CCFE
CULHAM CENTRE FOR FUSION ENERGY

UK Atomic Energy Authority

UKAEA-CCFE-PR(22)62

Pui-Wai Ma, T.-H. Hubert Chan

A Feedforward Unitary Equivariant Neural Network

A Feedforward Unitary Equivariant Neural Network

Pui-Wai Ma, T.-H. Hubert Chan

This is a preprint of a paper submitted for publication in
Neural Networks

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 Sciences, The University of Hong Kong, Hong Kong*

Abstract

We have devised a new type of feedforward neural network. It is equivariant with respect to unitary operators $U(n)$. 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.

Keywords: equivariant neural network, feedforward neural network, unitary equivariant, rotational equivariant

1. Introduction

Neural Networks (NN) gain popularity in many different areas because of its universal approximator property (Sonoda and Murata, 2017). In recent years, equivariant NN (ENN) in different architectures have been applied

*Corresponding Author
Email addresses: Leo.Ma@ukaea.uk (Pui-Wai Ma), Hubert@cs.hku.hk (T.-H. Hubert Chan)

in various areas, such as 3D object recognition (Thomas et al., 2018; Esteves et al., 2020), molecule classification (Weiler et al., 2018), interatomic potential development (Kondor, 2018; Batzner et al., 2022), and medical images diagnosis (Müller et al., 2021; Winkels and Cohen, 2018; Worrall and Brostow, 2018).

When NNs are employed to model some physical phenomenon, 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 a 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 is 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 is 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 (Thomas et al., 2018; Gerken et al., 2021). This is analogous to Fourier transforms in linear space. However, these kinds of implementation are computationally expensive (Cobb et al., 2021).

In physical systems, even though they can in principle be described by physical rules, analytical methods are not always feasible when the analytic 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, the following aspects are novel.

1. We designed a new type of feedforward neural network. The inputs and outputs are vectors in $\mathbb{C}^n$. They are equivariant with respect to unitary operators $U(n)$.
2. In each layer, in additional 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.
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, so it only works with respect to finite symmetry group.

Cohen et al. (2018) considered convolution NN of 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 point 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 coef-

ficients. Error is inherently introduced due to truncated higher order terms.

Our newly designed feedforward neural network guarantees equivariance without any convolution layer. We should note our ENN has structure in vector form which is different from 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 graph NN, their spatial coordinates are not filtered by activation functions. Their spatial coordinates are updated through averaging with respect to neighbors. The number of neurons in each layer is restricted to be the same, where our approach is general enough to allow different numbers of neurons in different layers. In addition to the spatial coordinates on which the operators act, their neural networks contain feature vectors which do not fall under the equivariant aspect. We will also discuss involving extra features in our approach below.

3. Theory

3.1. Definition of equivariance

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 $\mathbb{C}^n$. We consider the unitary group $U(n)$, where each element corresponds a unitary operator on $\mathbb{C}^n$. 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 $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}) \mapsto (\mathcal{U}\mathbf{x}^{(1)}, \mathcal{U}\mathbf{x}^{(2)}, \ldots, \mathcal{U}\mathbf{x}^{(M)})$.

3.2. Structure of equivariant neural network

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

$$\mathbf{x}_k \in \mathbb{C}^{n \times M_k}, \tag{2}$$

where M_k is the number of vector elements of

$$\mathbf{x}_k = \{\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}, ..., \mathbf{x}_k^{(M_k)}\}. \tag{3}$$

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

$$\mathbf{x}_{k+1} \in \mathbb{C}^{n \times M_{k+1}}. \tag{4}$$

We define a variable

$$\mathbf{y}_k = \mathbf{x}_k \mathbf{W}_k + \mathbf{e}_k \mathbf{b}_k. \tag{5}$$

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.

The weight and bias parameters matrices

$$\mathbf{W}_k \in \mathbb{C}^{M_k \times M_{k+1}} \tag{6}$$

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

and

$$\mathbf{e}_k = \left\{ \frac{\mathbf{x}_k^{(1)}}{||\mathbf{x}_k^{(1)}||}, \frac{\mathbf{x}_k^{(2)}}{||\mathbf{x}_k^{(2)}||}, ..., \frac{\mathbf{x}_k^{(M_k)}}{||\mathbf{x}_k^{(M_k)}||} \right\}, \tag{8}$$

where $||.||$ is the norm of an n-dimensional vector. We note $\mathbf{y}_k$ has the same dimension of $\mathbf{x}_{k+1}$. Observe that for any unitary operator $\mathcal{U}$, it holds that

$$||\mathbf{x}_k^{(\alpha)}|| = ||\mathcal{U}\mathbf{x}_k^{(\alpha)}||. \tag{9}$$

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

$$\mathbf{y}_k(\mathcal{U}\mathbf{x}_k) = \mathcal{U}\mathbf{x}_k \mathbf{W}_k + \mathcal{U}\mathbf{e}_k \mathbf{b}_k = \mathcal{U}\mathbf{y}_k(\mathbf{x}_k), \tag{10}$$

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

Then, we define an activation function

$$\boldsymbol{\sigma}_{k+1}(\mathbf{y}_k) = \mathbf{x}_{k+1}. \tag{11}$$

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 shall find an activation function that satisfies the following:

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

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 equation (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 0^{th} 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 $\mathbf{x}_L \to \mathcal{U}\mathbf{x}_L$.

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

$$\boldsymbol{\sigma}(\mathbf{u}) = \frac{\mathbf{u}}{1 + ||\mathbf{u}||} + \mathbf{u} \times a, \tag{13}$$

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 experiment.

Alternatively, one may choose the identity function, that is

$$\boldsymbol{\sigma}(\mathbf{u}) = \mathbf{u}, \tag{14}$$

which in scalar form is a popular choice of activation function for the output layer. Similarly, ReLu function and Leaky ReLu function in vector forms are also equivariant with respect to unitary operators.

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 $\mathbf{h}_0^{(\alpha)} = \{h_{0,1}^{(\alpha)}, h_{0,2}^{(\alpha)}, ..., h_{0,m}^{(\alpha)}\}$, we can rewrite the input vector element into

$$\mathbf{x'}_0^{(\alpha)} = \{\mathbf{x}_0^{(\alpha)}, \mathbf{h}_0^{(\alpha)}\} \in \mathbb{C}^{(n+m)} \tag{15}$$

and the unit vector

$$\mathbf{e'}_k^{(\alpha)} = \{\mathbf{e}_k^{(\alpha)}, \mathbf{1}\} \tag{16}$$

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

$$\mathcal{U}' = \begin{pmatrix} \mathcal{U} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \tag{17}$$

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.

$$||\mathbf{x'}_k^{(\alpha)}|| = ||\mathcal{U}'\mathbf{x'}_k^{(\alpha)}||. \tag{18}$$

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 mean 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 $\mathbf{x'}_0$ and $\mathbf{x'}_L$ become longer vectors.

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:

$$L = C\left(\mathbf{T}, \boldsymbol{\sigma}_L(\mathbf{y}_{L-1})\right) \tag{19}$$

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

$$\frac{\partial L}{\partial z_{L-1,pq}} = \boldsymbol{\delta}_{L-1}\frac{\partial \mathbf{y}_{L-1}}{\partial z_{L-1,pq}}, \tag{20}$$

where

$$\boldsymbol{\delta}_{L-1} = \frac{\partial C}{\partial \boldsymbol{\sigma}_L}\frac{\partial \boldsymbol{\sigma}_L}{\partial \mathbf{y}_{L-1}}. \tag{21}$$

For other layers, in general, we can write

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

where

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

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 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 $(x_1, \ldots, x_n)$,

$$\phi(x_1, \ldots, x_n) = \phi(x_{\pi(1)}, \ldots, x_{\pi(n)}). \tag{24}$$

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:

$$\phi(x_1, \ldots, x_n) = \phi(x_{\pi_a(1)}, \ldots, x_{\pi_a(m)}, x_{\pi_b(m+1)}, \ldots, x_{\pi_b(n)}). \tag{25}$$

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 $(\mathbf{u}^{(\xi)} \in \mathbb{C}^n : \xi \in \{1, 2, ..., N_0\})$. 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 $\alpha \in \{1, 2, ..., M_0\}$. Since our final neural network achieves equivariance, we require that for any unitary operator $\mathcal{U}$, the following holds:

$$\mathcal{U}\mathbf{D}^{(\alpha)}(\{\mathbf{u}^{(\xi)}\}) = \mathbf{D}^{(\alpha)}(\{\mathcal{U}\mathbf{u}^{(\xi)}\}). \tag{26}$$

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 Equation (25) can be:

$$\phi(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i w, \tag{27}$$

where w is some 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:

$$\phi(x_1, \ldots, x_n) = \sum_{\delta} \sum_{i=1}^{n} x_i v_{i\delta} w_{\delta}, \tag{28}$$

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,

$$D_i^{(\alpha)} = v'_{i\alpha} = \sigma \left(\sum_{j,\beta,\delta} e_{ij}^{\beta} v_{j\delta} w_{\beta\delta\alpha} \right) \tag{29}$$

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. 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

$$\mathbf{D}_i^{(\alpha)} = \mathbf{v}'_{i\alpha} = \boldsymbol{\sigma} \left(\sum_{j,\beta,\delta} \frac{\mathbf{u}_{ij}}{u_{ij}} e_{ij}^{\beta} v_{j\delta} w_{\beta\delta\alpha} \right), \tag{30}$$

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 cutoff 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

$$e_{ij}^{\beta} = \begin{cases} \exp(-\eta^{(\beta)}(r_{ij} - r_s^{(\beta)})^2) f_c(r_{ij}), & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases} \tag{31}$$

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. Equation 29 becomes:

$$D_i^{(\alpha)}(\{\mathbf{r}_j\}) = \sum_{j,j\neq i} \exp(-\eta^{(\alpha)}(r_{ij} - r_s^{(\alpha)})^2) f_c(r_{ij}), \tag{32}$$

where $\alpha \in \{1, 2, ..., M_0\}$. 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 interatomic potential.

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

$$\mathbf{D}_i^{(\alpha)}(\{\mathbf{r}_j\}) = \sum_{j,j\neq i} \frac{\mathbf{r}_{ij}}{r_{ij}} \exp(-\eta^{(\alpha)}(r_{ij} - r_s^{(\alpha)})^2) f_c(r_{ij}). \tag{33}$$

It is straightforward to check

$$\mathcal{U}\mathbf{D}_i^{(\alpha)}(\{\mathbf{r}_j\}) = \mathbf{D}_i^{(\alpha)}(\{\mathcal{U}\mathbf{r}_j\}), \tag{34}$$

which resembles equation 26. The subscribe j here is corresponding to the superscribe (ξ) of $\mathbf{u}$ in equation 26.

We are not going to show any example on this. It is only to show theoretically that one can introduce permutation symmetry by adding an extra layer of properly designed vector form GNN to our ENN.

3.6. Restrictions

Our implementation has strong restriction 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. 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

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} \tag{35}$$

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

$$\mathbf{L} = -\mathbf{r} \times -\mathbf{p}. \tag{36}$$

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 target and output data according to 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, such as $\mathbf{w}_p = f(\{\mathbf{x}_k\})$, where $\mathbf{w}_p \in \mathbb{C}^{M_p}$, and plugging them 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 and Parrinello, 2007) as mentioned in previous subsection, where energy has even parity symmetry.

4. Numerical Experiment

When the governing rules of a physical system are unknown, it is hard to apply any analytical methods to study the evolution of a system. A viable method nowadays is to adopt certain 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 (Kondor, 2018; Batzner et al., 2022; Behler and Parrinello, 2007) and moment tensor (Shapeev, 2016). Atomic positions and atomic energies are used as the input and target data, respectively. The atomic energies are usually obtained from density function theory (DFT) calculations (Hohenberg and Kohn, 1964; Kohn and 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 a measurable quantity.

We may ask two questions here. First, can we predict forces directly from

positions, without the need of knowing energies? This question is not limited to atomic scale modelling. 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 a ML machinery. We are going to use our ENN to show the possibility.

4.1. Model and data

We are going to simulate a system of 4-bodies motions governed by a model Hamiltonian. 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 adopted a pair-wise Lennard-Jones potential for Argon (Rahman, 1964):

$$U_{ij} = 4\epsilon \left(\left(\frac{r_0}{r_{ij}} \right)^{12} - \left(\frac{r_0}{r_{ij}} \right)^{6} \right), \tag{37}$$

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 figure 1.

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

$$U = \sum_{i,j,i>j} U_{ij}(r_{ij}), \tag{38}$$

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

$$\mathbf{F}_i = -\frac{\partial U}{\partial \mathbf{r}_i}. \tag{39}$$

We generated 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

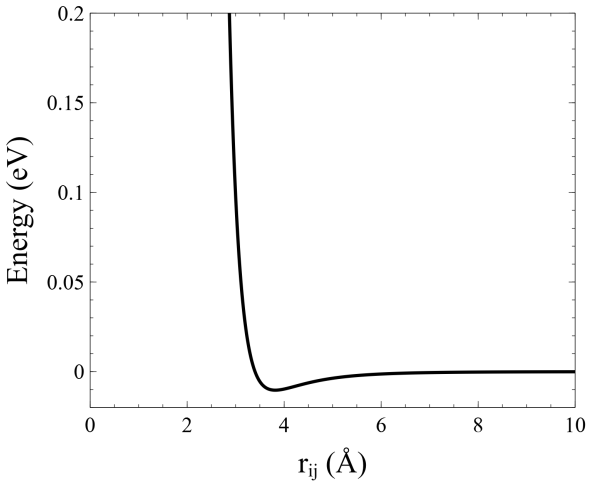

Figure 1: A plot of the Lennard-Jones potential for Argon according to equation 37.

is generated randomly according to Gaussian distribution with mean equals zero and standard deviation equals 3Å. 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 3Å. 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. It means we used

$$\mathbf{x}_0 = \{\mathbf{r}_{12}, \mathbf{r}_{13}, \mathbf{r}_{14}, \mathbf{r}_{23}, \mathbf{r}_{24}, \mathbf{r}_{34}\}. \tag{40}$$

This takes care of the translational symmetry.

The target data is simply the atomic forces

$$\mathbf{T} = \{\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3, \mathbf{F}_4\}. \tag{41}$$

We rescaled both the input and target data by their standard deviations before we use them to train a 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 target data are learned by a 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.

The Loss function is defined as

$$L = \frac{1}{N_{data}} \sum_{data} (\mathbf{T} - \mathbf{x}_L)^2, \tag{42}$$

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 and 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.

In Fig. 2, it shows the change of the value of the Loss function calculated using the training data and validation data. We performed 10 million iteration steps. We can see both of them drop significantly. As expected, the

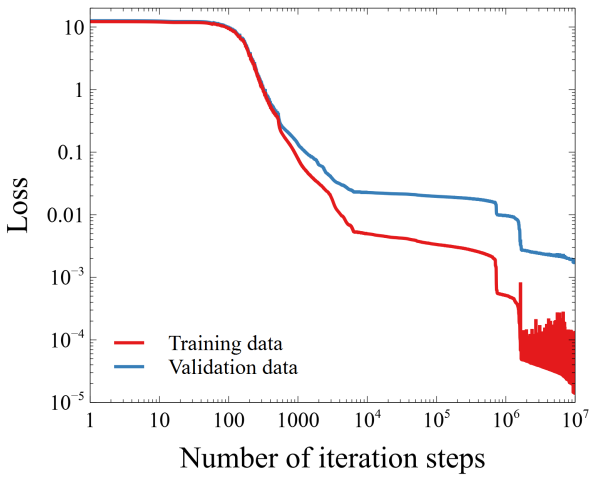

Figure 2: The value of Loss function (unitless) calculated using training data and validation data as a function of iterative steps using FIRE algorithm.

training loss drops more than validation loss. However, we can see both of them remain dropping. It deems to be not suffering from overfit. We stopped the iteration as we can observe fluctuations of 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 analytic and predict 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. As we can observe the training data that forces is in the order of 0.1 to 1 eV/Å, the average error is in the order of 0.001 eV/Å, the training is fairly satisfactory.

4.3. Dynamic simulations

We are going to use the forces predicted by our trained ENN to drive the evolution of a system of four Argon atoms using molecular dynamics (MD).

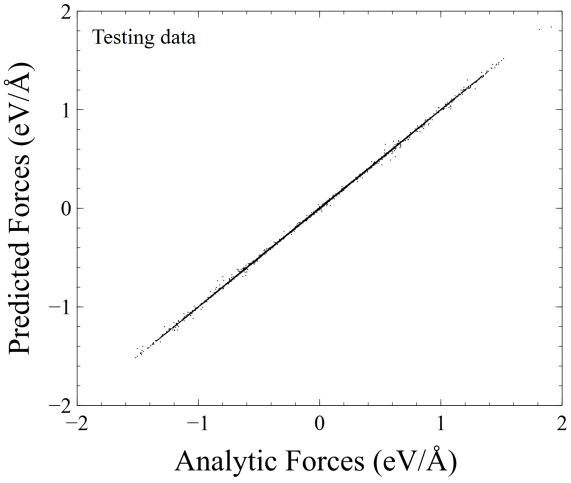

Figure 3: Each component of the atomic forces calculated analytically using the Lennard-Jones potential versus the prediction calculated using the trained ENN. They are calculated using the testing data.

We will also compare it with analytic solutions. We should note 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

$$\frac{d\mathbf{p}_i}{dt} = \mathbf{F}_i, \tag{43}$$

$$\frac{d\mathbf{r}_i}{dt} = \frac{\mathbf{p}_i}{m_i}, \tag{44}$$

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 analytic form of a Hamiltonian is know, the atomic force is

$$\mathbf{F}_i = -\frac{\partial \mathcal{H}}{\partial \mathbf{r}_i} \tag{45}$$

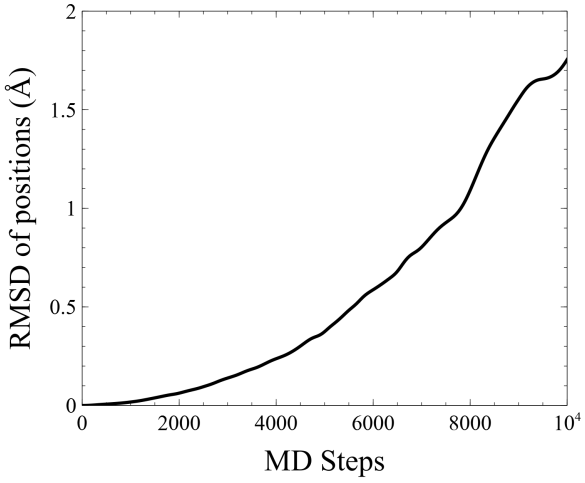

Figure 4: The root-mean-square-derivation (RMSD) of positions with respect to analytic solution and prediction by our trained ENN. The RMSD is calculated across 10 samples. Each sample contains 4 atoms.

where the a Hamiltonian:

$$\mathcal{H} = \sum_i \frac{\mathbf{p_i}^2}{2m_i} + U(\{\mathbf{r}_i\}). \tag{46}$$

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

We initialized ten samples. The positions of Argon 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 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 the Newton's equation using velocity Verlet algorithm. We used a time step of 1fs, which is a conventional value for MD simulations.

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

$$\text{RMSD}(\{\mathbf{r}_i\}) = \sqrt{\frac{1}{N_s N_{at}} \sum_{samples,atoms} (\mathbf{r}_i^a - \mathbf{r}_i^p)^2} \tag{47}$$

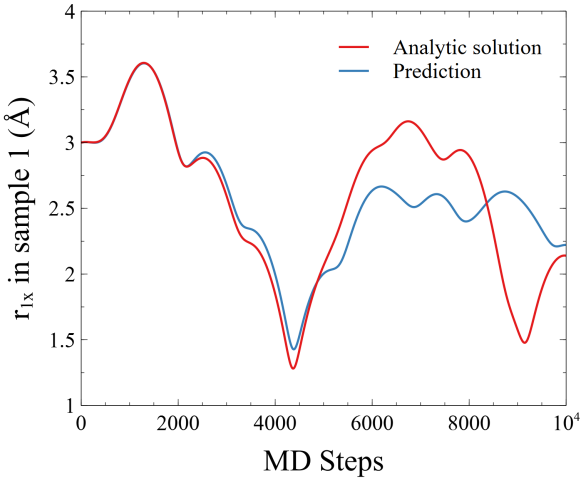

Figure 5: The x component of the position of atom 1 in sample 1. Analytic solution and prediction are shown.

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 analytic solution, and $\mathbf{r}_i^p$ is the position calculated using forces predicted by ENN.

In Fig. 4, it show 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. 5. It shows the x component of atom 1 in sample 1. We can 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

$$\mathrm{RMSD}(E) = \sqrt{\frac{1}{N_s} \sum_{samples} (E^a - E^p)^2} \tag{48}$$

where E^a and E^p are the total energy calculated using equation 46. The

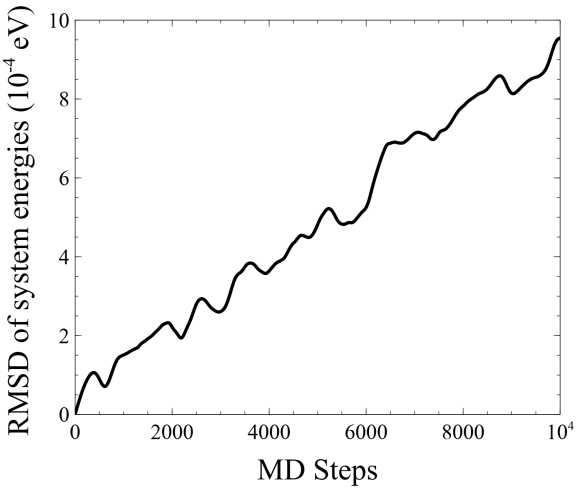

Figure 6: The RMSD of energy calculated using the positions calculated analytically or using ENN.

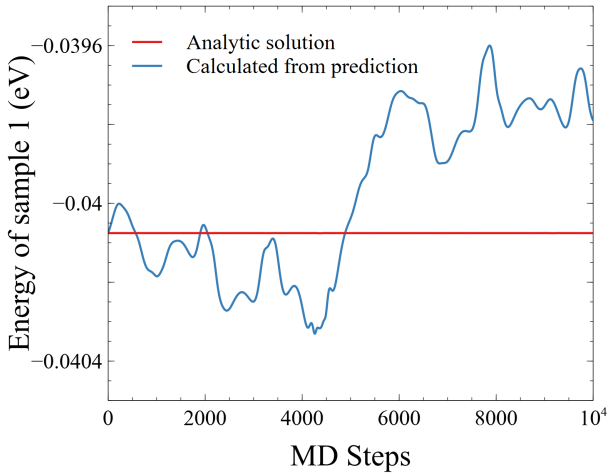

Figure 7: The system energy of sample 1. Analytic solution and prediction are shown.

potential energy are calculated using the positions of atoms that evolve according to forces calculated analytically or by prediction using ENN.

In Fig. 6, it shows the RMSD of energy across ten samples. Again, we can see deviation accumulating. However, we should know 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 even if we don't 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.

5. Conclusion

We have designed a new feedfoward ENN for unitary transformation. 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. An example on the dynamics of Argon atoms is given showing the practicality of our architecture via empirical simulations.

Declaration of competing interest

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

Acknowledgments

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 [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 and 17202121.

Appendix A. FIRE minimization algorithm

FIRE (fast inertial relaxation engine) (Bitzek et al., 2006) is a minimization algorithm commonly used in atomic scale simulation 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

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

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

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

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

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

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

In our case, we are minimising 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 treat 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. Phys. Rev. Lett. 104, 136403. URL: https://link.aps.org/doi/10.1103/PhysRevLett.104.136403, 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, 1–11.

Behler, J., Parrinello, M., 2007. Generalized neural-network representation of high-dimensional potential-energy surfaces. Physical Review Letters 98, 146401.

Bitzek, E., Koskinen, P., Gähler, F., Moseler, M., Gumbsch, P., 2006. Structural relaxation made simple. Phys. Rev. Lett. 97, 170201. URL: https://link.aps.org/doi/10.1103/PhysRevLett.97.170201, 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.

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., Welling, M., 2016. Group equivariant convolutional networks, in: Balcan, M.F., Weinberger, K.Q. (Eds.), Proceedings of The 33rd International Conference on Machine Learning, PMLR, New York, New York, USA. pp. 2990–2999. URL: https://proceedings.mlr.press/v48/cohenc16.html.

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.

Esteves, C., Allen-Blanchette, C., Makadia, A., Daniilidis, K., 2020. Learning SO(3) equivariant representations with spherical CNNs. International Journal of Computer Vision 128, 588–600. URL: https://doi.org/10.1007/s11263-019-01220-1, doi: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. URL: https://arxiv.org/abs/2105.13926, doi:10.48550/ARXIV.2105.13926.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks, in: Teh, Y.W., Titterington, M. (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR, Chia Laguna Resort, Sardinia, Italy. pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

Hohenberg, P., Kohn, W., 1964. Inhomogeneous elec-

tron gas. Phys. Rev. 136, B864–B871. URL: https://link.aps.org/doi/10.1103/PhysRev.136.B864, doi:10.1103/PhysRev.136.B864.

Kohn, W., Sham, L.J., 1965. Self-consistent equations including exchange and correlation effects. Phys. Rev. 140, A1133–A1138. URL: https://link.aps.org/doi/10.1103/PhysRev.140.A1133, doi:10.1103/PhysRev.140.A1133.

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

Kondor, R., Lin, Z., Trivedi, S., 2018. Clebsch–Gordan nets: a fully fourier space spherical convolutional neural network, in: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/a3fc981af450752046be179185ebc8b5-

Kondor, R., Trivedi, S., 2018. On the generalization of equivariance and convolution in neural networks to the action of compact groups, in: Dy, J., Krause, A. (Eds.), Proceedings of the 35th International Conference on Machine Learning, PMLR. pp. 2747–2755. URL: https://proceedings.mlr.press/v80/kondor18a.html.

Müller, P., Golkov, V., Tomassini, V., Cremers, D., 2021.

Rotation-equivariant deep learning for diffusion MRI. URL: https://arxiv.org/abs/2102.06942, doi:10.48550/ARXIV.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.

Rahman, A., 1964. Correlations in the motion of atoms in liquid argon. Phys. Rev. 136, A405–A411. URL: https://link.aps.org/doi/10.1103/PhysRev.136.A405, doi:10.1103/PhysRev.136.A405.

Satorras, V.G., Hoogeboom, E., Welling, M., 2021. E(n) equivariant graph neural networks, in: ICML, PMLR. pp. 9323–9332.

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, 241722. URL: https://doi.org/10.1063/1.5019779, doi: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 & Simulation 14, 1153–1173. URL: https://doi.org/10.1137/15M1054183, doi:10.1137/15M1054183, arXiv:https://doi.org/10.1137/15M1054183.

Sonoda, S., Murata, N., 2017. Neural network with unbounded activation functions is universal approximator. Ap-

plied and Computational Harmonic Analysis 43, 233–268. URL: https://www.sciencedirect.com/science/article/pii/S1063520315001748, doi:https://doi.org/10.1016/j.acha.2015.12.005.

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. URL: https://arxiv.org/abs/1802.08219, doi:10.48550/ARXIV.1802.08219.

Weiler, M., Geiger, M., Welling, M., Boomsma, W., Cohen, T.S., 2018. 3D steerable CNNs: Learning rotationally equivariant features in volumetric data, in: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/488e4104520c6aab692863cc1dba45af-

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: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Computer Vision – ECCV 2018, Springer International Publishing, Cham. pp. 585–602.