th layer. The activation function acts on each vector $\mathbf{y}_k^{(\alpha)} \in \mathbb{C}^n$ in element-wise manner. We note $\mathbf{y}_k$ has the same dimension of $\mathbf{x}_{k+1}$.

We shall find an activation function that satisfies the following:

$$\boldsymbol{\sigma}_{k+1}(\mathcal{U}\mathbf{y}_k) = \mathcal{U}\boldsymbol{\sigma}_{k+1}(\mathbf{y}_k). \tag{13}$$

This completes the construction of our feedforward equivariant neural network for unitary transformations.

Observe that each layer is equivariant with respect to unitary operators in the sense of Eq. (1). The reason is that if we transform the input $\mathbf{x}_k \to \mathcal{U}\mathbf{x}_k$, then its output will undergo the transformation $\mathbf{x}_{k+1} \to \mathcal{U}\mathbf{x}_{k+1}$; in this case, the unitary operator can act element-wise on both the input and the output spaces. Therefore, when we apply the group action, which is now the unitary operator $\mathcal{U}$, on the 0th layer input $\mathbf{x}_0$, the same operator will propagate to the final layer output $\mathbf{x}_L$. It means if we put $\mathbf{x}_0 \to \mathcal{U}\mathbf{x}_0$, the output will become $\mathbf{x}_L \to \mathcal{U}\mathbf{x}_L$. This completes the proof.

We note that if we consider the transposes of all variables and parameters from Eqs. (6) to (13), one can write $\mathbf{y}_k^T = \mathbf{W}_k^T\mathbf{x}_k^T + \mathbf{b}_k^T\mathbf{e}_k^T$. This resembles the form in conventional neural network, noting the difference in the bias term. In this case, one should apply the transpose of the unitary operator $\mathcal{U}^T$ on the right-hand side. Finally, $\mathcal{U}^T$ will come out on the right-hand side of Eq. (13). Although the application of an operator on either side is a matter of preference, it is more common to consider an operator on the left-hand side in physical sciences. For example, in quantum mechanics, a quantum state is more often to be written in a ket $|\Psi\rangle$ form, rather than in a bra $\langle\Psi|$ form, within the Dirac notation. Therefore, when describing an operation on a quantum state, it is more common to write $\mathcal{U}|\Psi\rangle$.

A possible choice of the activation function for each element can be a softsign function with a small residue, that is

$$\boldsymbol{\sigma}(\mathbf{u}) = \frac{\mathbf{u}}{1 + \|\mathbf{u}\|} + \mathbf{u} \times a, \tag{14}$$

where a is a (small) scalar constant, and $\mathbf{u} \in \mathbb{C}^n$. The small residue is to avoid vanishing gradient of loss function when $\mathbf{u}$ is large. We used this activation function in our numerical experiments and $a = 0.1$.

Alternatively, one may choose the identity function, that is

$$\boldsymbol{\sigma}(\mathbf{u}) = \mathbf{u}, \tag{15}$$

which in scalar form is a popular choice of activation function for the output layer.

ReLu function and Leaky ReLu function in vector forms can also be equivariant with respect to unitary operators, but their conditionals may require adjustments. Assuming $\mathbf{u} \in \mathbb{C}^n$, the Leaky ReLu function can be defined as

$$\boldsymbol{\sigma}(\mathbf{u}) = \begin{cases} \mathbf{u} & \texttt{if } \|\mathbf{u}\| \geq c, \\ k\mathbf{u} & \textit{otherwise}, \end{cases} \tag{16}$$

where $1 > k \in \mathbb{R}^{\geq 0}$ and $c \in \mathbb{R}^{\geq 0}$ are positive scalar constants. If $k = 0$, it is a ReLu function. Since identity function is equivariant, ReLu and Leaky ReLu functions in vector forms are equivariant functions. We should note the norm $\|\mathbf{u}\|$ is always non-negative. If $c = 0$, it essentially is an identity function. If one wants to have "*otherwise*" output with variable $\mathbf{u}$ other than $\mathbf{u} = \mathbf{0}$, a positive value c can be chosen.

3.3. Including local scalar features

We can introduce extra scalar features into our ENN, in addition to vector elements. The idea is that we will increase the number of coordinates from n to $n + m$, and we only consider unitary operations that do not change the extra m coordinates.

Formally, for each input $\mathbf{x}_0^{(\alpha)}$, we assume that it has m corresponding scalar features which can be written as a vector $\mathbf{h}_0^{(\alpha)} = \{h_{0,1}^{(\alpha)}, h_{0,2}^{(\alpha)}, \ldots, h_{0,m}^{(\alpha)}\}$, we can rewrite the input vector element into

$$\mathbf{x'}_0^{(\alpha)} = \{\mathbf{x}_0^{(\alpha)}, \mathbf{h}_0^{(\alpha)}\} \in \mathbb{C}^{(n+m)}, \tag{17}$$

and the unit vector

$$\mathbf{e'}_k^{(\alpha)} = \{\mathbf{e}_k^{(\alpha)}, \mathbf{1}\}, \tag{18}$$

where $\mathbf{1}$ is a vector with m elements and all equal 1. (Observe that in the actual implementation, we can reduce $\mathbf{1}$ and the associated weights in the model to a single scalar bias term.)

The operator can be rewritten in matrix form such that

$$\mathcal{U}' = \begin{pmatrix} \mathcal{U} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{19}$$

where $\mathbf{I}$ is an identity $m \times m$ matrix. Plugging them back to equations in previous subsection, they all hold, provided that the definition of norm can fulfill, i.e.

$$\|\mathbf{x'}_k^{(\alpha)}\| = \|\mathcal{U}'\mathbf{x'}_k^{(\alpha)}\|. \tag{20}$$

It essentially means we only apply the group action on part of the input vectors, and keep the features part of the vectors fixed. Features can be anything that are quantifiable, such as color, brightness, contrast, electronic charge, mass, humidity, level of pollutant, and etc.

Although at the output layer, we will get outputs $\mathbf{x'}_L \in \mathbb{C}^{(n+m)\times M_L}$, the loss function can be defined only using part of it. We also need to be careful that $\mathbf{x'}_L$ does not need to have the same unit or meaning as $\mathbf{x'}_0$. For example, if we considers a system of molecules, we may use positions as the vector elements, and charges and masses as features. It means we have vector inputs in $\mathbb{R}^{(3+2)\times M_0}$. Even we have vector outputs $\mathbb{R}^{(3+2)\times M_L}$, the prediction can be forces, atomic energy, and a dummy value that does not enter the loss function. On the other hand, one can also add dummy input features to make each element in $\mathbf{x'}_0$ and $\mathbf{x'}_L$ longer.

3.4. Backpropagation

We can derive an algorithm similar to the commonly known backpropagation. The essence of backpropagation is to reuse the information of the gradient of the loss function with respect to the elements in weight and bias parameters. First, we define our loss function:

$$L = C\left(\mathbf{T}, \boldsymbol{\sigma}_L(\mathbf{y}_{L-1})\right), \tag{21}$$

where $\mathbf{T} \in \mathbb{C}^{n\times M_L}$ is the target data, and C is a non-negative real value function being differentiable with respect to $\boldsymbol{\sigma}_L$. For convenient, we write a combined representation of the weight and bias parameters, such that $\mathbf{z}_k = \{\mathbf{W}_k, \mathbf{b}_k\}$. For each element in $\mathbf{z}_{L-1}$, the derivative

$$\frac{\partial L}{\partial z_{L-1,pq}} = \boldsymbol{\delta}_{L-1}\frac{\partial \mathbf{y}_{L-1}}{\partial z_{L-1,pq}}, \tag{22}$$

where

$$\boldsymbol{\delta}_{L-1} = \frac{\partial C}{\partial \boldsymbol{\sigma}_L}\frac{\partial \boldsymbol{\sigma}_L}{\partial \mathbf{y}_{L-1}}. \tag{23}$$

For other layers, in general, we can write

$$\frac{\partial L}{\partial z_{L-k,pq}} = \boldsymbol{\delta}_{L-k} \frac{\partial \mathbf{y}_{L-k}}{\partial z_{L-k,pq}}, \tag{24}$$

where

$$\boldsymbol{\delta}_{L-k-1} = \boldsymbol{\delta}_{L-k} \frac{\partial \mathbf{y}_{L-k}}{\partial \mathbf{x}_{L-k}} \frac{\partial \boldsymbol{\sigma}_{L-k}}{\partial \mathbf{y}_{L-k-1}}. \tag{25}$$

This allows us to reuse the information of $\boldsymbol{\delta}_{L-k}$ in $\boldsymbol{\delta}_{L-k-1}$. However, it is different from the conventional backpropogation, where the whole derivative of the deeper layer is reused. We only use a part of the derivative in our backpropagation procedure.

3.5. Permutation symmetry

The above construction of ENN has no permutation symmetry yet. We say that a function ϕ having n inputs achieves permutation symmetry, if for any permutation π on $\{1, 2, \ldots, n\}$ and any $(x_1, \ldots, x_n)$,

$$\phi(x_1, \ldots, x_n) = \phi(x_{\pi(1)}, \ldots, x_{\pi(n)}). \tag{26}$$

The order and the number of inputs of ϕ are fixed. In some applications, we may partition the inputs and consider permutations within each part. For instance, we can partition the inputs into $\mathcal{S}_a = \{1, 2, \ldots, m\}$ and $\mathcal{S}_b = \{m+1, \ldots, n\}$ such that for any permutations π_a and π_b on the corresponding parts, permutation symmetry means:

$$\phi(x_1, \ldots, x_n) = \phi(x_{\pi_a(1)}, \ldots, x_{\pi_a(m)}, x_{\pi_b(m+1)}, \ldots, x_{\pi_b(n)}). \tag{27}$$

Permutation symmetry should hold in some physical systems. For example, if we have a molecular composed of H, C and O atoms, then each input corresponds to an atom and the inputs can be partitioned according to the type of atom. Then, atoms in each part can be permuted without affecting the output.

One way to achieve permutation symmetry is to introduce a pre-processing step in the first layer. Suppose there are N_0 input vectors denoted by $(\mathbf{u}^{(\xi)} \in \mathbb{C}^n : \xi \in \{1, 2, \ldots, N_0\})$. This step produces M_0 vectors via a collection of functions $\mathbf{D}^{(\alpha)} : \mathbb{C}^{n \times N_0} \to \mathbb{C}^n$ for each $\alpha \in \{1, 2, \ldots, M_0\}$. Since our final neural network achieves equivariance, we require that for any unitary operator $\mathcal{U}$, the following holds:

$$\mathcal{U}\mathbf{D}^{(\alpha)}(\{\mathbf{u}^{(\xi)}\}) = \mathbf{D}^{(\alpha)}(\{\mathcal{U}\mathbf{u}^{(\xi)}\}). \tag{28}$$

In addition to being equivariant for unitary operators, we describe how each $\mathbf{D}^{(\alpha)}$ also achieves permutation symmetry.

Achieving Permutation Symmetry by Summation. Even though the functions implemented in a layer can look complicated, the principle behind them to achieve permutation symmetry is very simple. An example for ϕ in Eq. (27) can be:

$$\phi(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i w, \tag{29}$$

where w is a trainable weight. The key observation is that w does not depend on the index i, which is subject to permutation. Hence, when the indices i are permuted, the value of the function does not change.

We can also express the idea of partitioning the n inputs and consider permutation symmetry within each part. For example, each part is indexed by δ (also known as a feature), and an index i having feature δ can be represented by $v_{i\delta} = 1$ and 0, otherwise. Then, we can consider the following function:

$$\phi(x_1, \ldots, x_n) = \sum_{\delta} \sum_{i=1}^{n} x_i v_{i\delta} w_\delta, \tag{30}$$

where the trainable weights w_δ again does not depend on the index i. Observe that if indices i having the same feature δ are permuted, the value of the function does not change.

Graph Neural Network Example. Applying the above principles for permutation symmetry, we may consider adding a layer of GNN to our ENN. First, we can write a set of scalar functions for node i,

$$D_i^{(\alpha)} = v'_{i\alpha} = \sigma\left(\sum_{j,\beta,\delta} e_{ij}^{\beta} v_{j\delta} w_{\beta\delta\alpha}\right) \tag{31}$$

where β are features of edge $\mathbf{E}$, δ are features of node $\mathbf{V}$, α are features of node $\mathbf{V}'$, σ is the activation function, and $w_{\beta\delta\alpha}$ is a trainable rank-3 tensor weight between features β, δ and α. e_{ij}^{β}, $v_{j\delta}$, and $v'_{i\alpha}$ are elements in $\mathbf{E}$, $\mathbf{V}$ and $\mathbf{V}'$, respectively. We can see such scalar functions hold permutation symmetry, and are rotational invariant. However, they are not in vector equivariant form. We may resolve the issue by devising a set of equivariant vector functions

$$\mathbf{D}_i^{(\alpha)} = \mathbf{v}'_{i\alpha} = \boldsymbol{\sigma}\left(\sum_{j,\beta,\delta} \frac{\mathbf{u}_{ij}}{u_{ij}} e_{ij}^{\beta} v_{j\delta} w_{\beta\delta\alpha}\right), \tag{32}$$

provided that e_{ij}^{β} is invariant with respect to the application of group action on $\mathbf{u}_i$ and $\mathbf{u}_j$, where $\mathbf{u}_{ij} = \mathbf{u}_j - \mathbf{u}_i$. The vector activation function $\boldsymbol{\sigma}$ is also required to be a vector equivariant function.

If we now consider a system of atoms, for atom i, it has N_0 neighbors within a cut-off distance r_c. $\{\mathbf{r}_j \in \mathbb{R}^3 | r_{ij} < r_c\}$ is the set of positions of neighboring atoms of atom i. We may consider each atom as a node. As a special case, one can put

$$e_{ij}^{\beta} = \begin{cases} \exp(-\eta^{(\beta)}(r_{ij} - r_s^{(\beta)})^2) f_c(r_{ij}), & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases} \tag{33}$$

where the interatomic distance between atom i and j is $r_{ij} = |\mathbf{r}_{ij}| = |\mathbf{r}_j - \mathbf{r}_i|$. The f_c is a scalar smooth-out function such that at the cut-off distance r_c, $f_c(r_c) = 0$, and is continuous and differentiable up to at least second derivatives. This is similar to the implementation in SchNet (Schütt et al., 2018).

We further assume there is only one feature corresponding to δ, where $v_{j1} = 1$, the weight is a Kronecker delta function $w_{\beta 1\alpha} = \delta_{\beta\alpha}$, and the activation function is an identity function. Eq. (31) becomes:

$$D_i^{(\alpha)}(\{\mathbf{r}_j\}) = \sum_{j, j\neq i} \exp(-\eta^{(\alpha)}(r_{ij} - r_s^{(\alpha)})^2) f_c(r_{ij}), \tag{34}$$

where $\alpha \in \{1, 2, \ldots, M_0\}$. The set of hyper-parameters $\{\eta^{(\alpha)}, r_s^{(\alpha)}\}$ are predetermined values. Interestingly, this is in the same functional form as suggested by Behler and Parrinello (2007) who mapped the local atomic environment to a set of atom-centered symmetry functions (or called spatial descriptors) and used them to develop machine-learned (ML) interatomic potential.

Following similar logic, one may devise a set of vector function by augmenting above scalar spatial descriptor, such that

$$\mathbf{D}_i^{(\alpha)}(\{\mathbf{r}_j\}) = \sum_{j, j\neq i} \frac{\mathbf{r}_{ij}}{r_{ij}} \exp(-\eta^{(\alpha)}(r_{ij} - r_s^{(\alpha)})^2) f_c(r_{ij}). \tag{35}$$

It is straightforward to check

$$\mathcal{U}\mathbf{D}_i^{(\alpha)}(\{\mathbf{r}_j\}) = \mathbf{D}_i^{(\alpha)}(\{\mathcal{U}\mathbf{r}_j\}), \tag{36}$$

which resembles Eq. (28). The subscribe j here is corresponding to the superscribe (ξ) of $\mathbf{u}$ in Eq. (28). We will present an example using the vector spatial descriptors below.

We show theoretically that one can introduce permutation symmetry by adding an extra layer of properly designed vector form GNN to our ENN.

3.6. Limitations of our ENN

Our implementation has the limitation that the group action on the input data is the same group action on the output data, and the group action is restricted to unitary transformation. Therefore, if one applies the unitary operator on the input data, but the target data does not experience the same transformation, our method does not apply.

For example, in physical systems, quantities can have odd or even parity symmetry, e.g., consider the parity transformation $\mathcal{P} : (x, y, z) \mapsto (-x, -y, -z)$. For quantities with odd parity symmetry, they will have sign change according to the parity transformation. Our ENN can be applied to predict these quantities. However, for vector quantities with even parity symmetry, our ENN does not apply. For example, in classical mechanics, the angular momentum

$$\mathbf{L} = \mathbf{r} \times \mathbf{p}. \tag{37}$$

If we apply the parity transformation, we get $\mathbf{r} \to -\mathbf{r}$ and $\mathbf{p} \to -\mathbf{p}$, but we still get

$$\mathbf{L} = -\mathbf{r} \times -\mathbf{p}. \tag{38}$$

If we use $\mathbf{r}$ and $\mathbf{p}$ as the input data, we cannot use our ENN to predict $\mathbf{L}$. A possible solution is to manually apply sign change to the output data according to the input data.

For scalar quantities with even parity symmetry, it can be remedied by converting $\mathbf{x}_k$ at arbitrary k layers, where $\mathbf{x}_k \in \mathbb{C}^{n \times M_k}$, to invariant scalar quantities. For example we can set $\mathbf{w}_p$ as a function of $\mathbf{x}_k$, where $\mathbf{w}_p \in \mathbb{C}^{M_p}$, and plug $\mathbf{w}_p$ into other implementations of neural networks with M_p scalar inputs. An obvious example is the scalar spatial descriptors for predicting the interatomic potential energy (Behler & Parrinello, 2007) as mentioned in previous subsection, where energy has even parity symmetry.

4. Empirical experiments

When the governing rules of a physical system are unknown, it is hard to apply any analytical method to study the evolution of a system. A viable method nowadays is to adopt an ML model supplied with a substantial amount of data. After proper training, the model will attain certain predictive power.

In atomic scale simulations, there are many developments on the interatomic potentials using different ML methods, such as Gaussian process (Bartók et al., 2010), neural network (Batzner et al., 2022; Behler & Parrinello, 2007; Kondor, 2018) and moment tensor (Shapeev, 2016). Atomic positions and atomic energies are used as the input and output data, respectively. The atomic energies are usually obtained from density function theory (DFT) calculations (Hohenberg & Kohn, 1964; Kohn & Sham, 1965). Atomic forces are then calculated as the derivative of the total energy (or Hamiltonian). This approach is viable only if energy can be calculated. Unfortunately, in many observations, energy is not an included quantity.

We ask two questions here. First, can we predict forces directly from positions, without knowing the energies explicitly? This question is not limited to atomic scale modeling. We can ask similar questions in meteorology and cosmology. Second, can we predict multiple forces in a single calculation? In conventional ML interatomic potential, only one force vector is calculated from an ML model. We are going to use our ENN to demonstrate the possibility of answering "yes" to both questions.

In addition, we demonstrate our ENN is capable of predicting the atomic forces of a system with a large number of particles though the vector spatial descriptors.

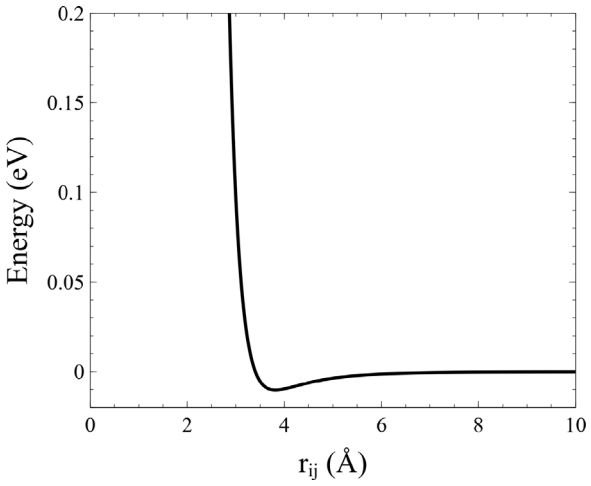

Fig. 1. A plot of the Lennard-Jones potential for Argon according to Eq. (39).

4.1. Model and data

We simulate a system of 4-body motion governed by a Hamiltonian model. Our aim is to predict the forces when atoms are locating at different positions, and simulate the dynamics. We generate data using a well defined physical model, which allows us to examine the errors.

We adopt a pair-wise Lennard-Jones potential for Argon (Rahman, 1964):

$$U_{ij} = 4\epsilon \left(\left(\frac{r_0}{r_{ij}} \right)^{12} - \left(\frac{r_0}{r_{ij}} \right)^{6} \right), \tag{39}$$

where $\epsilon/k_B = 120$ Kelvin, $r_0 = 3.4$ Å, and k_B is the Boltzmann constant. A plot of the potential energy is shown in Fig. 1.

The interatomic potential energy of the system is written as a sum of pair-wise interaction energies:

$$U = \sum_{i,j,i>j} U_{ij}(r_{ij}), \tag{40}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$. The force acting on atom i is

$$\mathbf{F}_i = -\frac{\partial U}{\partial \mathbf{r}_i}. \tag{41}$$

We generate 100,000 set of positions in three dimensional space. Each set contains the positions of 4 atoms. The x, y, and z coordinates of each atom is generated randomly according to the Gaussian distribution with mean equals zero and standard deviation equals 3 Å. Then, we calculate the interatomic distance of each pair of atoms. If any of them smaller than $r_{min} = 2.8$ Å, we discard this set of positions and generate a new one. We repeat this procedure until no interatomic distance is small than r_{min}. This is to avoid the occurrence of very large atomic force due to small separation. We can readily understand it by inspecting Fig. 1. The energy have a drastic increase at around 3 Å. Atoms can hardly be in such small separation in dynamic simulations. Using these positions, we can obtain a set of four atomic forces for each set of positions using the Lennard-Jones potential.

Instead of using the positions as inputs directly, we use the relative positions as inputs:

$$\mathbf{x}_0 = \{\mathbf{r}_{12}, \mathbf{r}_{13}, \mathbf{r}_{14}, \mathbf{r}_{23}, \mathbf{r}_{24}, \mathbf{r}_{34}\}. \tag{42}$$

This takes care of the translational symmetry.

The target data are simply the atomic forces:

$$\mathbf{T} = \{\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3, \mathbf{F}_4\}. \tag{43}$$

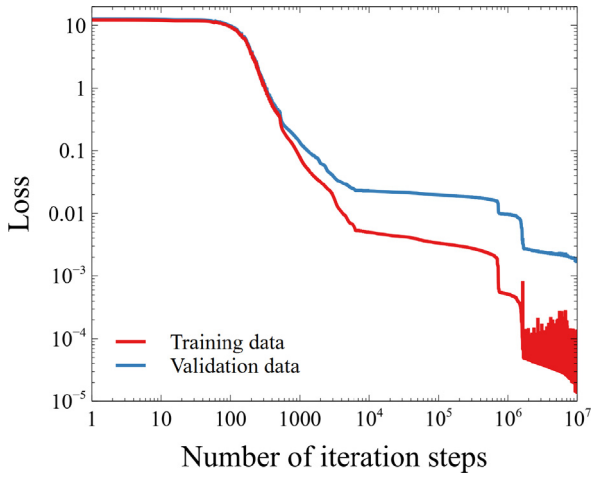

Fig. 2. The value of the loss function (unitless) calculated using the training and validation data as a function of iterative steps using FIRE algorithm.

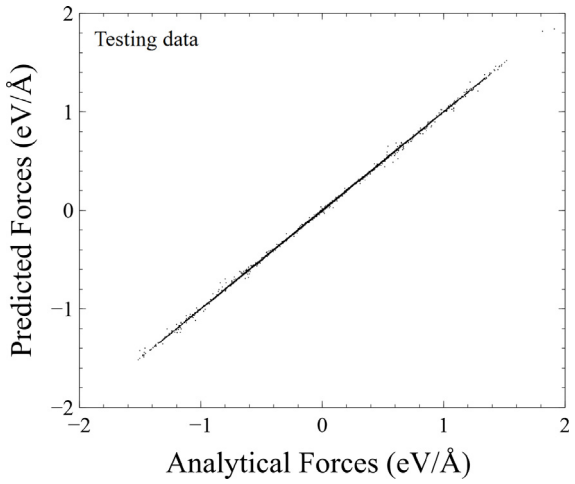

Fig. 3. Each component of the atomic forces calculated analytically using the Lennard-Jones potential versus the predictions calculated using the trained ENN. They are calculated using the testing data.

We rescale both the input and output data by their standard deviations before we use them to train an ENN. Data are split, where 60% is for training, 20% is for validation, and 20% for testing.

4.2. Learning and errors

The relationship between the input and output data is learned by an ENN, which has five hidden layers. The number of nodes in each layers counting from input to output layers are 6, 50, 90, 100, 80, 50, and 4. In total, there are 52,000 variables. The determination of the number of nodes in each layer is based on intuition. We did not optimize over the choice of the number of nodes in each layer, other than making sure that we have enough memory; this shows that our method is robust and we do not need to spend too much time to tune the model. It is also to demonstrate the difference between feedforward NN and GNN that one can easily adjust the feedforward NN by changing, not only the depth, but also the number of nodes according to need. We should note the total number of parameters is significantly less than the training data to avoid overfitting.

The loss function is the mean-squared error, which is defined as

$$L = \frac{1}{N_{data}} \sum_{data} (\mathbf{T} - \mathbf{x}_L)^2, \tag{44}$$

where N_{data} is the number of used data and $\mathbf{x}_L$ is the output data. The weight parameters $\mathbf{W}$ are initialized according to normalized Xavier method (Glorot & Bengio, 2010). The bias parameters $\mathbf{b}$ are initialized to zeros.

The training of ENN is performed through minimizing the loss function with respect to $\{\mathbf{W}, \mathbf{b}\}$. We used the FIRE algorithm (Bitzek et al., 2006). It is a minimization method commonly used for relaxing atomic structures. It is similar to the Nesterov momentum method (Nesterov, 1983), and has fast convergence behavior in practice. We briefly discuss the method and our adaptation in Appendix.

Fig. 2 shows the change of the value of the loss function calculated using the training data and the validation data. We performed 10 million iterations. We see that both of them drop significantly. As expected, the training loss drops faster than validation loss. However, it seems that the model does not suffer from overfitting. We stop the iteration process as soon as we observe fluctuations in the training loss, i.e., further iterations might actually increase the training loss.

Using testing data, we can calculate the atomic forces analytically according to the Lennard-Jones potential and predict them by our trained ENN. In Fig. 3, the analytical and predicted values are plotted against each others. We plotted all the x, y, and z components of the data. The root mean square deviation (RMSD) is 0.00118 eV/Å. Observe that the training data are in the order of 0.1 to 1 eV/Å, and the average error is in the order of 0.001 eV/Å, which is fairly satisfactory.

4.3. Non-equivariant feedforward neural network

We compared our ENN with the conventional scalar feedforward NN, which is not equivariant. The same set of data is used. We flattened the input and output layers into column vectors, so they have $6 \times 3 = 18$ and $4 \times 3 = 12$ components, respectively. Even though we use the same number of nodes in each layers, since the bias $\mathbf{b}_k$ is a vector, not a matrix, the total number of variables in the conventional model is not the same as in our ENN.

For better comparison, we trained two conventional scalar feedforward NNs. We labeled them as model (A), with 18, 50, 90, 100, 80, 50, and 12 nodes from input to output layers, in total 27,382 variables, and model (B), with 18, 75, 120, 150, 110, 70, and 12 nodes, in total 53,927 variables. The number of hidden nodes in each layer of model (A) is the same as we used in previous ENN, where the number of variables in model (B) is comparable.

In Fig. 4, it shows the loss function as a function of iteration steps up to 100,000 steps. An additional difference from the previous training on ENN is that the maximum step size in FIRE algorithm is half in (A) and one third in (B), because we found that the models do not converge if the original maximum step size was used.

The training losses of both model (A) and (B) drop comparing to initial values and keep almost constant after 60,000 steps. The training loss of model (B) is lower. We can understand that model (B) has more variables, so it is more flexible to learn the training data. However, we can clearly observe that the two validation losses increase, after a small drop up to about 10,000 iterative steps. A clear generalization gap can be observed. It is a typical indication that the model cannot generalize the data. We should also note that the training losses do not drop further. Comparing the values of training losses with the values in Fig. 2, we can conclude the two conventional feedforward NNs cannot be trained successfully.

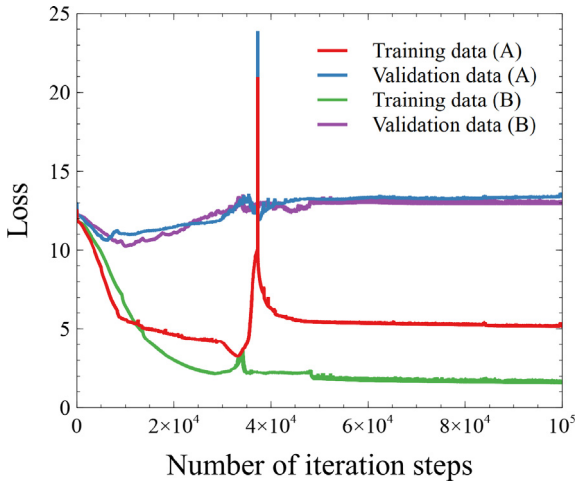

Fig. 4. The value of the loss function calculated using the same training and validation data in Fig. 2, as a function of iterative steps, where a conventional feedforward neural network is used. Label (A) and (B) are models described in text.

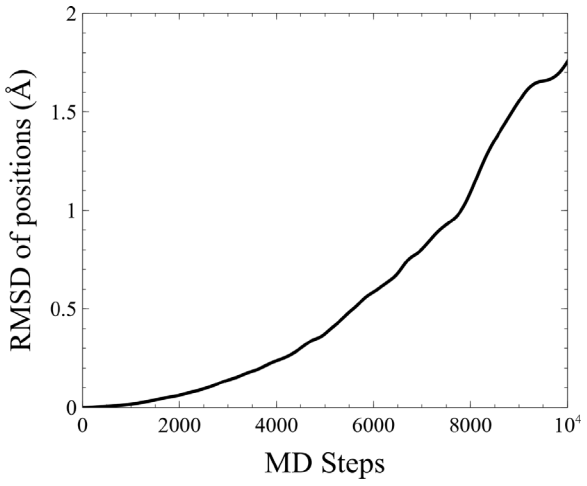

Fig. 5. The root-mean-square-derivation (RMSD) of positions with respect to analytical solution and prediction by our trained ENN. The RMSD is calculated across 10 samples. Each sample contains 4 atoms.

We demonstrate that having the symmetry being properly built in the NN is important for the current data set. The total number of variables and number of nodes in each layer are not the key of getting a good learning result.

4.4. Dynamic simulations

We use the forces predicted by our trained ENN to drive the evolution of a system of four Argon atoms using molecular dynamics (MD). We will also compare our predictions with analytical solutions. Note that our ENN was not trained to any trajectory of atomic motion. All training data are static. No history dependent information was involved in the training.

The motion of atoms are governed by the Newton's equations

$$\frac{d\mathbf{p}_i}{dt} = \mathbf{F}_i, \tag{45}$$

$$\frac{d\mathbf{r}_i}{dt} = \frac{\mathbf{p}_i}{m_i}, \tag{46}$$

where the position and momentum of atom i are $\mathbf{r}_i \in \mathbb{R}^3$, $\mathbf{p}_i \in \mathbb{R}^3$ and the atomic mass is m_i.

Using our trained ENN, we can predict the atomic forces $\{\mathbf{F}_i\}$. On the other hand, if the analytical form of a Hamiltonian is known, the atomic force is

$$\mathbf{F}_i = -\frac{\partial \mathcal{H}}{\partial \mathbf{r}_i}, \tag{47}$$

where the Hamiltonian is:

$$\mathcal{H} = \sum_i \frac{\mathbf{p}_i{}^2}{2m_i} + U(\{\mathbf{r}_i\}). \tag{48}$$

Without introducing perturbation and dissipation, this dynamic system is a closed system, and so the total energy should conserve.

We initialized ten samples. The positions of Argon atoms are initialized at $(3, 0, 0.1)$, $(-3, -0.1, 0)$, $(0.1, 2.5, 0)$, and $(0, -2.5, -0.1)$, where unit is in Å. Velocities are generated randomly with kinetic energy corresponding to a temperature of 10 Kelvin. The mass of an Argon atom is $39.948u$. We integrated Newton's equation using velocity Verlet algorithm. We used a time step of 1fs, which is a conventional value for MD simulations.

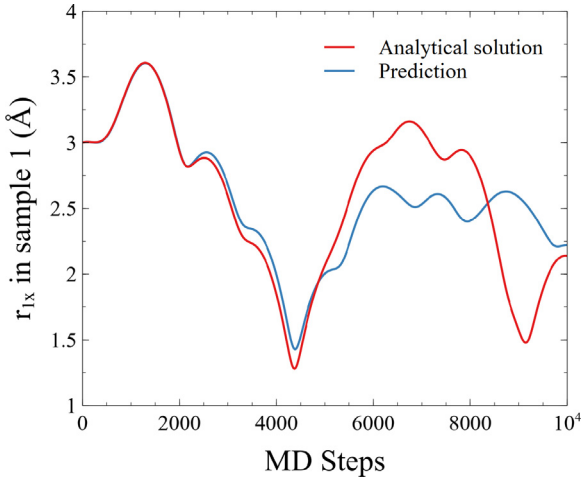

Fig. 6. The x component of the position of atom 1 in sample 1. Analytical solution and prediction are shown.

We calculated the RMSD of the positions of atoms. It is defined as:

$$\mathrm{RMSD}(\{\mathbf{r}_i\}) = \sqrt{\frac{1}{N_s N_{at}} \sum_{samples,atoms} (\mathbf{r}_i^a - \mathbf{r}_i^p)^2}, \tag{49}$$

where $N_s = 10$ is the number of sample, $N_{at} = 4$ is the number atoms in a sample, $\mathbf{r}_i^a$ is the position of atom i calculated according to analytical solution, and $\mathbf{r}_i^p$ is the position calculated using forces predicted by ENN.

Fig. 5 shows the RMSD of positions as a function of MD steps. As expected, they deviate more and more as a function of steps, because the error is accumulating throughout the simulation. We may inspect the real trajectory of an atom in Fig. 6. It shows the x component of atom 1 in sample 1. We see the initial 1500 MD steps predictions are fairly good, and up to 4000 MD steps are acceptable. Our ENN shows certain predictive power, and the predictions are three dimensional vectors.

We calculated the RMSD of the system energies. It is defined as:

$$\mathrm{RMSD}(E) = \sqrt{\frac{1}{N_s} \sum_{samples} (E^a - E^p)^2}, \tag{50}$$

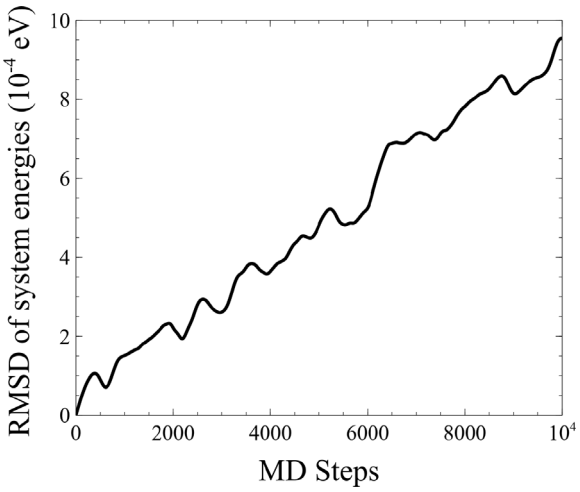

Fig. 7. The RMSD of energy calculated using the positions calculated analytically or using ENN.

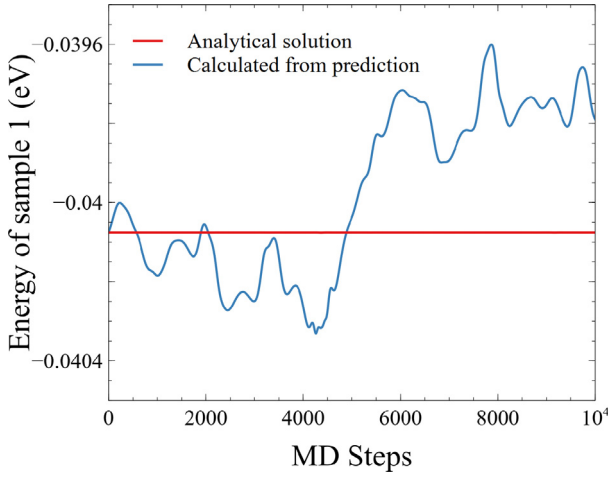

Fig. 8. The system energy of sample 1. Analytical solution and prediction are shown.

where E^a and E^p are the total energy calculated using Eq. (48). The potential energy are calculated using the positions of atoms that evolve according to forces calculated analytically or by prediction using ENN.

Fig. 7 shows the RMSD of energy across ten samples. Again, we can see the deviation accumulates. However, we should remember that in the training of ENN, we did not provide any information about energy to the training. If we look at around 4000 steps, the RMSD is less than about 4×10^{-4} eV, which is about 2 order of magnitude smaller than the system energy. Our results are encouraging. It shows that even if we do not know the system total energy (or Hamiltonian), we can still predict forces, which are vectors, using our ENN. We can also predict multiple forces at the same time. Fig. 8 even shows the prediction of the energy of a sample is fairly good.

4.5. Many particles case

In many physical systems, we need to consider a large number of objects that can be ranging from a few hundreds to the order of the Avogadro number (6.02×10^{23}), or even larger. However, in reality, the features of an object can be a result of contributions from all other objects, and it is not always tractable to consider such a large number of interactions. Instead, one can assume that the neighborhood of nearby objects contribute the most and the remaining further objects are negligible. This allows spatial descriptors (Behler & Parrinello, 2007) to describe the local atomic environment and predict atomic energy. Since prediction is based on a fixed neighborhood, the calculation time becomes proportional to the number of objects, which essentially is $O(N)$ calculation, instead of $O(N^2)$.

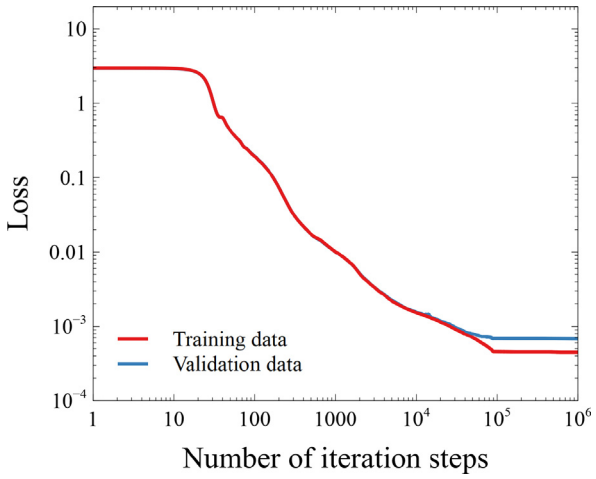

Fig. 9. The value of the loss function calculated using the training and validation data of the many particles case as a function of iterative steps.

In Section 3.5, we discuss a viable way to write down vector spatial descriptors. We demonstrate that it is feasible to predict a vector feature in practice, and use them to predict the atomic forces of an ensemble of atoms according to their neighborhood.

We write our vector spatial descriptors of atom i as:

$$\mathbf{D}_i^{(\alpha)}(\{\mathbf{r}_j\}) = \sum_{j,j\neq i} \frac{\mathbf{r}_{ij}}{r_{ij}} \exp(-\eta(r_{ij} - r_s^{(\alpha)})^2) f_c(r_{ij}), \tag{51}$$

which are vector functions depending on the relative position vectors $\mathbf{r}_{ij}$. In total, we use 73 vector spatial descriptors to describe the atomic environment of a particular atom. We set $\eta = 0.1^{-2}\text{Å}^{-2}$, $r_s^{(\alpha)} = 2.8 + \alpha \times 0.1$ Å, where $\alpha = 0, 1, 2, \ldots, 72$. The cut-off function is chosen as:

$$f_c(r_{ij}) = \frac{1}{2}\left(\cos\left(\frac{\pi r_{ij}}{r_c}\right) + 1\right)\Theta(r_c - r_{ij}) \tag{52}$$

where Θ is a Heaviside step function and the cut-off distance is $r_c = 10$ Å.

We create data for training, validation and testing by randomly putting Argon atoms in cubic boxes with side lengths of 45, 50, 55, 60, 65, 70, 75, 80, 85, and 90 Å. We insert up to 4000 atoms in each box. However, as we discussed before, atoms in general do not come very close in dynamic simulations, we set a minimum distance of 2.8 Å between each pair of atoms. For each insertion, if one cannot put an atom, according to the minimum distance criteria, after 50 trials, we give up. Therefore, some boxes with smaller side lengths may have less atoms. Then, we calculate the atomic forces analytically using Eq. (39). We create 10 sets of data with different random seeds. We use 6 sets as training data, 2 sets as validation data, and 2 sets as testing data.

Then, we create an ENN with 73, 200, 100, 50, 10, and 1 nodes from the input to output layers. The inputs are descriptors divided by their standard deviations across all data, and the target is the atomic force divided by its standard deviation across all data.

We train our ENN using the training data and monitor the change of training loss and validation loss as defined in Eq. (44).

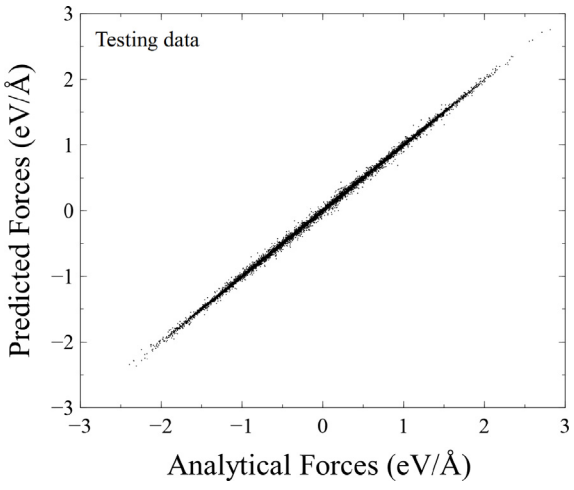

Fig. 10. Each component of the atomic forces calculated analytically using the Lennard-Jones potential versus the predictions calculated using the trained ENN with input using spatial descriptors. They are calculated using the testing data of the many particles case.

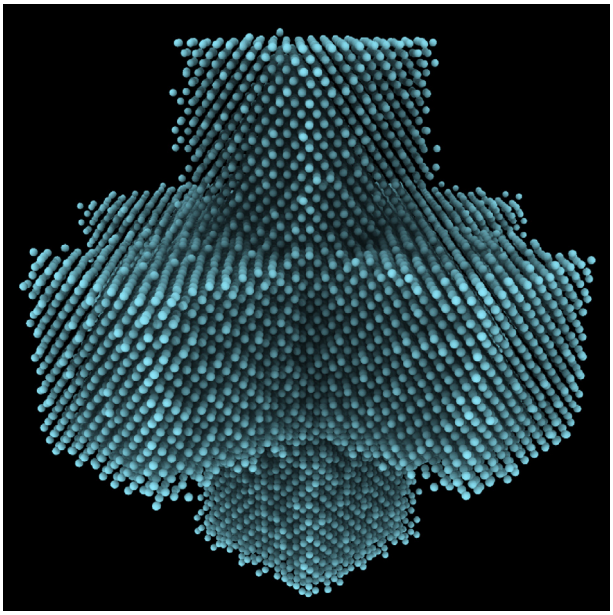

Fig. 11. We thermalize a system of Argon atoms in face-centered cubic structure to 60 K. Then, we cut out a 3-dimensional cross shape, which contains 26,679 atoms. It is used as the initial configuration for further dynamic simulations.

We can observe in Fig. 9 that both the training loss and validation loss drop steadily until about 10^5 steps. No significant changes can be observed afterwards, and so we stop the iteration at 10^6 steps.

We examine the trained ENN using our testing data set, which has 74,506 atoms. Since each force vector has three components, we have in total 223,518 points plotted in Fig. 10. Each component of a predicted force is plotted against its analytical value. We observe fairly good prediction from the figure. The RMSD is 0.00588 eV/Å. We found that the results are encouraging, according to the spread of data from about -2 to 2 eV/Å.

Finally, we use our ENN to drive a dynamic simulations. First, we created a simulation box with Argon atoms using $50 \times 50 \times 50$ face-centered cubic unit cells. Each unit cell contains 4 atoms and has a lattice constant of 5.24411 Å. In total, there are 500,000 atoms. Then, we thermalize the system to 60 K for 10 ps. At the same time, the box can change volume, so that the internal pressure can attain zero. Once it is done, we fix the simulation box, and cut the final configuration into a 3-dimensional cross shape, as shown in Fig. 11. Such configuration contains 26,679 atoms. All these manipulations are performed using LAMMPS (Thompson et al., 2022). The visualization of the atomic configuration is done by OVITO (Stukowski, 2010).

Using this as the starting configuration, we perform MD simulations using the predicted forces and analytical forces according to the Newton's equation. We integrate 20,000 MD time steps, where each time step is 1 fs. Fig. 12 shows the RMSD of the positions of atoms. We observe that the value becomes larger and larger as expected. It is because the trajectory of each individual atom can deviate more whenever there is a slight difference on the analytical and predicted forces.

However, if we inspect the overall picture in Fig. 13, the change of the shapes in both cases are similar. It is more surprising if we look at the change of the system temperature in Fig. 14, which is calculated from the kinetic energy. We observe fairly good prediction. We should note that temperature is a thermodynamic quantity that describes the overall state of a system, instead of individual atoms.

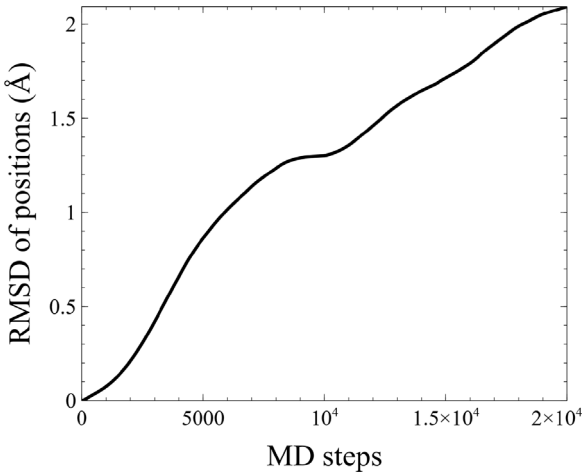

Fig. 12. The root mean square deviation (RMSD) of the positions with respect to analytical solution and prediction by our trained ENN. It is calculated using the positions of all atoms of the many particles case.

As a proof of concept, we show that our ENN is applicable to molecular dynamics simulations and our ENN can predict reasonably well the atomic forces and vector features of a large system.

5. Conclusion

We have designed a new feedfoward ENN for unitary transformations. It does not involve convolution with higher order representation, such as spherical harmonics and Wigner matrices. Moreover, our model works for vectors in arbitrary dimensions. Our ENNs can be trained by efficient backpropagation and an extra layer of GNN can be added to achieve permutation symmetry. Examples on the dynamics of Argon atoms are given showing the practicality of our architecture via empirical simulations.

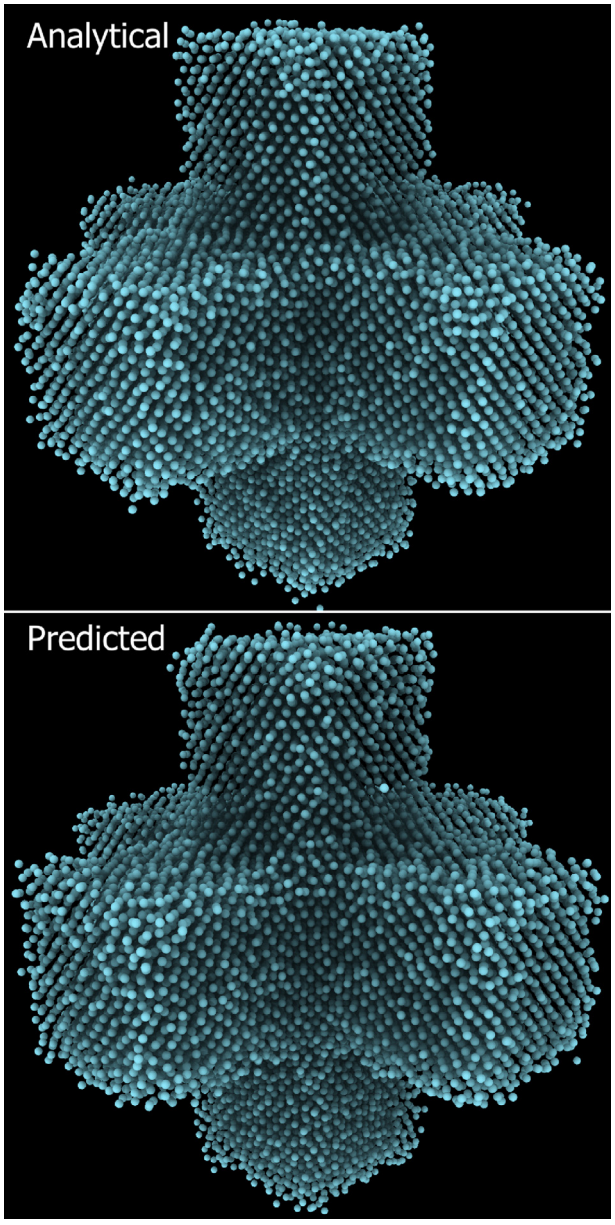

Fig. 13. After running for 20,000 time steps, where each time step is 1 fs, we observe that the change of the shapes of the two simulations with atomic forces calculated analytically or predicted using a trained ENN. They look similar.

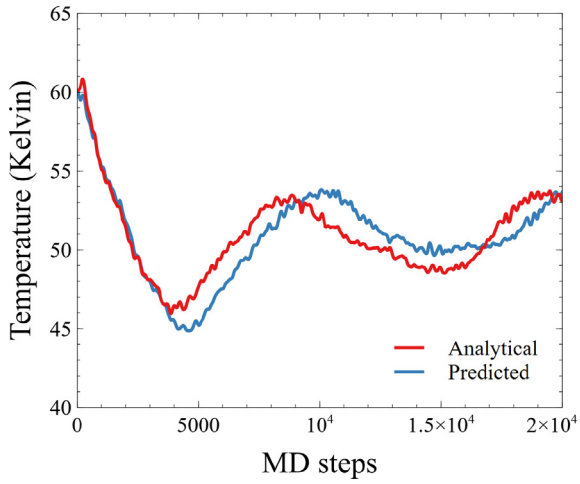

Fig. 14. The temperatures of the two systems with forces calculated analytically or predicted using ENN. The temperatures are calculated from the kinetic energy of atoms.

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.

Data availability

Data will be made available on request.

Acknowledgments

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion) and from the EPSRC, United Kingdom [grant number EP/W006839/1]. To obtain further information on the data and models underlying this paper please contact PublicationsManager@ukaea.uk. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them. T.-H. Hubert Chan was partially supported by the Hong Kong RGC under the grants 17201220, 17202121, and 17203122.

Appendix. FIRE minimization algorithm

FIRE (fast inertial relaxation engine) (Bitzek et al., 2006) is a minimization algorithm commonly used in atomic scale simulations for structural relaxation. We are going to briefly mention the algorithm below and discuss our adaptation.

Assuming we have a system of atoms governed by the Hamiltonian $\mathcal{H}$, we may find a configuration with potential energy at local minimum through following steps.

Step 1: Define parameters N_{min}, f_{inc}, f_{dec}, α_{start}, f_α, Δt, Δt_{max}, and i_{max}. Set $\alpha = \alpha_{start}$, $N = 0$, and $i = 0$.

Step 2: Set the initial positions $\mathbf{x}$ and atomic mass m. Initialize velocities $\mathbf{v} = 0$.

Step 3: Calculate the atomic forces $\mathbf{F} = -\nabla\mathcal{H}(\mathbf{x})$.

Step 4: Put

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{v}\Delta t,$$
$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{\mathbf{F}}{m}\Delta t.$$

Step 5: Calculate $P = \mathbf{F}\cdot\mathbf{v}$.

Step 6: Put $N \to N+1$ and set

$$\mathbf{v} \to (1-\alpha)\mathbf{v} + \alpha|\mathbf{v}|\frac{\mathbf{F}}{|\mathbf{F}|}. \tag{A.1}$$

Step 7: if $P > 0$ and $N > N_{min}$, set

$$\Delta t \to \min(\Delta t f_{inc}, \Delta t_{max})$$
$$\alpha \to \alpha f_\alpha$$

Step 8: if $P \le 0$, set

$$\Delta t \to \Delta t f_{dec}$$
$$\mathbf{v} \to \mathbf{0}$$
$$\alpha \to \alpha_{start}$$
$$N \to 0$$

Step 9: Set $i \to i+1$. Go to Step 3, or end if $i > i_{max}$.

In our case, we are minimizing the loss function with respect to the weight and bias parameters $\{\mathbf{W}, \mathbf{b}\}$. We flattened $\{\mathbf{W}, \mathbf{b}\}$ to a column vector and treated it as $\mathbf{x}$. We also flattened the gradient of the loss function and treated it as the negative of $\mathbf{F}$. After some trials and errors, we used a pseudo mass $m = 0.1$, $\Delta t = 0.001$, and $\Delta t_{max} = 0.01$. For other parameters, we follow the original suggestions (Bitzek et al., 2006), $N_{min} = 5$, $f_{inc} = 1.1$, $f_{dec} = 0.5$, $\alpha_{start} = 0.1$, and $f_\alpha = 0.99$.

References

Bartók, A. P., Payne, M. C., Kondor, R., & Csányi, G. (2010). Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons. *Physical Review Letters*, *104*, Article 136403. http://dx.doi.org/10.1103/PhysRevLett.104.136403, URL https://link.aps.org/doi/10.1103/PhysRevLett.104.136403.

Batzner, S., Musaelian, A., Sun, L., Geiger, M., Mailoa, J. P., Kornbluth, M., Molinari, N., Smidt, T. E., & Kozinsky, B. (2022). E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, *13*(1), Article 2453. http://dx.doi.org/10.1038/s41467-022-29939-5, URL https://doi.org/10.1038/s41467-022-29939-5.

Behler, J., & Parrinello, M. (2007). Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical Review Letters*, *98*, Article 146401. http://dx.doi.org/10.1103/PhysRevLett.98.146401, URL https://link.aps.org/doi/10.1103/PhysRevLett.98.146401.

Bitzek, E., Koskinen, P., Gähler, F., Moseler, M., & Gumbsch, P. (2006). Structural relaxation made simple. *Physical Review Letters*, *97*, Article 170201. http://dx.doi.org/10.1103/PhysRevLett.97.170201, URL https://link.aps.org/doi/10.1103/PhysRevLett.97.170201.

Brandstetter, J., Welling, M., & Worrall, D. E. (2022). Lie point symmetry data augmentation for neural PDE solvers. In *ICML*. http://dx.doi.org/10.48550/ARXIV.2202.07643.

Cobb, O., Wallis, C. G. R., Mavor-Parker, A. N., Marignier, A., Price, M. A., d'Avezac, M., & McEwen, J. (2021). Efficient generalized spherical CNNs. In *International conference on learning representations*. URL https://openreview.net/forum?id=rWZz3sJfCkm.

Cohen, T. S., Geiger, M., Köhler, J., & Welling, M. (2018). Spherical CNNs. In *International conference on learning representations*. URL https://openreview.net/forum?id=Hkbd5xZRb.

Cohen, T., & Welling, M. (2016). Group equivariant convolutional networks. In M. F. Balcan, & K. Q. Weinberger (Eds.), *Proceedings of machine learning research*: *48*, *Proceedings of the 33rd international conference on machine learning* (pp. 2990–2999). New York, New York, USA: PMLR, URL https://proceedings.mlr.press/v48/cohenc16.html.

Esteves, C., Allen-Blanchette, C., Makadia, A., & Daniilidis, K. (2020). Learning SO(3) equivariant representations with spherical CNNs. *International Journal of Computer Vision*, *128*(3), 588–600. http://dx.doi.org/10.1007/s11263-019-01220-1.

Gerken, J. E., Aronsson, J., Carlsson, O., Linander, H., Ohlsson, F., Petersson, C., & Persson, D. (2021). Geometric deep learning and equivariant neural networks. http://dx.doi.org/10.48550/ARXIV.2105.13926, URL https://arxiv.org/abs/2105.13926.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh, & M. Titterington (Eds.), *Proceedings of machine learning research*: *9*, *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR, URL https://proceedings.mlr.press/v9/glorot10a.html.

Hohenberg, P., & Kohn, W. (1964). Inhomogeneous electron gas. *Physical Review*, *136*, B864–B871. http://dx.doi.org/10.1103/PhysRev.136.B864, URL https://link.aps.org/doi/10.1103/PhysRev.136.B864.

Kohn, W., & Sham, L. J. (1965). Self-consistent equations including exchange and correlation effects. *Physical Review*, *140*, A1133–A1138. http://dx.doi.org/10.1103/PhysRev.140.A1133, URL https://link.aps.org/doi/10.1103/PhysRev.140.A1133.

Kondor, R. (2018). N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. http://dx.doi.org/10.48550/ARXIV.1803.01588, URL https://arxiv.org/abs/1803.01588.

Kondor, R., Lin, Z., & Trivedi, S. (2018). Clebsch– gordan nets: a fully Fourier space spherical convolutional neural network. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems, vol. 31*. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2018/file/a3fc981af450752046be179185ebc8b5-Paper.pdf.

Kondor, R., & Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the action of compact groups. In J. Dy, & A. Krause (Eds.), *Proceedings of machine learning research*: *80*, *Proceedings of the 35th international conference on machine learning* (pp. 2747–2755). PMLR, URL https://proceedings.mlr.press/v80/kondor18a.html.

Müller, P., Golkov, V., Tomassini, V., & Cremers, D. (2021). Rotation-equivariant deep learning for diffusion MRI. http://dx.doi.org/10.48550/ARXIV.2102.06942, URL https://arxiv.org/abs/2102.06942.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, *269*, 543–547, URL http://mi.mathnet.ru/dan46009.

Rahman, A. (1964). Correlations in the motion of atoms in liquid argon. *Physical Review*, *136*, A405–A411. http://dx.doi.org/10.1103/PhysRev.136.A405, URL https://link.aps.org/doi/10.1103/PhysRev.136.A405.

Satorras, V. G., Hoogeboom, E., & Welling, M. (2021). E(n) equivariant graph neural networks. In *Proceedings of machine learning research*: *139*, *ICML* (pp. 9323–9332). PMLR, URL https://proceedings.mlr.press/v139/satorras21a.html.

Schütt, K. T., Sauceda, H. E., Kindermans, P.-J., Tkatchenko, A., & Müller, K.-R. (2018). SchNet – A deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, *148*(24), Article 241722. http://dx.doi.org/10.1063/1.5019779, arXiv:https://doi.org/10.1063/1.5019779.

Shapeev, A. V. (2016). Moment tensor potentials: A class of systematically improvable interatomic potentials. *Multiscale Modeling and Simulation*, *14*(3), 1153–1173. http://dx.doi.org/10.1137/15M1054183, arXiv:https://doi.org/10.1137/15M1054183.

Sonoda, S., & Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, *43*(2), 233–268. http://dx.doi.org/10.1016/j.acha.2015.12.005, URL https://www.sciencedirect.com/science/article/pii/S1063520315001748.

Stukowski, A. (2010). Visualization and analysis of atomistic simulation data with OVITO-the open visualization tool. *Modelling and Simulation in Materials Science and Engineering*, *18*(1), http://dx.doi.org/10.1088/0965-0393/18/1/015012.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., & Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. http://dx.doi.org/10.48550/ARXIV.1802.08219, URL https://arxiv.org/abs/1802.08219.

Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., in 't Veld, P. J., Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., & Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, *271*, Article 108171. http://dx.doi.org/10.1016/j.cpc.2021.108171, URL https://www.sciencedirect.com/science/article/pii/S0010465521002836.

Weiler, M., Geiger, M., Welling, M., Boomsma, W., & Cohen, T. S. (2018). 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems, vol. 31*. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2018/file/488e4104520c6aab692863cc1dba45af-Paper.pdf.

Winkels, M., & Cohen, T. S. (2018). 3D G-CNNs for pulmonary nodule detection. In *Medical imaging with deep learning*. URL https://openreview.net/forum?id=H1sdHFiif.

Worrall, D., & Brostow, G. (2018). CubeNet: Equivariance to 3D rotation and translation. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision – ECCV 2018* (pp. 585–602). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-01228-1_35.