Learning inter-atomic forces with an SE(3)-Transformer

Bryce E. Hedelius

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Dennis Della Corte, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Learning inter-atomic forces with an SE(3)-Transformer

Bryce E. Hedelius
Department of Physics and Astronomy, BYU
Bachelor of Science

Density-functional theory (DFT) is the gold standard for quantum chemistry calculations but is prohibitively expensive to use in molecular dynamics simulation. Deep learning has the potential to speed up these calculations by five orders of magnitude. Atomistic systems may be represented as point clouds so SE(3) equivarient neural networks (graph networks that preserve translational and rotational symmetry) provide the appropriate architecture. I present several SE(3)-Transformer models designed to predict DFT energy and forces. Given a point cloud of atoms, the network predicts the overall energy and the net force on each atom. Unlike other SE(3) networks, the SE(3)-Transformer architecture has attention weights that add degrees of freedom in the angular direction. I demonstrate that the network can learn inter-atomic energy and forces.

Keywords: density-functional theory, deep learning, inter-atomic potential, force fields, molecular dynamics

ACKNOWLEDGMENTS

First and foremost, I have to thank my research advisor, Dennis Della Corte, whose expertise was invaluable while performing this research. Thank you for your support and understanding over these past three years.

I would also like to thank Fabian Fuchs, whose conversations helped me to understand and develop the methods I used in this thesis.

Most importantly, none of this could have been possible without my family. I'm grateful to them for encouraging me to pursue my goals and supporting me in the process.

Contents

List of Figures

Chapter 1

Introduction

Predicting the properties of an atomistic system is an outstanding problem in computational chemistry [1]. All properties of atomistic systems are determined from first principles but it is usually impossible to solve first principle calculations analytically [2] [3]. Instead, these calculations require careful approximations and numerical methods. My hypothesis is an SE(3)-Transformer [4] [5]–a novel machine learning algorithm built around the symmetries of Euclidean space–can approximate first principle methods with minimal loss of accuracy.

Molecular dynamics (MD) is a common method of studying systems containing thousands of atoms with computer simulations. The trajectories of atoms and molecules are determined by Newton's equations of motion. The forces are calculated by taking the negative gradient of the inter-atomic potential–a mathematical function to describes the potential energy of an atomistic system–with respect to the position.

Force fields are a method of calculating the inter-atomic potential. Force fields consist of a functional form and a set of parameters, which are usually derived from experimental studies and quantum calculations. The parameters allow you to bypass the expensive integral calculations that first principle calculations require but can substantially degrade the accuracy.

Density functional theory (DFT) [6] is another method of calculating the inter-atomic potential.

It is based on first principles and is considered to be the gold standard for quantum chemistry calculations. Unfortunately, accurate DFT calculations are computationally expensive [7], limiting their application.

Deep learning has proven to be a useful approximation for DFT energy and force calculations [8]. Additionally, it much faster than DFT calculations. My goal is to develop and test a deep learning model that predicts DFT forces and energies.

1.1 Background Information

1.1.1 Molecular Dynamics

Molecular dynamics is a computational method of studying the movement of atom and molecules [9] [10] [11]. Molecular dynamics software calculates the forces on the atoms and iteratively uses numerical methods to determine the positions in a future time step [12].

The forces are commonly calculated by taking the negative gradient of an inter-atomic potential function:

$$\vec{f} = -\nabla V(\vec{r}). \tag{1.1}$$

Forcefields are a type of inter-atomic potential commonly used in molecular dynamics and have a functional form that is usually similar to [13] [14] [15] [15]:

$$V = \sum_{\text{bonds}} k_b(b-b_0) + \sum_{\text{angle}} k_\theta(\theta-\theta_0)^2 + \sum_{\text{dihederals}} k_\phi\left[1+\cos(n\phi-\theta)\right]. \tag{1.2}$$

This functional form has terms for equilibrium and spring constant terms.

The positions are updated using Newton's equations of motion by an algorithm called the integrator [16]. Integrators should obey conservation of energy, be reversible, and be efficient [17] [18] [19]. I'll derive and use the simplest integrator, called the verlet integrator [20]. The Taylor

expansion of positions can be written as

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \frac{d\vec{x}}{dt}\Delta t + \frac{d^2\vec{x}}{dt^2}\frac{\Delta t^2}{2} + \mathscr{O}(\Delta t^3) = \vec{x}(t) + \vec{v}\Delta t + \frac{\vec{f}}{m}\frac{\Delta t^2}{2} + \mathscr{O}(\Delta t^3). \tag{1.3}$$

Velocity can be approximated by

$$\vec{v}_{n+1} = \frac{\vec{x}_n - \vec{x}_{n-1}}{\Delta t} \tag{1.4}$$

so the integrator algorithm can be expressed as

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \frac{1}{2m}\vec{f}_n\Delta t^2. \tag{1.5}$$

1.1.2 Density Functional Theory

Density functional theory (DFT) is a quantum mechanical computational method of studying many-atom systems [21]. Rather than considering the electrons individually, it considers their overall density. The properties of many-electron systems can then be determined using functionals (functions of another functions) of the electron density [21].

DFT is based on the Born-Oppenheimer approximation [22], which says that electrons move much faster than the nuclei, so the nuclei are treated as being stationary.

The Hamiltonian [23] for electrons in an atomistic system can be written as:

$$\hat{H} = \left[\hat{T} + \hat{V} + \hat{U}\right]\Phi = \left[-\left(\frac{h}{2\pi}\right)^2 \sum_{i=1}^{N}\left(\frac{1}{2m_i}\nabla_i^2\right) + \sum_{i=1}^{N} V(\vec{r}_i) + \sum_{i<j}^{N} U(\vec{r}_i, \vec{r}_j)\right]\Psi. \tag{1.6}$$

The operators represent the different contributions to energy: $\hat{T}$ is the kinetic energy, $\hat{V}$ is the to nuclei-electron potential energy, and $\hat{U}$ is the electron-electron interaction energy [24].

An atomistic system with N electrons would have a wavefunction with $3N$ components (disregarding spin) [25]. The electron density $n(\vec{r})$, defined as

$$n(\vec{r}) = N\int d^3\vec{r}_2 ... \int d^3\vec{r}_N \Psi^*(\vec{r}, \vec{r}_2, ..., \vec{r}_N)\Psi(\vec{r}, \vec{r}_2, ..., \vec{r}_N), \tag{1.7}$$

contains 3 components. Making this transformation significantly reduces the number of components so might seem reasonable that information in lost in the process. Surprisingly, this is not the case.

The Hohenberg-Kohn theorems [26] demonstrate that the wavefunction formulation has the same information as the density formulation. Therefore, the DFT formulation is equivalent to Schrodinger's equation [27].

1.1.3 Deep Learning

Deep learning [28] is a subset of machine learning that can solve more complicated problems compared to other machine learning algorithms. Deep learning is based on artificial neural networks (hereafter referred to as neural networks or networks), which are inspired by neural systems in biology. Each neuron is a mathematical function which accepts a vector as inputs and outputs a scalar. Typically, these neurons are arranged in layers. All neurons in a layer accepts the same vector as input and the outputs of all the neurons are concatenated to create a new vector. Since the input and output are both vectors, the output of one layer can be fed into another. When a network has many layers it is called a deep neural network.

A neuron consists of three steps . First, the components of the input vector are used to generate a scalar by taking the scalar product between the input vector and a weight vector unique to that neuron. Next, the scalar output of the first layer is added to a scalar bias unique to the neuron. Finally, the output of the second step passes through a nonlinear function known as the activation function. Mathematically, a neuron $N : \mathbb{R}^m \to \mathbb{R}$ is defined as

$$N(\vec{x}) = f(\vec{w} \cdot \vec{x} + b) = f\left[\left(\sum_{i=1}^{m} w_i x_i\right) + b\right] \tag{1.8}$$

where $\vec{x}$ is the input vector, $\vec{w}$ is the weight vector, b is the scalar bias, and f is the activation function [29].

Working with each neuron individually can be tedious and computationally inefficient so, as previously mentioned, a layer may be formed from a set of neurons. The layer $l : \mathbb{R}^m \to \mathbb{R}^n$ is

defined as:

$$L(\vec{x})_i = f(\mathbf{W}\cdot\vec{x}+\vec{b})_i = f\left[\left(\sum_{j=1}^{m} W_{ij}w_j\right)+b_i\right]. \tag{1.9}$$

This paradigm allows for efficient computation and is easier to work with than considering each neuron individually.

A nonlinear activation is essential to learning more complicated representations. For sake of argument, assume that the activation is the identity function: $f(x) = x$. Then the result of two consecutive layers is

$$g(x) = L_2(L_1(x)) = \mathbf{W}_2\cdot(\mathbf{W}_1\cdot\vec{x}+\vec{b}_1)+\vec{b}_2 = (\mathbf{W}_2\cdot\mathbf{W}_1)\cdot x+(\mathbf{W}_2\cdot\vec{b}_1+\vec{b}_2) = \mathbf{W}\cdot x+b, \tag{1.10}$$

which is another linear layer with different parameters. In general, if the activation is any linear operation, the result will still be a linear layer. Therefore, stacking many linear layers isn't any more powerful than a single linear layer.

Many activation functions exist [30] [31] [32] but the most commonly used one (and the only one I used in my network) is the Rectified Linear Unit (ReLU) [33]. The $\text{ReLU} : \mathbb{R} \to \mathbb{R}$ activation function is defined as

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}. \tag{1.11}$$

A plot of the ReLU function is shown in Fig. 1.1.

A neural network is a chain of neuron layers, where the output of a layer is the input to the next. Collectively, the weights and biases are known as the parameters. The parameters are random at the start of training. The parameters are adjusted to useful value during a process known as learning. In supervised learning, the algorithm is given labeled data, which consists of the inputs and the labels. The optimizer algorithm adjusts the parameters so that the network outputs are closer to the true values. The loss function says how close the networks outputs are to the true values. The most common loss function used in regression problems is mean squared error (MSE), which is the

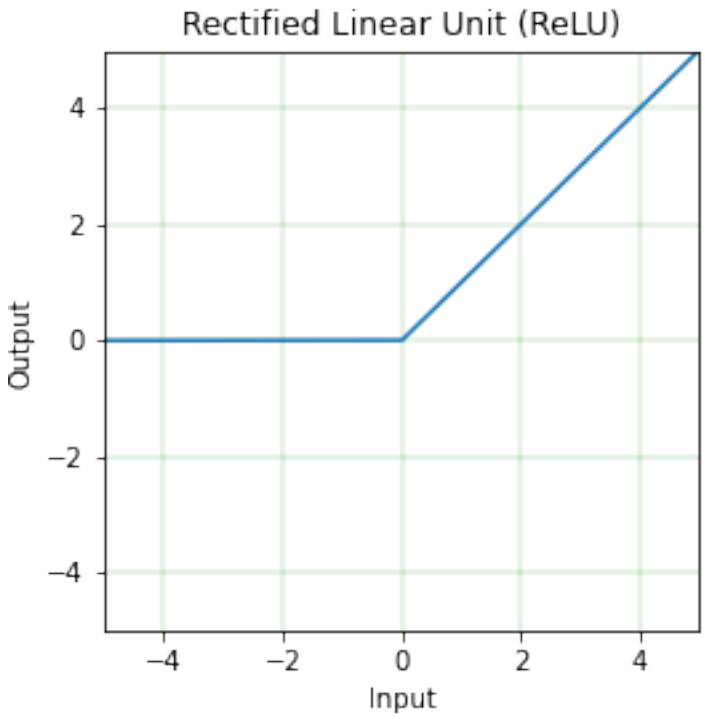

Figure 1.1 Plot of the Rectified Linear Unit commonly used as an activation function in neural networks

function MSE : $\mathbb{R}^n \to \mathbb{R}$ define by

$$\text{MSE}(Y) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1.12}$$

where Y is the output of the network and $\hat{Y}_i$ is the label values.

Most optimizer algorithms determine how much each parameter contributes to the network error by calculating the gradient of the error with respect to the parameters [34]. When gradients are usually calculated mathematically, an analytic expression for the gradient is derived and all the appropriate values are plugged in. This algorithm is infeasible for deep neural networks because the number of terms in each gradient function grows with the number of neurons and layers which is why neural networks weren't been widely used until recently [35]. One of the fundamental advancements in deep learning was the development of the backpropogation algorithm [36].

Backpropogation calculates the gradient by going backward in the computational graph and makes uses of the gradients of successive layers, hence its name. It is based on the chain rule, which says how the gradient of values in a composite function contribute to the gradient of the input. Thus, values that would occur in different gradient calculations only are calculated once.

The optimizer uses the gradient values to optimize the parameters. The simplest optimizer is stochastic gradient descent [37], which multiplies a parameter's gradient by a constant called the learning rate, then decreases the parameter by that value. This process is called a learning step.

All optimizer are based off of stochastic gradient descent optimization. The weights are updated with

$$w := w - \eta \nabla Q_i(w) \tag{1.13}$$

where η is the learning rate and $\nabla Q_i(w)$ is the gradient of the objective with respect to the parameter w. The optimizer I used is called Adam (short for Adaptic Moment Estimate) [38]. It estimates the second moments of the gradients, enabling it to compensate for the curvature of the parameter space and find a faster route to the minimum.

Attention [39] is a technique used in deep learning that mimics cognitive attention. It amplifies the important parts of the input data and quiets the rest. Transformers are a neural network architecture that uses a form of attention called scaled dot-product attention. The transformer learns three weight matrices: the query weights W_K, the key weights W_K, and the value weights W_V. The attention is then calculated as

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{W_Q W_K^\intercal}{\sqrt{d_k}}\right) W_V \tag{1.14}$$

where d_k is the dimension of the key vectors and softmax : $\mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$\text{softmax}(\vec{x}) = \frac{e^{\vec{x}}}{\sum_{i=1}^{n} e^{x_i}}. \tag{1.15}$$

1.1.4 Graph Neural Networks

In mathematics, a graph is a pair $G = (V,E)$, where V is a set of vertices and E is a set of vertex pairs, or edges [40]. Nodes that are connected by an edge are called adjacent. Graphs are useful in a variety of fields, such as computer science, linguistics, physics, and chemistry, social science, and biology [41]. In my approach, I represent the atomic systems as a graphs. Nodes represent atoms and edges represent inter-atomic interactions.

Graph neural networks (GNNs) are deep learning models that operate on data in a graph [42] [41]. Convolutional operations on a graph are defined in an analogous way to the convolutional operations in a convolutional neural network [43]: information from nodes are passed to adjacent nodes in graph neural networks just how information from pixels are passed to adjacent pixels in a convolutional neural network. The process of transferring information between nodes is known as message passing. After the messages are passes, the message are aggregated to update the data on the node. An example of a graph network is shown in Fig. 1.2.

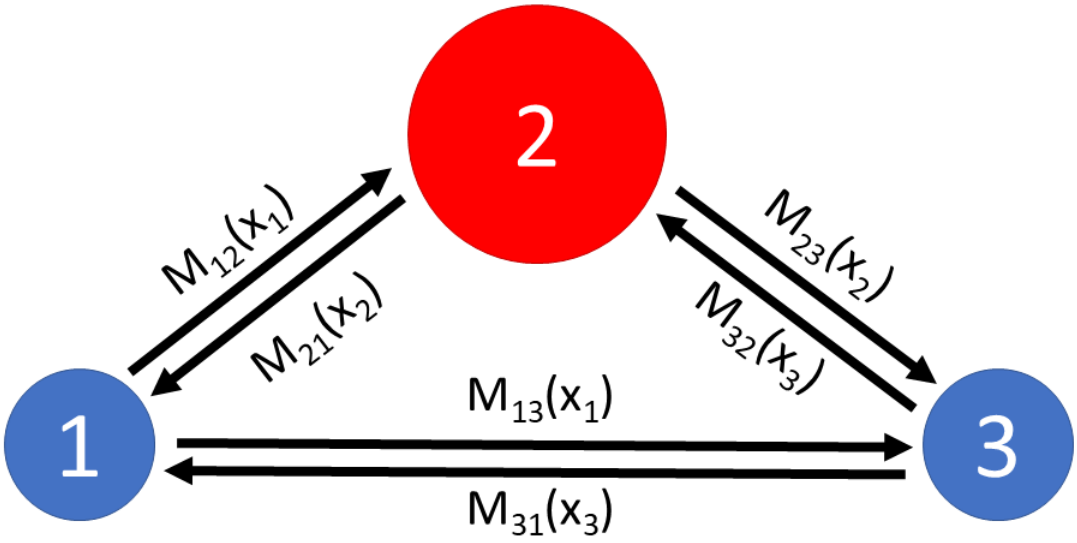

Figure 1.2 Architecture of a graph neural network for a water network. Here, the blue balls represent hydrogen and the red ball represents oxygen. The input data x_j associated with atom j is used to calculate the message vector, $M_{ji}(x_j)$, which gets passed to node i. The way the message vector is generated depends on the type of graph neural network. The message vectors $M_{jk}(x_j)$ are then aggregated (usually by summing or averaging) at each destination node and combined with the original input vector x_i to calculate the new x_i.

1.1.5 Group Theory

Groups are a mathematical objects that are useful for representing certain relationships [44]. They find use in physics, chemistry, computer science, and even music.

Definition 1.1.1 *A group G is a set along with a binary operation $\cdot$ on G that satisfies the following properties:*

- *Closure: For all elements $A, B \in G$, $A \cdot B \in G$.*
- *Associativity: For all elements $A, B, C \in G$, $(A \cdot B) \cdot C = A \cdot (B \cdot C)$.*
- *Identity: There exists an element $I \in G$ such that for all $A \in G$, $I \cdot A = A \cdot I = A$.*
- *Inverse: For each element $A \in G$, there exists an inverse $B \in G$ such that $A \cdot B = B \cdot A = I$.*

It is possible to represent many groups and their operators as a set of matrices and matrix multiplication. This is useful because it allows a group-theoretic problems to be reduced to linear algebra problems.

Definition 1.1.2 *A representation is a map $\rho : G \to GL(N)$* [1] *such that*

$$\rho(g_1 g_2) = \rho(g_1)\rho(g_2) \tag{1.16}$$

for all $g_1, g_2 \in G$.

A useful class of functions are those that are symmetric under the group. That is, if you get the same result whether you put rotated inputs into the network or you rotated the output of the network raw inputs. These functions are called equivariant functions.

[1] The general linear group $GL(N)$ is the set of invertible $N \times N$ matrices. The binary operation of such a representation is matrix multiplication. In general, the matrix can be defined over any field but I will only be using the real numbers.

Definition 1.1.3 *Let X and Y be representations of the group G. The function $f : X \to Y$ is equivariant if*

$$f(g \cdot x) = g \cdot f(x) \tag{1.17}$$

for all $g \in G$ and all $x \in X$.

Since the algebraic properties of group and its representations are equivalent, a group can be studied by studying its representations [45]. A representation D can be decomposed if there exists a similarity transformation P such that

$$D'(g) = P^{-1}D(g)P \tag{1.18}$$

for all $g \in G$ which diagonalizes every matrix D in the representation into the same pattern of diagonal blocks D' [46]. Each matrix in the block is also a representation of the group. Any non-zero representation that cannot be decomposed into two non-trivial representations is call an irreducible representation or irrep.

In linear algebra, matrices are linear transformations [47]. Similarly, in group representation theory, representation matrices transform a vector space [48]. For example, a vector $\vec{x}$ in $\mathbb{R}^3$ can be rotated about the z-axis by some angle θ with the matrix

$$S = \begin{bmatrix} \cot\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.19}$$

and the transformation would be written as

$$\vec{x}' = S\vec{x}. \tag{1.20}$$

Since atoms live in Euclidean space, which is rotationally and translationally symmetric, I'm interested in the special Euclidean group in $\mathbb{R}^3$, written as $SE(3)$, which consists of the proper

rotations and translations respectively in 3D space [49]. The group $SO(3)$ is a subgroup of $SE(3)$ that consists of the proper rotations in $\mathbb{R}^3$, has the property that irreducible representations of order l are $(2l+1)\times(2l+1)$ orthogonal matrices with determinant 1.

The spherical harmonics $Y_l^m : S(2) \to \mathbb{R}$ are a set of basis functions on the sphere [50] [51]. They can be found by solving Laplace's equation $\nabla^2 u = 0$ in spherical coordinates. The spherical harmonics provide a set of basis functions for the irreducible representations of $SO(3)$ of order l.

Two irreducible representations of $SO(3)$ may be combined using a tensor product. Clebsch-Gordan decomposition [52] transforms the result from the product basis to the irrep basis. If $D_k(g)$ and $D_l(g)$ are irreps of $g \in G$ of orders k and l respectively, then their tensor product may be expressed in the irrep basis as:

$$D_k(g) \otimes D_l(g) = Q^{lk\top} \left[\oplus_{J=k-l}^{k+l} D_J(g) \right] Q^{lk} \tag{1.21}$$

where the change of basis matrices Q^{lk} are called the Clebsch-Gordan coefficients [52], $\otimes$ is the direct product, and $\oplus$ is the direct sum. The direct sum results in a block matrix if the operands are matrices, otherwise the direct sum results in a concatenation vector if the operands are vectors.

1.2 Situation of general area of research

Several networks exist that approximate the inter-atomic potential and have been used in molecular dynamics simulations [53] [54] [55] [56] [57] [58]. The ANI-1 network [8] encodes the geometric information of the system using symmetric functions to create the input vectors. The input vectors are then fed through different multilayer perceptrons (MLP's) to estimate the energy associated with that atom. The outputs from all the MLP's are added to get the total energy. Next, using the standard backprop algorithm, the gradients are calculated from the energy back to the the positions of the atoms to get the forces.

Since each atom type has its own MLP, ANI-1 isn't able to generalize between different atom

types [59]. In order to add another atom type, you would have to train a new network. Most other existing networks have the same property. My method can be easily extended to include more atom types because atomic species is an input feature rather a fundamental change to the architecture. Thus, my method maybe able to improve on one of the weaknesses of the ANI-1 network.

Chapter 2

Methods

2.1 Existing methods

A variety of machine learning approaches are emerging to predict inter-atomic potentials and forces. The most common approaches involve neural networks, gaussian approximation, kernel ridge regression, or moment tensor potentials [54]. These different approaches have been used to model material science, chemical reactions, and phase changes. I'll focus on the neural network methods.

First, the chemical environment of the atom using symmetry functions [60]. The vector is then passed through a network corresponding to the atoms' species. The outputs of all the networks are added together to produce the potential. An example of an MLP network are the ANAKIN-ME (Accurate NeurAl networK engINe for Molecular Energies) networks or ANI for short. Accompanying the ANI-1x network is a database of about 5 million molecular positions along with their energies and forces which I used for my network.

Most of the neural network methods predict inter-atomic potentials. These methods are then predict the forces indirectly using the relationship $\vec{f}(\vec{r}) = -\nabla V(\vec{r})$ where $\vec{f}(\vec{r})$ is the vector of forces and $V(\vec{r})$ is the potential. The method of taking the gradient of the potential to find the force works

across all networks topologies because all neural networks are end-to-end differentiable. A major benefit to using a potential network is that the forces are ensured to be conservative.

However, there are few main drawbacks to using the gradient of potential networks to calculate forces. First, the gradient calculation increases computational cost because the network must feed forward before the gradient is propagated backward. Second, the network isn't actually trained on the forces, so there is no guarantee that the forces are accurate. Third, the output of potential networks is a single scalar whereas the output of force networks is an array of vectors. Therefore, a force network has more values to predict, leading to a better informed gradient while training the network. Few networks have been trained this way [61].

Within the last few years, the field of SE(3)-equivarient graph neural networks has emerged. Several of the different methods have been used on toy models for inter-atomic potentials and inter-atomic energies. The Tensor Field Network was able to learn inter-atomic forces and energies for simple systems [62]. The SE(3)-transformer [4] was able to learn the same energies as the Tensor field network with a 50% increase in accuracy and was used in N-Body simulations. I base my approach on the SE(3)-transformer.

2.2 Approach

I base my method on the SE(3)-Transformer [4]. This network has two main layers types that I describe below. The networks makes extensive use of group theory and group theory to ensure that it preserves symmetries. For more information on $SE(3)$ theory, refer to the introduction 1.1.5

2.2.1 Equivariant kernels and attention

Kernels

The SE(3)-Transformer uses equivarient kernels as its basis. A basis kernel $W_J^{lk}(\vec{x})$ is a matrix of size $\mathbb{R}^{(2l+1)\times(2k+1)}$ defined for a vector $\vec{x}$ as follows:

$$\mathbf{W}_J^{lk}(\vec{x}) = \sum_{m=-J}^{J} Y_{Jm}(\vec{x}/||\vec{x}||)Q_{Jm}^{lk}. \tag{2.1}$$

where $Y_{Jm}(\vec{x}/||\vec{x}||)$ are the spherical harmonics of degree J and Q_{Jm}^{lk} are Clebsch-Gordan matrices 1.1.5. The basis kernels have no learnable parameters.

Basis kernels are combined with a MLP radical basis function $\psi_J^{lk} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ to produce a kernel matrix $\mathbf{W}^{lk}$ of size $\mathbb{R}^{(2l+1)\times(2k+1)}$ using

$$\mathbf{W}^{lk}(x) = \sum_{J=|k-l|}^{k+l} \psi_J^{lk}(||\vec{x}||)\mathbf{W}_J^{lk}(\vec{x}). \tag{2.2}$$

Attention

The power of the SE(3) transformer lies in the attention mechanism, which was shown by Fuchs et. al to increase a network's accuracy.

First, a query vector q_i associated node i is calculated according to

$$q_i = \bigoplus_{l\geq 0} \sum_{k\geq 0} \mathbf{W}_Q^{lk} f_{\text{in},i}^k \tag{2.3}$$

where $\bigoplus$ is the direct sum and $f_{\text{in},i}^k$ are the input features of degree k associated with node j.

Next, a set of key vectors $\{k_{ij}\}$ between each node j adjacent to node i is computed with

$$k_{ij} = \bigoplus_{l\geq 0} \sum_{k\geq 0} W_K^{lk}(\vec{x}_j - \vec{x}_i) f_{\text{in},j}^k. \tag{2.4}$$

Next, the query and key vectors are used to calculate the attention weight α_{ij} analogous to how attention is calculated in vanilla transformer architectures.

$$\alpha_{ij} = \frac{\exp(q_i^\intercal k_{ij})}{\sum_{j' \in \mathcal{N}_i i} \exp(q_i^\intercal k_{ij'})} \tag{2.5}$$

Here, $\mathcal{N}_i$

i represents the set of nodes that are adjacent to i and are not i.

Attentive self-interaction

The attentive self-interaction weights $w^{ll}_{i,c'c}$ are defined by

$$w^{ll}_{i,c'c} = \text{MLP}\left(\bigoplus_{c,c'} f^{l\intercal}_{\text{in},i,c'} f^{l\intercal}_{\text{in},i,c}\right). \tag{2.6}$$

where *MLP* is a multilayer perception.

These weights are then used to calculate the attentive self-interaction using

$$f^{l}_{\text{out},i,c'} = \sum_{c} w^{ll}_{i,c',c} f^{l}_{\text{in},i,c}. \tag{2.7}$$

2.2.2 SE(3) Transformer Layer

Using the weight kernels, attention mechanism, and self-interaction, the layer is defined as

$$f^{ll}_{\text{out},i} = \mathbf{W}^{ll}_V f^{l}_{\text{in},i} + \sum_{k \geq 0} \sum_{j \in \mathcal{N}_i i} \alpha_{ij} \mathbf{W}^{lk}_V (\vec{x}_j - \vec{x}_i) f^{K}_{\text{in},j}. \tag{2.8}$$

Like other transformer architectures, I have multiple of these layers together in parallel to form a multihead attention block. I concatenate the outputs from each head then apply an equivarient linear layer.

2.2.3 Nonlinear norm layer

I also used the nonlinear norm layer from the Tensor Field Network paper [62], defined as

$$f_{out} = \text{ReLU}\left(\text{LN}\left(||f_{\text{in}}||\right)\right) \cdot \frac{f^l}{||f^l||} \tag{2.9}$$

where ReLU is the Rectified Linear Unit commonly used in deep learning and LN is a layer norm.

2.3 Development of method

I used the ANI-1x dataset [63] because it has the energy forces for about 5 million different molecular configurations. The molecules in the dataset contain the elements C, H, N, and O.

I implemented the SE(3)-Transformer [4] using PyTorch and the Deep Graph Library [42]. The network uses the position of atoms to create the kernels and the chemical species are converted into a one-hot encoding that is fed into the network. Atomic species are invariant to rotation in Euclidean space so the inputs for the network were all of order 0. The network consisted of 3 of the SE(3) Transformer layers with 8 heads and 4 channels each. The input to the first layer is tensors of order 0. The other layers used tensors of order 0, 1, and 2. The final layer outputs tensors of order 0 and 1, corresponding to potential and forces respectively. This is represented in fig 2.1.

2.4 Challenges encountered and resolution

The biggest challenge I encountered was ensuring that the outputs from each layer respect the symmetry of the network. An equivariance bug is especially difficult to locate because not every piece of an equivariant function is equivariant. While testing the implementation, I found that our SE(3)-Transformer layer was invariant but not equivariant, which led me to conclude that the problem was likely in the Clebsch-Gordan matrices, which were used to calculate the kenrels.

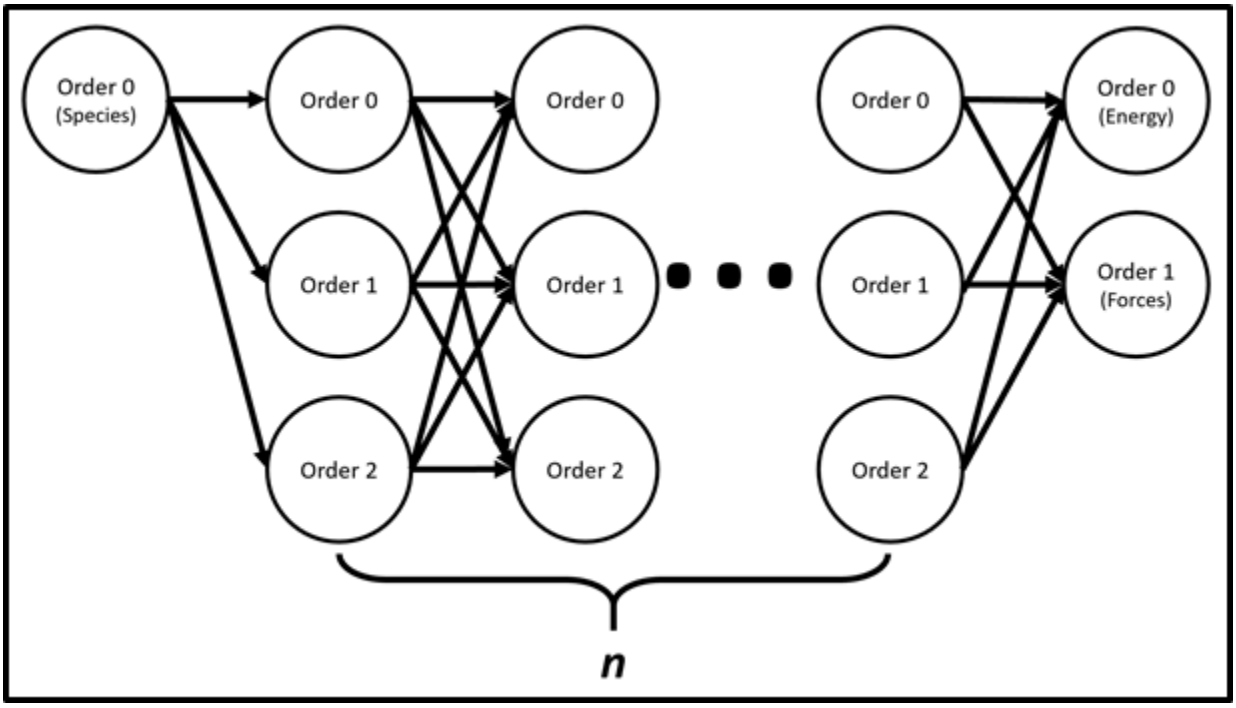

Figure 2.1 Architecture of the SE(3)-Transformer network. The input to the network is an embedding of the atomic species. The first layer transforms the species into three tensors of orders 0, 1, and 2. That is followed by 2 additional sets of layers that change the order of the input and output tensors. If the outputs of layers in a set are the same type, the outputs are concentrated, increasing the number of channels. The final layer transformers the input to tensors of order 0 and 1, representing energies and forces respectively. The energies are summed over the nodes to calculate the total energy.

After switching to the same implementation of Clebsch-Gordan matrices as the original SE(3)-Transformer, the problem was resolved.

While training, I discovered that the network is inheritantly unstable because the spherical harmonics have undefined derivatives as certain places. After training for a while, the errors would accumulate and the network would no longer produce a numerical result, halting training. I opted to used a shorter network. Based off a suggestion from Fabian Fuchs, I clipped the gradient norm so that the gradient values didn't become too large. If the losses produced nonnumerical values, the optimization step was skipped. However, this solution is somewhat of a hack. Further work is needed to improve the stability of the network.

Chapter 3

Results

3.1 Evaluating network implementation

I implemented the methods from chapter 2. I tested all layers for equivalence by calculating their equivariance errors Δ_{EQ}, which is defined as

$$\Delta_{EQ} = ||L\Phi(f) - \Phi L(f)||_2 / ||L\Phi(f)||_2 \tag{3.1}$$

where L represents a roto-translational transformation, Φ represents a network layer, f is the input to the network, and $||.||_2$ is the L_2 norm.

A layer is said to be equivariant if $\Delta_{EQ} = 0$ for all transformations L. In practice, the errors are nonzero due to round off error. I used float32, which has a machine epsilon of 1.19×10^{-7} and all the layers had equivarient errors smaller than 10^{-6} so my implementation is equivarient.

I trained both my implementation and Fabian's implementation [4] on a toy dataset to ensure that they perform similarly. The dataset is a set of 50000 trajectories from 5-body simulations of charged particles. The simulations were run for 4901 time steps and saved every 100 time steps so each simulation has 50 saved points. I trained both models to use the current positions, velocities, and charges to predict the future relative positions and velocities 500 time steps into the future. My

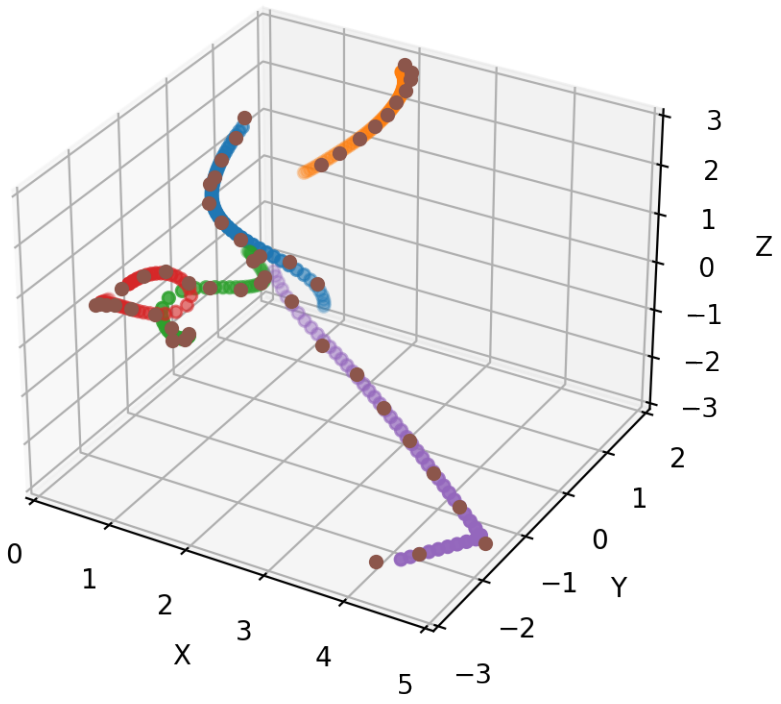

Figure 3.1 This plot shows true trajectories of five charged particles and the trajectories predicted by an SE(3)-Transformer. The true trajectories are red, green, blue, orange, and purple and the predicted trajectories are brown. The true trajectories are shown every 100 time steps over 4901 time steps and the predicted trajectories are predicted every 500 time steps.

model achieved an MSE of 0.0389 and Fabian's implementation achieved an MSE of 0.0511. There weren't any fundamental difference between the implementations so I believe the difference in MSE is a result of different hyperparameters and architectures. Achieving similar errors indicates that the implementations have similar learning capacities. Fig. 3.1 shows the true trajectories and predicted trajectories from the test set.

3.2 Training network on forces and energies

I trained several SE(3)-Transformer networks to predict both energies and forces. The networks use the positions and one-hot encoding of atomic species as inputs and outputs a scalar (type-0) and vector (type-1) for each node. The sum of the scalars is the total energy and the vectors are the net forces on each atom. I also trained some networks to predict just energies. The networks were trained on batches of about 100 molecules. The loss function was

$$L = \frac{1}{N}\sum_{i=1}^{n}\left[W_e\left(\hat{E}_i - E_i\right)^2 + \frac{1}{M_i}\sum_{j=1}^{M_i}\left(\hat{f}_{ij} - f_{ij}\right)^2\right], \tag{3.2}$$

similar to the loss function used in the ANI2 network [64]. The network was trained by the ADAM optimizer and I used a variety of learning rates.

I used two different dataset sets: The ANI1 and ANI1x datasets [8] [63]. The ANI1 dataset contains about 20 million conformations of small molecules and their energies. The ANI1x dataset contains about 5 million conformations of small molecules and their forces and energies.

I multiplied the energy losses by 10^{-5} due to the spread of energy values. My model has 8 heads, 3 layers, and 2 channels. I trained both models for 200,000 iterations. I began the training with a learning rate of 10^{-3}. After 86,000 iterations I dropped the learning rate to 10^{-5} and at 176,000 iterations I dropped the learning rate to 10^{-7}. I clipped the gradient norm of the parameters to stabilize the training. The results are shown in Fig. 3.2.

The force errors were comparable to the result of the the ANI-1 and ANI-1x networks but the energies were completely off. I later learned that self-interaction energies of many atomistic systems (the energy of the system if it were completely atomized) makes up over 99% of the total energy. I believe that the network was unable to effectively learn the self-interaction energy because there were only two channels which created a bottleneck. I also tried doing rudimentary molecular dynamics simulations of a water molecule to see if the forces were useful. The initial position of the water molecule was in its equilibrium position and with no velocity. After a few hundred time

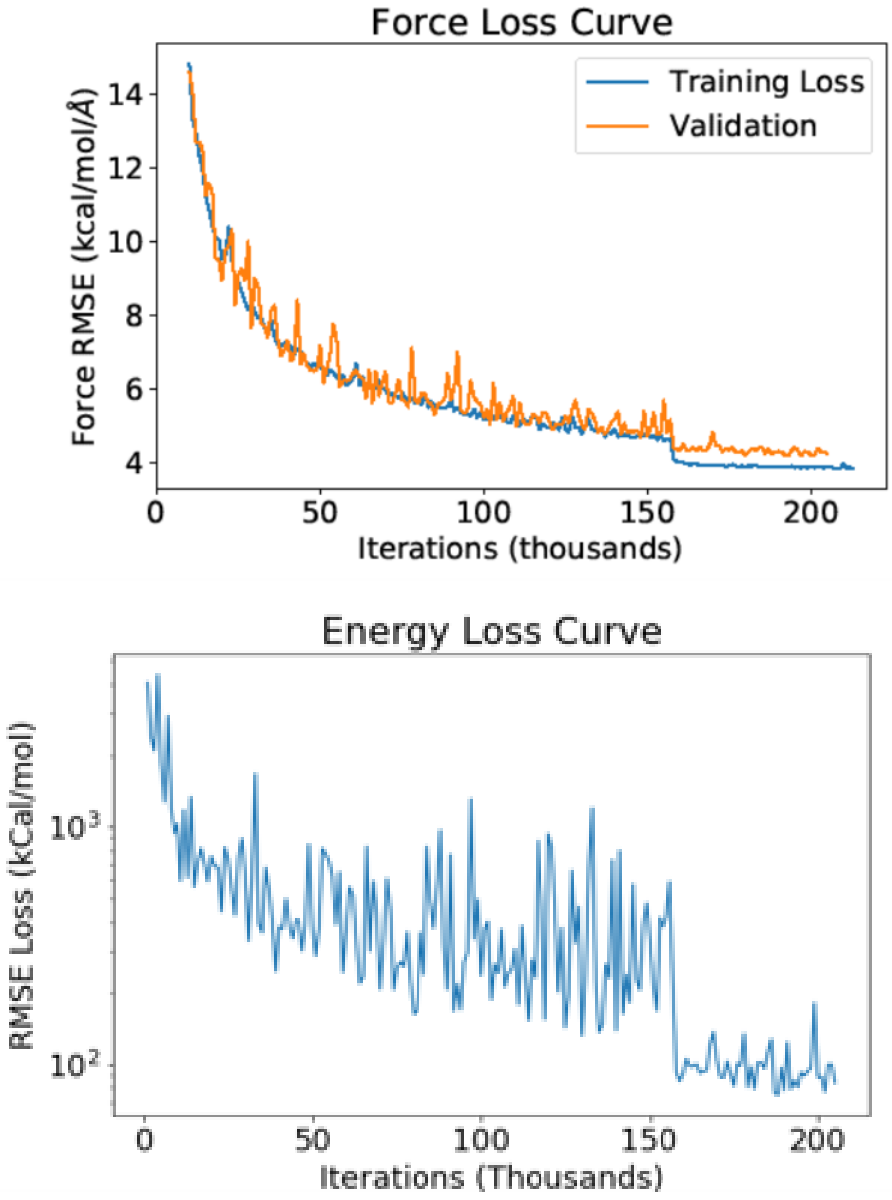

Figure 3.2 Plot of the learning curves for the original training. Force errors went down to about 4 kcal·mol^{-1}Å^{-1}, but energy errors went down to 100 kcal·mol^{-1}.

Dataset	Mean (Ha)	STD (Ha)
ANI1	-338	60.9
ANI1 w/o SI	0.0574	0.110
ANI1x	-395	139
ANI1x w/o SI	0.0184	0.106

Table 3.1 Comparison between the ANI1 and ANI1x datasets and adjusted versions of the dataset with the self-interaction energies subtracted. The majority of the energy is attributed to the self-interaction energy. Subtracting the self-interaction energies greatly decreases the standard deviation of the dataset, making it easier for the model to learn.

steps, the atoms were completely separated by dozens of Angstroms so the predicted forces didn't seem to be very useful.

I did a parameter sweep of the energy weights to determine which hyperparameter value would improve the performance. Suprisingly, force losses seemed much more affected by the energy weight than the energy losses. I hypothesize that is due to gradient clipping: energy caused the largest gradients so they were always clipped, so the magnitude of the energy losses wouldn't matter. The energy losses rapidly decreased when the learning rate was dropped. This also may be due to large parameters associated with energy. The losses curves are shown in Fig. 3.3.

All of the energy values root-mean-squared (RMS) values were several times larger than those achieved by the ANI networks. From a conversation with Olexandr Isayev, I learned that when training the ANI networks, they shifted energies by subtracting "self-interaction energy". The core electrons contribute much more to the total energy than chemical bonds, and this contribution is the self interaction energy. I found that subtracting the self-interaction energy greatly decreased the spread of the energies in the dataset, as shown in Table 3.1. In both cases, subtracting the self interaction energy brought the mean closer to zero and decreased the standard deviation by over 99%.

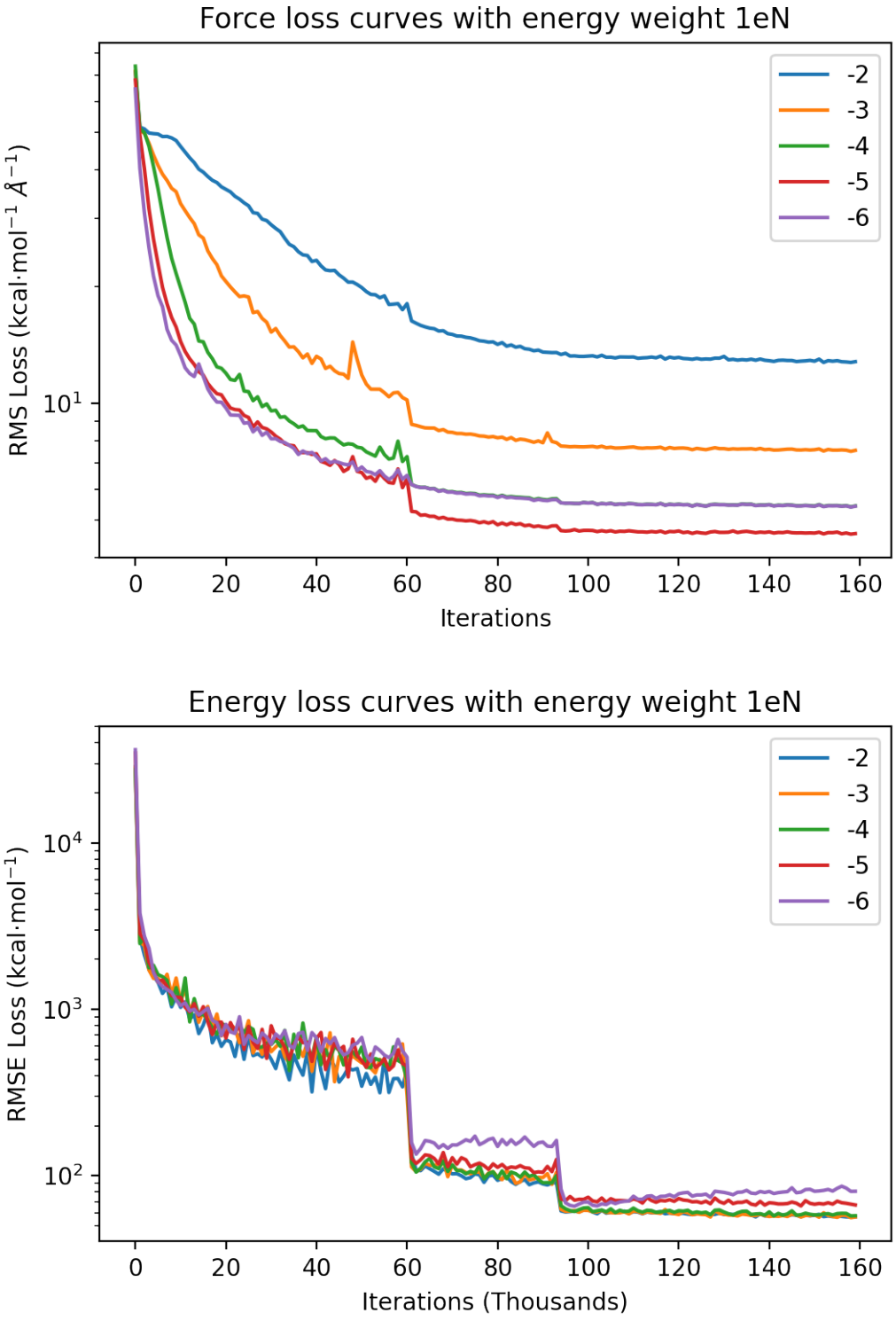

Figure 3.3 Plot of the learning curves for different energy weight values. I varied the energy weight from 10^{-2} to 10^{-6} by powers of 10. I started all trainings with a learning rate of 10^{-3}, lowered it to 10^{-5} at 62,000 iterations, and lowered it to 10^{-7} at 94,000.

Model	Energy error ($\text{kcal} \cdot \text{mol}^{-1}$)	Force error ($\text{kcal} \cdot \text{mol}^{-1} \cdot \text{Å}^{-1}$)
basic	3.211	4.104
wider	3.305	4.339
higher-order	5.459	6.294

Table 3.2 Comparison of the energy and force losses between the different models.

I subtracted the self-interaction energies from the datasets and normalized. I then used the adjusted datasets for new trainings. Since the energy and forces were normalized, I set the energy weight equal to 1. The training curves are shown in Fig. 3.4.

After training the SE(3)-Transformer networks for 100,000, the model with a learning rate of 10^{-3} had the lowest losses so I continued training that model for another 241,000 iterations for a total of 341,000 iterations. A plot of the energy and force losses are shown in Fig. 3.5.

I also trained a energy only model on the ANI-1 dataset. The training curve is shown in Fig. 3.6.

I trained two other models with 4 channels instead of 2 channels. Additionally, one of the models using up to type-3 vectors instead of just type-2 vectors. The second model had batch sizes about the third of the size of the first due to being higher-ordered. The results of those training is show in Fig. 3.7.

The overall results of the three models is shown in Table 3.2.

Now that models are trained, my next steps would be to compare how the losses in the training set compare to the validation set. This will allow me to select which model actually performs best on the test instead of memorizing the training set. Then I need to see how the best model performs on the randomly selected test set and the COMP6 test set [65]. Next, I would see how the models perform when used in molecular dynamics simulations and whether it can reproduce physical properties of various molecules.

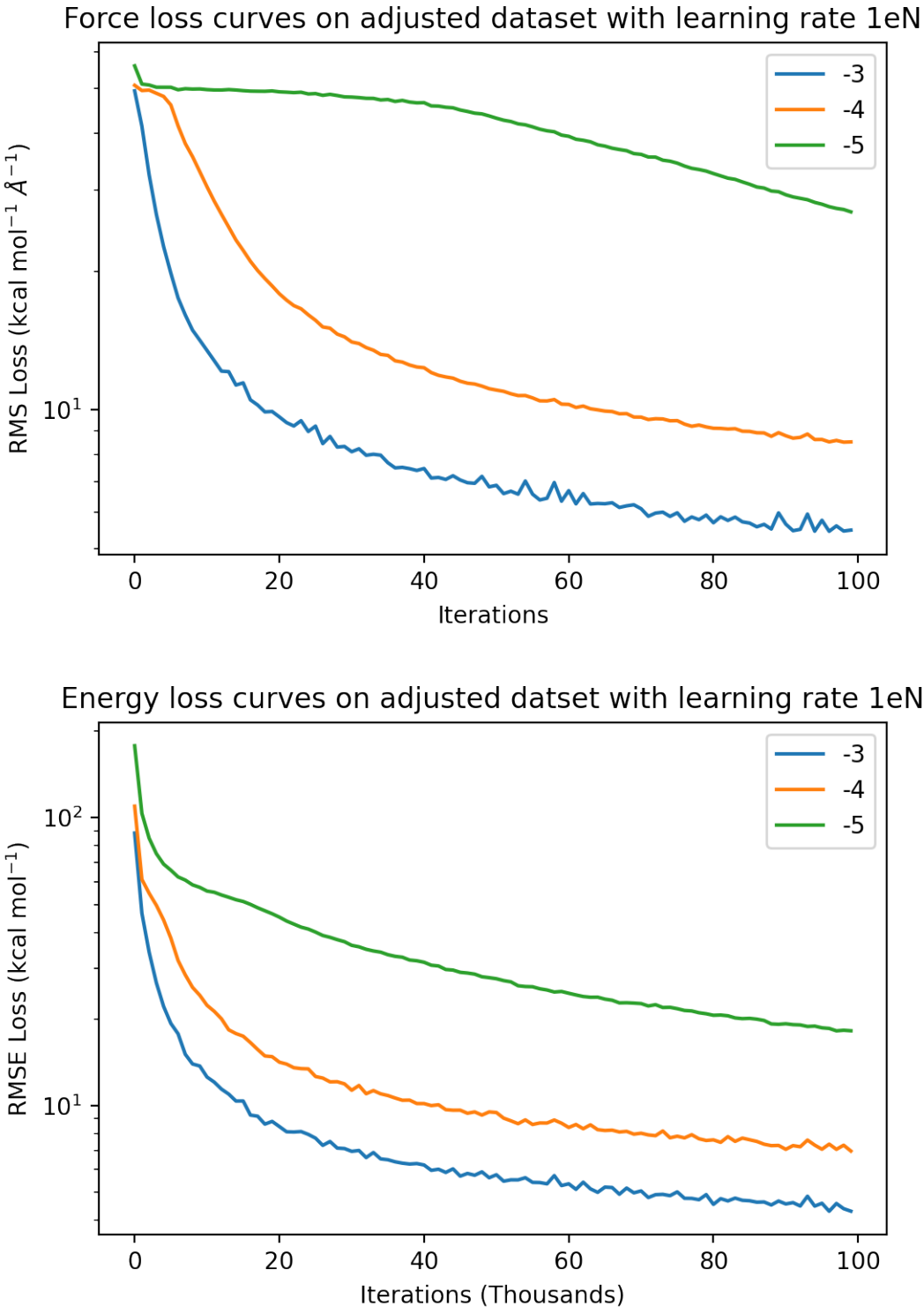

Figure 3.4 Plot of the learning curves for learning rates of 10^{-3}, 10^{-4}, 10^{-5}. The networks were trained using the rescaled and normalized ANI1x dataset for 100,000 iterations.

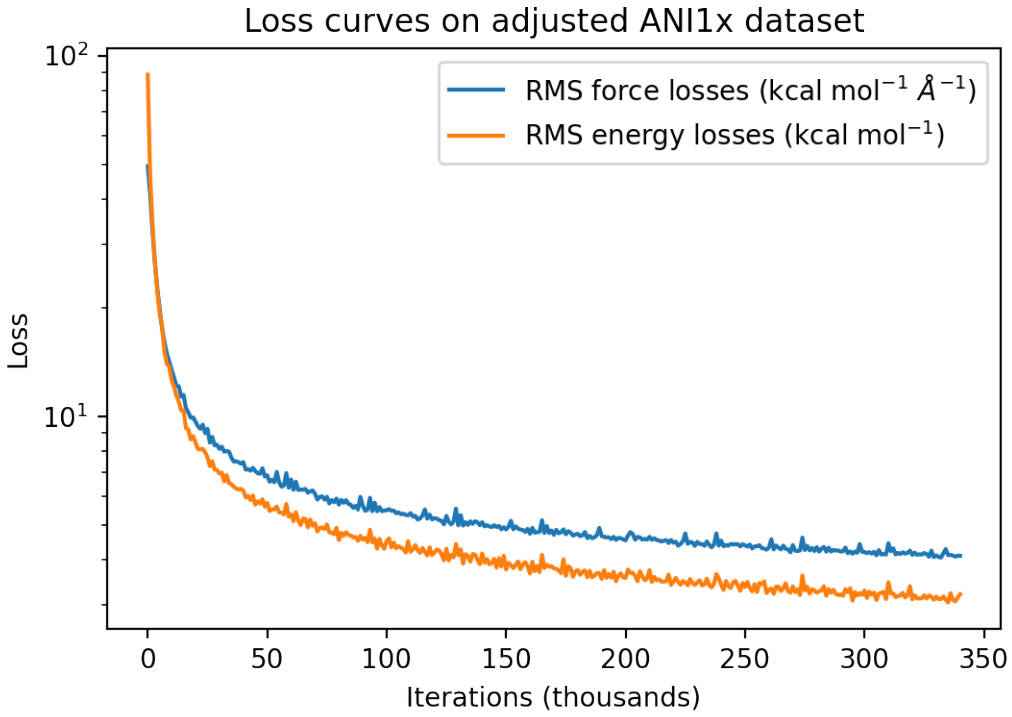

Figure 3.5 Plot of the learning curves for learning rates of 10^{-3}. The networks were trained using the rescaled and normalized ANI1x dataset for 341,000 iterations. The latest force loss was $4.104\text{kcal}\cdot\text{mol}^{-1}\cdot\text{Å}^{-1}$ and the latest energy loss was $3.211\text{kcal}\cdot\text{mol}^{-1}$.

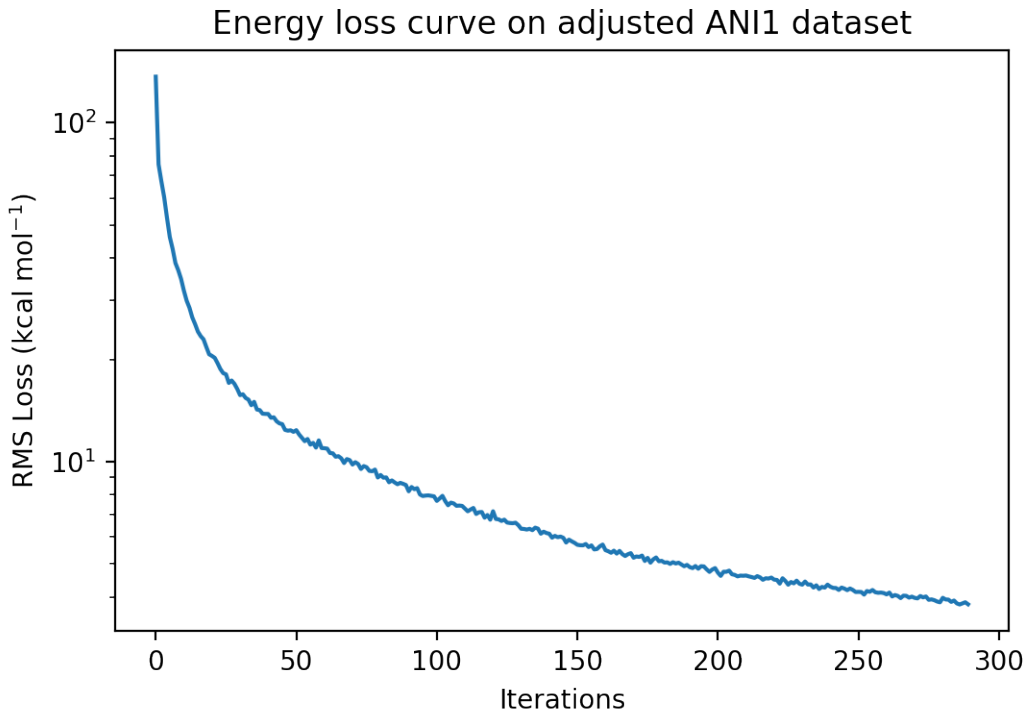

Figure 3.6 Plot of the learning curves a network trained on the rescaled and normalized ANI1 dataset. The latest energy loss was $3.802\text{kcal} \cdot \text{mol}^{-1}$

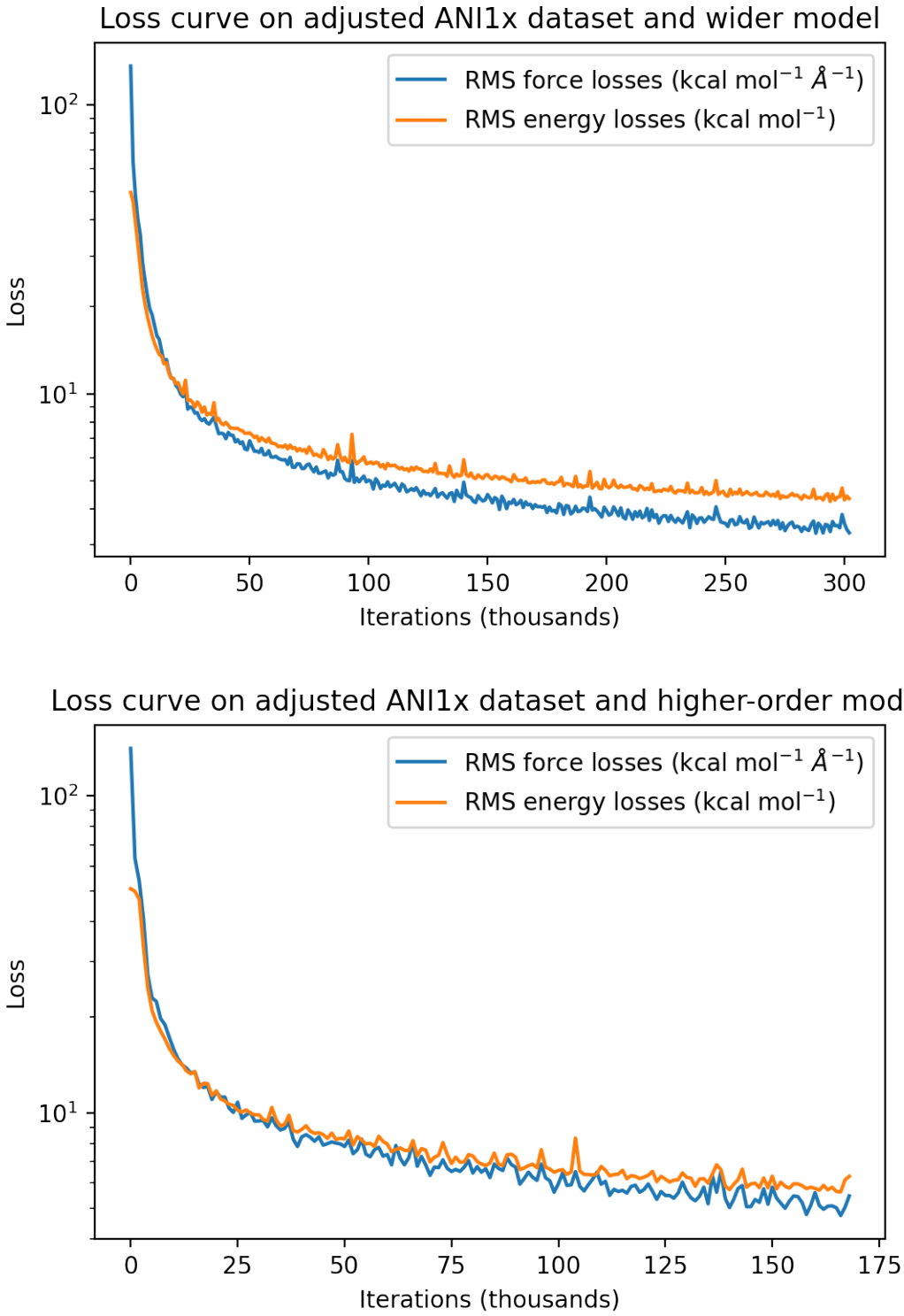

Figure 3.7 Plot of the learning curves networks trained on the rescaled and normalized ANI1x dataset. Both networks has 4 channels instead of 2 and the second one uses up to type-3 vectors. The latest errors on the first model were 3.305kcal $\cdot$ mol^{-1} and 4.339kcal $\cdot$ mol$^{-1}\cdot$Å^{-1}; the latest errors on the second model were 5.459kcal $\cdot$ mol^{-1} and 6.294kcal $\cdot$ mol$^{-1}\cdot$Å^{-1}. The second model never reached the same losses of the first model, possibly due to there being so many more parameters and smaller batch sizes.

3.3 Suggestions for further work

Further investigation is needed to see whether a similar network would work on a dataset consisting of more atom types. Additionally, the size of molecules in my dataset was limited. It is unknown how well the network can generalize to more complicated systems.

In my implementation of the network, I represent the five atom types using a one-hot encoding. That is, I represented each type of an atom with a vector of length 5 with zeros everywhere except for a 1 at an index that represents that atomic species. I would eventually like a network to be able to generalize to all atomic species. This would require the network to have a better representation of the atom types. This could be achieved by either having a fixed atomic embedding or switching the network's radial functions to those that are derived from DFT densities.

When the network has finished training, I'll integrate the network into molecular dynamics software. I'll then see if the network is capable of simulating chemical reactions.

3.4 Potential impact

The take away message is that I have successfully implemented a deep-learning model that will be able to predict inter-atomic forces. It will be able to predict forces much faster and hopefully more accurately than conventional force fields. The network will be eventually integrated into software so that it can be used in molecular dynamics simulations.

This is the first time that I am aware of an SE(3) equivariant neural network being trained on a large dataset of density functional theory data. SE(3)-equivariant neural networks have increased learning power over manually designed chemical encoding functions so it is likely that accuracy will increase as this field develops. It is possible that such network will be able to replace conventional force fields in molecular dynamics simulations and will be a valuable tool to investigate molecular properties.

Bibliography

[1] F. Müller-Plathe, "Coarse-graining in polymer simulation: from the atomistic to the mesoscopic scale and back," ChemPhysChem **3,** 754–769 (2002).

[2] F. Ercolessi and J. B. Adams, "Interatomic potentials from first-principles calculations: the force-matching method," EPL (Europhysics Letters) **26,** 583 (1994).

[3] C. G. Van de Walle and P. E. Bloechl, "First-principles calculations of hyperfine parameters," Physical Review B **47,** 4244 (1993).

[4] F. B. Fuchs, D. E. Worrall, V. Fischer, and M. Welling, "SE (3)-transformers: 3D roto-translation equivariant attention networks," arXiv preprint arXiv:2006.10503 (2020).

[5] F. B. Fuchs, E. Wagstaff, J. Dauparas, and I. Posner, "Iterative SE (3)-Transformers," arXiv preprint arXiv:2102.13419 (2021).

[6] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," Physical review **140,** A1133 (1965).

[7] M. J. Frisch *et al.*, "Gaussian~16 Revision C.01,", 2016, gaussian Inc. Wallingford CT.

[8] J. S. Smith, O. Isayev, and A. E. Roitberg, "ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost," Chem. Sci. **8,** 3192–3203 (2017).

[9] M. P. Allen *et al.*, "Introduction to molecular dynamics simulation," Computational soft matter: from synthetic polymers to proteins **23,** 1–28 (2004).

[10] M. Karplus and G. A. Petsko, "Molecular dynamics simulations in biology," Nature **347,** 631–639 (1990).

[11] J. M. Haile, I. Johnston, A. J. Mallinckrodt, and S. McKay, "Molecular dynamics simulation: elementary methods," Computers in Physics **7,** 625–625 (1993).

[12] P. Eastman and V. Pande, "OpenMM: A hardware-independent framework for molecular simulations," Computing in science & engineering **12,** 34–39 (2010).

[13] S. L. Mayo, B. D. Olafson, and W. A. Goddard, "DREIDING: a generic force field for molecular simulations," Journal of Physical chemistry **94,** 8897–8909 (1990).

[14] K. Vanommeslaeghe *et al.*, "CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields," Journal of computational chemistry **31,** 671–690 (2010).

[15] J. Wang, R. M. Wolf, J. W. Caldwell, P. A. Kollman, and D. A. Case, "Development and testing of a general amber force field," Journal of computational chemistry **25,** 1157–1174 (2004).

[16] W. Humphrey, A. Dalke, and K. Schulten, "VMD: visual molecular dynamics," Journal of molecular graphics **14,** 33–38 (1996).

[17] M. Tuckerman, B. J. Berne, and G. J. Martyna, "Reversible multiple time scale molecular dynamics," The Journal of chemical physics **97,** 1990–2001 (1992).

[18] B. J. Leimkuhler, S. Reich, and R. D. Skeel, "Integration methods for molecular dynamics," in *Mathematical Approaches to biomolecular structure and dynamics* (Springer, 1996), pp. 161–185.

[19] J. A. Izaguirre, D. P. Catarello, J. M. Wozniak, and R. D. Skeel, "Langevin stabilization of molecular dynamics," The Journal of chemical physics **114,** 2090–2098 (2001).

[20] E. Hairer, C. Lubich, and G. Wanner, "Geometric numerical integration illustrated by the Störmer-Verlet method," Acta numerica **12,** 399–450 (2003).

[21] R. G. Parr, "Density functional theory of atoms and molecules," in *Horizons of quantum chemistry* (Springer, 1980), pp. 5–15.

[22] J.-M. Combes, P. Duclos, and R. Seiler, "The born-oppenheimer approximation," in *Rigorous atomic and molecular physics* (Springer, 1981), pp. 185–213.

[23] W. Koch and M. C. Holthausen, *A chemist's guide to density functional theory* (John Wiley & Sons, 2015).

[24] N. Raychev, "Universal quantum operators," International Journal of Scientific and Engineering Research **6,** 1369–1371 (2015).

[25] L. C. Snyder and H. Basch, "Molecular wave functions and properties," (1972).

[26] F. M. Bickelhaupt and E. J. Baerends, "Kohn-Sham density functional theory: predicting and understanding chemistry," Reviews in computational chemistry **15,** 1–86 (2000).

[27] T. L. Gilbert, "Hohenberg-Kohn theorem for nonlocal external potentials," Physical Review B **12,** 2111 (1975).

[28] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning* (MIT press Cambridge, 2016), No. 2.

[29] M. Anthony and P. L. Bartlett, *Neural network learning: Theoretical foundations* (cambridge university press, 2009).

[30] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," arXiv preprint arXiv:1412.6830 (2014).

[31] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," arXiv preprint arXiv:1811.03378 (2018).

[32] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," arXiv preprint arXiv:1710.05941 (2017).

[33] A. F. Agarap, "Deep learning using rectified linear units (relu)," arXiv preprint arXiv:1803.08375 (2018).

[34] R. Sun, "Optimization for deep learning: theory and algorithms," arXiv preprint arXiv:1912.08957 (2019).

[35] J. Leonard and M. Kramer, "Improvement of the backpropagation algorithm for training neural networks," Computers & Chemical Engineering **14,** 337–341 (1990).

[36] H. Leung and S. Haykin, "The complex backpropagation algorithm," IEEE Transactions on signal processing **39,** 2101–2104 (1991).

[37] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010* (Springer, 2010), pp. 177–186.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980 (2014).

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," arXiv preprint arXiv:1706.03762 (2017).

[40] D. B. West *et al.*, *Introduction to graph theory* (Prentice hall Upper Saddle River, 2001), Vol. 2.

[41] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE transactions on neural networks **20,** 61–80 (2008).

[42] M. Wang *et al.*, "Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs.," (2019).

[43] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," IEEE transactions on neural networks **8,** 98–113 (1997).

[44] W. R. Scott, *Group theory* (Courier Corporation, 2012).

[45] J. Ping, F. Wang, and J.-q. Chen, *Group representation theory for physicists* (World Scientific Publishing Company, 2002).

[46] E. Williams, *Representation theory* (MIT Press, 2002).

[47] G. Strang, G. Strang, G. Strang, and G. Strang, *Introduction to linear algebra* (Wellesley-Cambridge Press Wellesley, MA, 1993), Vol. 3.

[48] V. S. Varadarajan, *Lie groups, Lie algebras, and their representations* (Springer Science & Business Media, 2013), Vol. 102.

[49] A. Baker, *Matrix groups: An introduction to Lie group theory* (Springer Science & Business Media, 2012).

[50] M. L. Boas, *Mathematical methods in the physical sciences* (John Wiley & Sons, 2006).

[51] C. Müller, *Spherical harmonics* (Springer, 2006), Vol. 17.

[52] D. J. Griffiths and D. F. Schroeter, *Introduction to quantum mechanics* (Cambridge University Press, 2018).

[53] T. Mueller, A. Hernandez, and C. Wang, "Machine learning for interatomic potential models," The Journal of chemical physics **152,** 050902 (2020).

[54] V. L. Deringer, M. A. Caro, and G. Csányi, "Machine learning interatomic potentials as emerging tools for materials science," Advanced Materials **31,** 1902765 (2019).

[55] V. Ladygin, P. Y. Korotaev, A. Yanilkin, and A. Shapeev, "Lattice dynamics simulation using machine learning interatomic potentials," Computational Materials Science **172,** 109333 (2020).

[56] H. Chan, B. Narayanan, M. J. Cherukara, F. G. Sen, K. Sasikumar, S. K. Gray, M. K. Chan, and S. K. Sankaranarayanan, "Machine learning classical interatomic potentials for molecular dynamics from first-principles training data," The Journal of Physical Chemistry C **123,** 6941–6957 (2019).

[57] G. C. Sosso, G. Miceli, S. Caravati, J. Behler, and M. Bernasconi, "Neural network interatomic potential for the phase change material GeTe," Physical Review B **85,** 174103 (2012).

[58] S. Bukkapatnam, M. Malshe, P. Agrawal, L. Raff, and R. Komanduri, "Parametrization of interatomic potential functions using a genetic algorithm accelerated with a neural network," Physical Review B **74,** 224102 (2006).

[59] K. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, "Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions," Nature communications **10,** 1–10 (2019).

[60] Y. Zuo *et al.*, "Performance and cost assessment of machine learning interatomic potentials," The Journal of Physical Chemistry A **124,** 731–745 (2020).

[61] C. W. Park, M. Kornbluth, J. Vandermause, C. Wolverton, B. Kozinsky, and J. P. Mailoa, "Accurate and scalable multi-element graph neural network force field and molecular dynamics with direct force architecture," arXiv preprint arXiv:2007.14444 (2020).

[62] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, "Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds," arXiv preprint arXiv:1802.08219 (2018).

[63] J. S. Smith, R. Zubatyuk, B. Nebgen, N. Lubbers, K. Barros, A. E. Roitberg, O. Isayev, and S. Tretiak, "The ANI-1ccx and ANI-1x data sets, coupled-cluster and density functional theory properties for molecules," Scientific data **7,** 1–10 (2020).

[64] C. Devereux, J. S. Smith, K. K. Davis, K. Barros, R. Zubatyuk, O. Isayev, and A. E. Roitberg, "Extending the Applicability of the ANI Deep Learning Molecular Potential to Sulfur and Halogens," Journal of Chemical Theory and Computation **16,** 4192–4202 (2020).

[65] J. S. Smith, B. Nebgen, N. Lubbers, O. Isayev, and A. E. Roitberg, "Less is more: Sampling chemical space with active learning," The Journal of chemical physics **148,** 241733 (2018).