Characterizing the Potential Energy Surface of Two Dimensional and Bulk Materials using High Dimensional Neural Network Potentials

by

Amber Maharaj

A thesis submitted to the School of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of

Master of Science in Modelling and Computational Science

Faculty of Science
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
August 2018

Declaration of Authorship

I, Amber MAHARAJ, declare that this thesis titled, "Characterizing the Potential Energy Surface of Two Dimensional and Bulk Materials using High Dimensional Neural Network Potentials" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: August 20th, 2018

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Science
Department of Physics

Master of Science in Modelling and Computational Science

Characterizing the Potential Energy Surface of Two Dimensional and Bulk Materials using High Dimensional Neural Network Potentials

by Amber MAHARAJ

Computing material properties at the *ab-initio* level of detail is computationally prohibitive for large systems or long timescales. As a result, such methods cannot be used to efficiently sample configuration space. Force field methods can efficiently sample configuration space, but rely on large parameter sets that are tuned to specific contexts.

In this work we will explore the ænet approach and its application to six systems: 2D silica, bulk silica, graphene, diamond, hexagonal boron nitride, and cubic boron nitride. Here, a general mapping from atomic coordinates to the potential energy surface is obtained using a feed-forward artificial neural network. An approximate Density Functional Theory method, Density Functional Tight Binding (DFTB+), is used to compute quantities required for the reference dataset. It is found that a network made up of linear activation functions in ænet is (almost) equivalent to a one-layer radial basis function network, and is sufficient to learn a reference dataset consisting of structures sampled from a canonical ensemble at various temperatures. We look at how sampling outside of these frequently visited energy states, through data augmentation, significantly increases the complexity of the problem.

Acknowledgements

First, I would like to thank my supervisor, Isaac Tamblyn. His mentorship, guidance and compassion over the course of my studies has been invaluable. He has made so many opportunities available to me during my research, and I will always be grateful. Next I would like to thank Hendrick de Haan for his guidance and input during my thesis preparations, as a member of my supervisory committee. I would like to extend a thank you to Ken Pu for agreeing to be my external examiner. I am sincerely grateful to have had the opportunity to work alongside the members of CLEAN lab. It has been incredibly rewarding to be a member of this research group. I would specifically like to thank Elizabeth Selinger and Kyle Mills for all their help and support.

My family has been amazingly supportive throughout my Master's degree, and I would like to thank them for their support and encouragement. Last but certainly not least, I would like to thank my significant other Martin Magill. I am extremely grateful to have gone through this experience with him by my side. He has helped me through some very difficult times, and has provided unwavering support. Our discussions have helped me flesh out a lot of ideas and stay organized.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Given the expensive and time consuming nature of experiments, materials simulations are an invaluable tool for of guiding chemical experiments and/or interpreting their results. Material simulations can provide insight into what is possible to achieve experimentally. A major barrier to gathering a substantial amount of data is the large computational cost of highly accurate simulation methods. The Schrödinger equation cannot be solved exactly for most relevant systems. For this reason, approximate methods such as DFT were developed. Such electronic structure methods, can generate reliable potentials and are very costly to compute. The most popular implementation is density functional theory (DFT), which requires solving the many body Schrödinger equation:

$$H\Psi = \left[\sum_{i}^{N}\left(-\frac{\hbar^2}{2m_i}\nabla_i^2\right) + \sum_{i}^{N} V(\vec{r}_i) + \sum_{i<j}^{N} U(\vec{r}_i,\vec{r}_j)\right]\Psi \tag{1.1}$$

This task has an algorithmic scaling of $O(n^3)$ where n is the number of electrons in the system. This restricts simulations to system sizes of a few hundred atoms and timescales to hundreds of picoseconds.

There is frequently a trade-off between accuracy and scalability (as seen in figure 1.1) when studying a system using computational methods. Approximations can be made to *ab-initio* methods whereby matrix elements of the Hamiltonian are pre-tabulated. These are called semi-empirical potentials, and they can improve the computational complexity by 1-2 orders of magnitude [1]. An

example of this method is Density Functional Tight Binding (DFTB+), which is an approximate DFT method.

At a low level of accuracy, there are classical force field methods. These methods are highly efficient (fast and scalable), but require a large number of parameters to fit classical potential energy surfaces. The parameters used to fit these force fields are tuned to fit specific experimental or *ab-initio* situations. While force field methods can extend the timescales of and sizes of systems with thousands of atoms (compared to hundreds), they may not capture details of systems that enter situations outside of those for which the potential was parameterized.

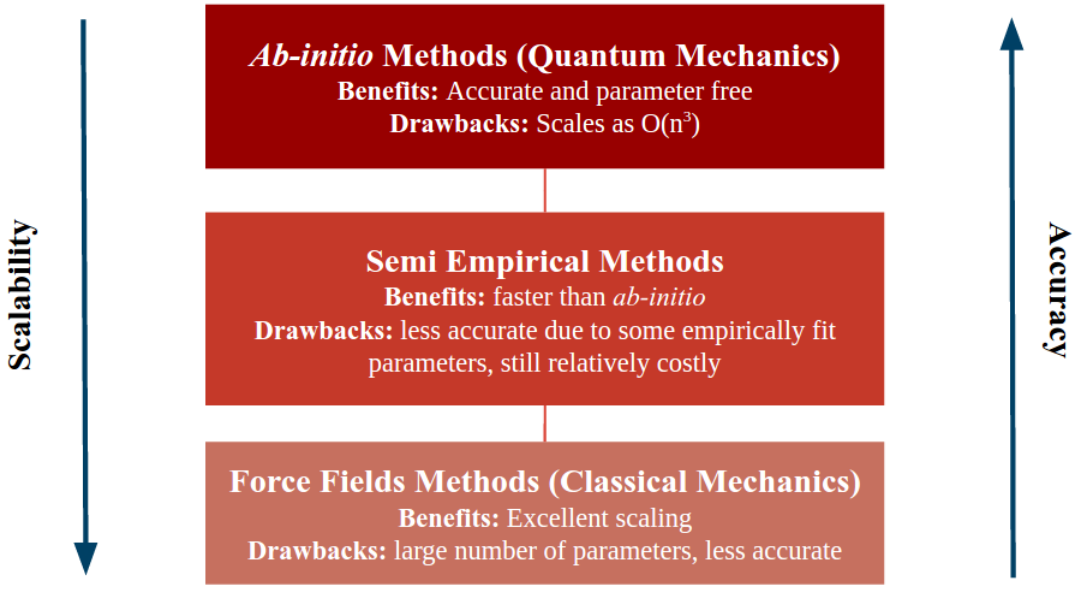

FIGURE 1.1: Levels of theory for atomistic simulations.

Molecular Dynamics (MD) is a popular algorithm used in materials simulation. In an MD simulation, structural information is read as an input. The functional form is determined by the level of theory that is being used in the simulation. At this point one must choose a method for calculating the potential. An *ab-initio* or semi-empirical method will involve solving the Schrödinger equation, whereas a forcefield method will involve solving a classical potential with a set of parameters that describe the chemical environment. Once the potential is obtained, the force can be calculated by taking the derivative of the

potential with respect to the atomic coordinates. Since $a = \frac{F}{m}$, the atomic coordinates can then be updated using the equations of motion. The time step is increased, and this process is repeated iteratively until specified to stop. A schematic of this algorithm can be seen in figure 1.2. The bottleneck in algorithmic complexity for MD simulations corresponds to the evaluation of this interaction potential.

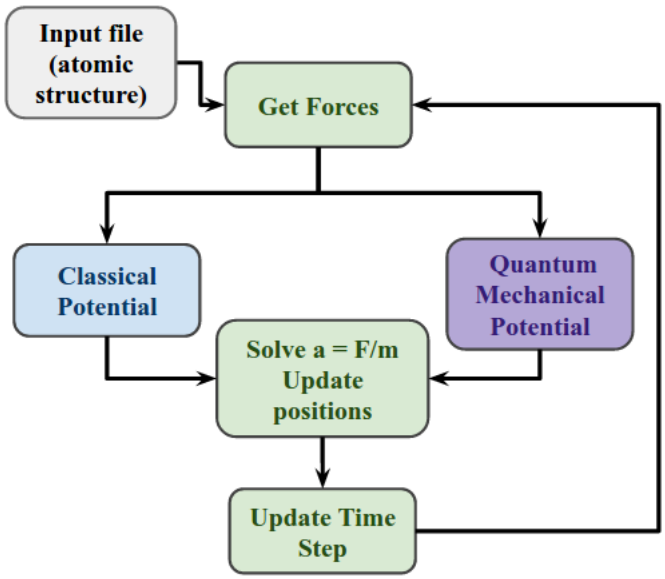

FIGURE 1.2: Molecular Dynamics algorithm.

The relationship between nuclear coordinates and potential energy is, in most cases, too difficult to determine analytically [11]. Therefore, physically inspired models can fall short of providing a complete description of the system. Most classical force fields are not able to accurately model bond breaking and formation for example. Many classical force fields also fail to model the effects of atomic environment on bonding properties for metals and alloys [11], since these force fields explicitly include the bond strength as part of the functional form. To work around such challenges, separate force fields have been designed to model chemical reactions. These force fields are bond order based [77, 82] allowing for continuous bond formation or breaking. Typically, *ab-initio* level detail would be needed to describe bond breaking and formation. While these efforts achieve some bridging between quantum mechanical and classical methods, they still rely heavily on a physically inspired functional form. The

model might predict some features well, but not others. For instance, in a recent paper (2018), Manzano et al [57] showed that the reaxFF force field could reproduce hydrogen bonding, microstructure, proton subtransfer and diffusion of super- and sub- critical water with a very good quantitative agreement to experimental data. However, for high temperatures, the static dielectric constant was overestimated. While the model captures many things well, it does not capture everything. In some cases, for very complex potential energy surfaces, the physical potentials are difficult or even impossible to derive [11].

It is desirable to design an interatomic potential where the mapping from nuclear coordinates to potential energy surface is general. This can be achieved through using neural networks that train on electronic structure data. In this work, an implementation of the Behler and Parinello approach [10] is used to learn the interatomic potentials for various systems of interest. In this approach, atomic coordinates are projected onto a set of symmetry functions that model the local atomic environments within the system. The symmetry functions are the input for a neural network made up of subnets, that each output an atomic energy contribution. The atomic energy contributions are then summed in the output layer to provide the total energy for the system. This Neural Network Potential (NNP), represents a nested function that can be evaluated at a fraction of the cost of an *ab-initio* method, with a comparable accuracy. Also, by defining the total energy as a sum of atomic energy contributions, atoms can be added to or removed from the system while using the potential. This allows for increased scalability.

1.2 Review of Literature

Various methods have been used to construct the potential energy surface of materials through interpolation of ab-initio calculations [39, 18, 7, 15, 6, 10]. Such methods do not rely on a physically inspired functional form. Instead the potential energy surface is constructed in an unbiased and automated way. Here we will breifly review various methods for interpolating the potential energy surface. Following this, we will explore the neural network potential approach developed by Behler and Parinello [10], and applications of this method using open source software ænet [3]. After reviewing interpolation methods, the materials explored in this work will be introduced.

1.2.1 Interpolating the Potential Energy Surface

In general, computing the entire potential energy surface for materials is computationally intractable. While classical methods are able to explore more of the potential energy surface, they have a lower accuracy than is sometimes required. *ab-initio* methods explore much less of the potential energy surface during molecular dynamics, but they have the highest level of accuracy for computing material properties. By interpolating a set of reference *ab-initio* level data, the potential energy surface does not need to be completely evaluated. A naive approach to interpolating the potential energy surface is to sample the potential energy surface on a uniform grid and compute the solution at an *ab-initio* level of theory. This would be very expensive to implement. Also, the most frequently visited configurations occur near the minima. Thus, a key step in interpolating *ab-initio* points, is sampling the potential energy surface around regions of interest.

In one approach, a modified Shepard Interpolation method was used to construct the PES. This was implemented in the context of describing reaction dynamics. In this method, a classical molecular dynamics simulation is used to explore configuration space and identify dynamically important regions of configuration space. The potential energy surface is given by an interpolation of local Taylor expansions that are centered around a configuration whose potential is computed at ab-initio level detail. Instead of interpolating over a regular grid of points on configuration space, the classical MD is used to identify dynamically important regions that are sampled more densely. The result is that the sampling of configuration space is non-uniform, and allows for interpolation over a fewer number of data points.

While earlier methods employed this algorithm for only low-dimensional systems [39], more recent methods have extended this methodology to deal with systems with higher degrees of freedom [18]. Crespos et al [18], discuss how the PES can be iteratively developed by using the a preliminary interpolated potential in a classical dynamics simulation to explore dynamically important regions of configuration space. From these trajectories, more points of configuration space can be sampled and used to iteratively refine the PES. They discuss how a high degree of accuracy is important in regions that are frequently visited by classical MD trajectories. However, the accuracy of the

interpolated potential can also be improved by adding points in configuration space that are less likely to be visited during MD. The accuracy of the potential is considered to be converged when a computed observable is converged within a specified tolerance. Convergence is consistently shown to improve with number of data points [39, 18].

Another method of interpolating *ab-initio* data is the moving least squares interpolation [40, 55, 22, 21]. Using this method, the potential energy surface, gradients and Hessian data are interpolated around an *ab-initio* data point. Here the interpolated potential energy surface around an *ab-initio* data point is expressed as a set of linearly independent basis functions. To determine the coefficients for the linear combination of basis functions, the weighted deviation between the combination of basis functions and the true *ab-initio* potential is minimized. This weighted deviation is computed for each reference point and then summed. Here the weights are drawn from a weight function that decays as the distance from the *ab-initio* data point increases. Once the basis function coefficients are determined, the fitted potential can be determined by solving the normal equation.

The idea to use neural networks to describe interatomic potentials dates back to the early 90s [81, 67, 15, 73]. In early works, neural networks were generally restricted to generating the potential energy surfaces for small molecules or low dimensional systems [15, 73, 58]. Some issues complicate the application of NN to map coordinates to high dimensional PES. For one, the weights of a neural network are generally set up to be different from one another. Thus, interchanging the coordinates of two atoms of the same type can lead to a different total energy. Also, if the inputs to the neural network depend on something like the number of degrees of freedom in the system (as implemented by Lorenz et al [53]), the neural network will not generalize. The network is relevant only for a fixed number of input nodes.

In 2007, Behler and Parinello [10] introduced a method where the input coordinates are mapped onto a set of symmetry functions that are made up of radial and angular basis functions. These symmetry functions describe a local atomic environment (defined within a cutoff radius) that is invariant to translations, rotations and permutations of atoms. To deal with the high-dimensional potential energy surfaces, Behler and Parinello proposed a scheme where the total energy

of the system was expressed as a sum of the atomic energy contributions:

$$E = \sum_i E_i(\widetilde{\sigma}_i) \tag{1.2}$$

Where each atomic energy is a function of a set of symmetry functions $\widetilde{\sigma}$ which describe the local atomic environments.

Other groups have avoided selecting parameters to define the shape of the basis functions used in the Behler and Parinello approach. Bartòk et al [7] discusses how breaking down the total energy into atomic energy contributions (as suggested by Behler et al [9]), allows for a better approximation of the potential energy surface as atoms are added or removed from the region of interest. The basis function description is avoided by expanding the local atomic environment in a series of spherical harmonics. The descriptors for the atomic environments are created by constructing a neighbour density at each point in space for each atom up to a specified cutoff distance. The neighbouring atoms are multiplied by a set of weights used to discriminate which atomic species is at these positions, and a cutoff function to ensure a smooth decay at the cutoff radius. To get the angular distribution for the neighbours, all neighbour densities are projected onto a sphere that is then expanded in spherical harmonics. The radial distributions are converted to an additional angle extending the spherical harmonics to four dimensions [12]. These positions on the four dimensional sphere are expanded in hyperspherical harmonics. The coefficients of these harmonics produce a bispectrum matrix. The bispectrum matrix is then mapped to atomic energy contributions as implemented by Behler and Parinello.

Since the neighbour densities uses δ-functions centered around each atom in the atomic environment in the bispectrum neighbour density method, there can be instabilities when the positions deviate between two atomic environments. To prevent this issue, in 2013, Bartòk et al [6] released the Smooth Overlap of Atomic Orbitals (SOAP) method, that models the neighbour density with gaussians centered on all atoms in the local environment. These gaussians are similar to the radial basis functions used in Behler's work [10]. While many radial basis functions are mapped to a single atomic energy in the Behler approach, in the

SOAP method, only one neighbour density is needed to map to the atomic energy. The kernal[1] is defined by the overlap of two atomic environments. Gaussian processes are used to predict atomic energies of new configurations given previous observations.

In 2016, an open source implementation of the Behler-Parinello method was released called ænet (this implementation is the one that will be used to train network potentials throughout this work). At this point, the number of basis functions scaled with the number of species in the system. Consequently, only up to four atomic species were used while implementing the Behler-Parrinello neural network potentials [2]. The algorithm was recently updated to avoid scaling with the number of atomic species.

In 2017 the implementation was extended to systems of many body compositions. Artrith et al [5], showed that the same model complexity required for a ternary material was also sufficient to describe a material composed of 11 atomic species. In this approach, the local structure and composition were used to separately encode atomic positions and species using two invariant sets of coordinates. To get a combined descriptor, they take the union of each set of coordinates (belonging to structure and composition). The expansion for the structural descriptor corresponds to the bond length for the radial basis functions, and bond angle for angular basis functions. The expansion coefficients for the compositional descriptor have the same coefficients as the structural descriptor, but they are weighted differently based on the atomic species.

In 2018, Artith et al [4] addressed the difficulties involved in sampling configuration space to generate reliable potentials. To build a reliable potential, an extensive database of ab-initio calculations must be available which contains all important interactions. Sampling using ab-intio methods limits system sizes and how much of configuration space can be sampled. Traditionally, classical MD trajectories might be implemented to sample configuration space, and then an ab-initio level calculation would be taken on a subset of those configurations. Such an approach might require a dataset ranging from thousands to tens of thousands of reference data points depending on number of atomic species [4]. In this approach, the phase diagram of amorphous Li_xSi is explored using a genetic algorithm. The sampling is constrained to be near ground state structures. Using this *Specialized* potential (using genetic algorithm for sampling) only 1000

[1] A kernal is a measure of similarity between points.

reference data points were required to be competitive with the *general* potential (sampling configuration space by using classical MD) using approximately 45 000 reference data points.

1.2.2 Materials

In this section we will review the materials explored in this work. 6 materials will be used in molecular dynamics simulations to produce reference data for learning neural network potentials. Three of the materials are two-dimensional: 2D silica, graphene and hexagonal boron nitride. 2D materials are materials that are 1-2 atoms in thickness. Each material will be compared to a polymorph[2] (or allotrope[3]) of the same chemical composition. Graphene is compared to diamond, 2D silica is compared to bulk silica, and hexagonal boron nitride is compared to cubic boron nitride.

2D Silica and Bulk Silica

2D Silica, (also referred to as Hexagonal Bilayer Silica (HBS)) is part of a class of 2 dimensional materials that has gained a large amount of interest in the past 10 years [56]. HBS is the thinnest gate dielectric oxide layer and support in catalysis, and it has also been proven to be useful in separating graphene from a metal substrate [13]. Given the interest in 2D materials and their unique physical properties, 2D silica is a well studied 2D material along with graphene. While 2D silica appears to be three atoms thick in figure 4.1, the thickness is measured by the distance between the two silicon atoms.

[2] A polymorph refers to the property of a solid material of more than one element to exist in more than one form.

[3] an allotrope is a property of an element to exist in two or more different forms in the same physical state.

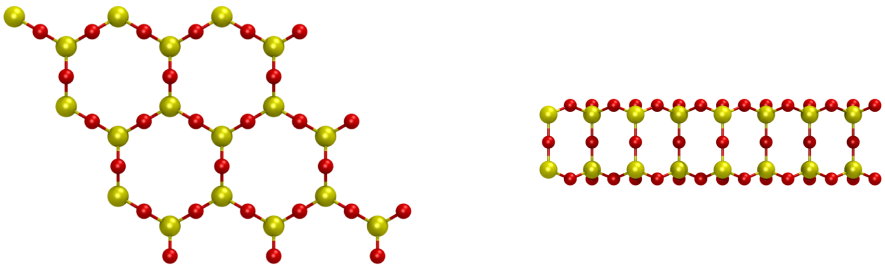

FIGURE 1.3: **Left:** View of the 2D silica surface. **Right:** View from the side of 2D silica.

Bulk SiO_2 is a common gate dielectric that has been used as a substrate for materials such as graphene and MoS_2 [41, 86]. SiO_2 Is frequently used in the production of metal-oxide-semiconductor field effect transistors (MOSFETS) due to properties such as having low interfacial defect density with Silicon, high resistivity, large band gap and high dielectric strength [26].

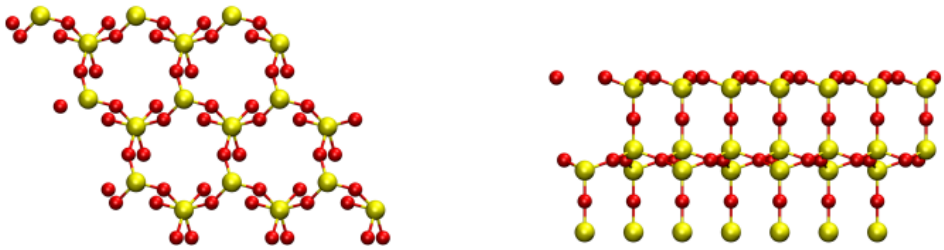

FIGURE 1.4: **Left:** Bulk silica surface, view from the top. **Right:** Bulk silica surface view from the side.

Graphene and Diamond

We will look at two allotropes of carbon: graphene and diamond. Graphene is a freestanding monolayer material arranged in a hexagonal lattice (figure 1.5). It is equivalently one isolated layer of graphite (a material made up of graphene sheets). There are many properties of graphene that make it a material of interest to study. For instance, it is extremely strong and lightweight. In fact, it is the strongest material that has ever been tested (tensile strength of $\sigma = 130$GPa)

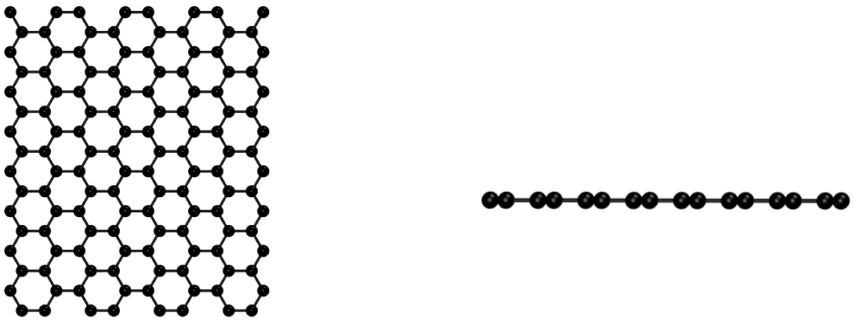

FIGURE 1.5: **Left:** Graphene view from top. **Right:** Graphene view from side.

[51]. It is also a very good conductor of heat and electricity with an electron mobility of up to $\mu_e = 15000\ \mathrm{cm^2V^{-1}s^{-1}}$ at room temperature [84], and a thermal conductivity ranging between $\kappa = 1500 - 2500\ \mathrm{Wm^{-1}K^{-1}}$ [16, 52, 27].

Diamond is a 3D crystal of carbon atoms. Diamond has an extremely rigid lattice with strong covalent bonding, and is known to be the hardest naturally forming material. It has good thermal conductivity ranging from $\kappa = 900 - 2320\ \mathrm{Wm^{-1}K^{-1}}$ [85]. Due to its hardness, it can be used for cutting tools eg. diamond-tipped drill bits and saws. Given its high thermal conductivity, it is also an ideal heat sink for electronics [69].

FIGURE 1.6: **Left:** Diamond view from top. **Right:** Diamond view from side.

Hexagonal Boron Nitride and Cubic Boron Nitride

Hexagonal boron nitride (hBN) is a wide bandgap material that has a similar structure to graphite, where hexagonal sheets are held together by weak Van

der Waals forces. The hexagonal form of boron nitride is the most stable form of the polymorphs. The monolayer hBN could be used be used as a complimentary 2D dielectric substrate for graphene electronics [79]. Cubic Boron Nitride

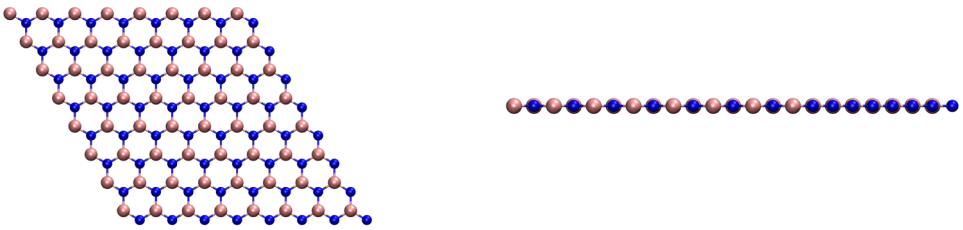

FIGURE 1.7: **Left:** Hexagonal boron nitride view from the top. **Right:** Hexagonal boron nitride view from the side.

(cBN) is a polymorph of boron nitride that is analogous to diamond. It is the second hardest material, after diamond [62]. While it is softer than diamond, it has better thermal and chemical stability making it an attractive alternative for various applications. cBN has use for abrasives, cutting tools [83].

FIGURE 1.8: **Left:** Cubic boron nitride, view from the top. **Right:** Cubic boron nitride, view from the side.

Experiments

2D materials are interesting for their unique physical properties and role in various applications eg. semiconductors and electrodes. In this work, we will characterize the potential energy surface of each of the 2D materials introduced above, and the corresponding bulk materials through temperature sampling and data augmentation.

We will see that the neural network converges to a lower error on some materials compared to others. We will then discuss how different potential energy surface complexity can impact how easily a neural network can learn.

Chapter 2

Background

The quality of a Neural Network Potential (NNP) is limited by the quality of the reference data set. The accuracy is limited by the level of theory at which the reference data is generated. A primary motivation for using NNPs is that it is difficult to generate large amounts of data at a high level of theory. However, this also serves a barrier for creating reference data. The most accurate reference data is generated using *ab-initio* methods, which are costly for large systems or extended timescales. One approach to generating data is to sample various simulation trajectories at uncorrelated times, thereby sampling configuration space. Since, it is very costly to generate a sufficiently long simulation trajectory, it makes sense to sample up by generating a large number of configurations using a cheaper simulation method and run a single higher level computation on the configurations to get the material properties.
This chapter will discuss the theory behind these proposed simulation methods from the perspective of specific simulation programs. To generate reference data, an approximate Density Functional Theory (DFT) method was used called Density Functional Tight Binding (DFTB). As much of the theory behind DFTB is built on DFT, the theory behind DFT will be discussed. Following this review will be an introduction to machine learning methods and the application of the Behler-Parinello approach to material simulations using atomic simulation environment (ænet).

2.1 Atomistic Simulations

2.1.1 Density Functional Theory

Density Functional Theory (DFT) is a method derived from quantum mechanics used in computational materials physics. It belongs to the class of *ab-initio* or first principles methods, meaning that it is free of empirical parameters. DFT is used primarily to determine the electronic structure[1] of materials in the ground state (lowest energy state). By solving the electronic structure of a given system over a range of nuclear configurations, the potential energy surface is obtained. In DFT, the Schrödinger equation is solved by finding the functional of the electron density. Once the wave function is obtained, all other quantum observables can easily be computed. In the following, I will discuss the derivation of DFT as it is relevant to DFTB.

We will start from the non-relativistic time-independent many-body Schrödinger equation:

$$\hat{H}\Psi(\{r_{i=1..N}, R_{I=1..M}\}) = E\Psi(\{r_{i=1..N}, R_{I=1..M}\}) \tag{2.1}$$

where $\hat{H}$ is the Hamiltonian, E is the total energy, and Ψ is the wave function which depends on the positions and spins of electrons, r_i, and the positions and spins of nuclei, R_I. N refers to the number of electrons and M refers to the number of nuclei.

Using Hartree atomic units ($\hbar = m_e = e = \frac{4\pi}{\epsilon_o} = 1$), the Hamiltonian term can be expressed as:

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \frac{1}{2}\sum_{I=1}^{M}\frac{1}{M_I}\nabla_I^2 - \sum_{i=1}^{N}\sum_{I=1}^{M}\frac{Z_M}{r_{iI}} + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} + \sum_{I=1}^{M}\sum_{K>I}^{M}\frac{Z_I Z_K}{R_{IK}} \tag{2.2}$$

The first two terms correspond to the kinetic energy of electrons and nuclei respectively. The third term describes the attractive interaction between electron and nuclei. The last two terms are the repulsive potentials of electron-electron and nuclei-nuclei interactions [59].

The Born-Oppenheimer approximation states that since the mass of nuclei

[1] Electronic structure describes the motion of electrons in atoms or molecules.

are much larger than the mass of electrons in the system, the nuclei can be regarded as fixed. Since the motion of electrons is much faster than the motion of nuclei, it is assumed that for any given nuclear configuration, electrons will be optimally distributed. From this assumption it follows that the kinetic energy term for nuclei can be removed from the equation. Only a static potential energy term will remain from atomic nuclei.

The equation for the electronic part of the Hamiltonian is then:

$$\hat{H}_{elec} = \underbrace{-\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2}_{\hat{T}} \underbrace{-\sum_{i=1}^{N}\sum_{I=1}^{M}\frac{Z_M}{r_{iI}}}_{\hat{V}_{e-n}} + \underbrace{\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}}}_{\hat{V}_{e-e}} \tag{2.3}$$

To get the total energy, the constant external potential generated by the nuclei is added to the electronic part of the Hamiltonian.

$$E_{tot} = E_{elec} + E_{nuc} \qquad \text{where} \qquad E_{nuc} = \sum_{I=1}^{M}\sum_{K>I}^{M}\frac{Z_I Z_K}{R_{IK}} \tag{2.4}$$

Now one can solve specifically for the many electron system:

$$\hat{H}\Psi(r_{i=1..N}) = E\Psi(r_{i=1..N}) \tag{2.5}$$

Note that the many electron Schrödinger equation is very difficult to solve for most systems of interest. For N electrons in a system with 3N spatial coordinates, the dimensionality of the system increases rapidly with the number of atoms. As an example, for one H_2O molecule, there are 10 electrons with 3 spatial coordinates each. For a single water molecule, there would be 30 degrees of freedom. To solve for a system containing 100 atoms, this number grows to 3000 degrees of freedom.

In 1964, Hohenberg and Kohn proved two theorems [37]:

Theorem 1 *For any system of interacting particles in an external potential $V_{ext}(r)$, the potential $V_{ext}(r)$ is determined uniquely, except for a constant, by the ground state particle density $n_o(r)$*

Theorem 2 *A universal functional[2] for the energy E[n] in terms of the density n(r)*

[2] A functional is a function of a function. Here the Energy is a function of the electron density function.

can be defined, valid for any external potential $V_{ext}(r)$. For any particular $V_{ext}(r)$, the exact ground state energy of the system is the global minimum value of this functional, and the density n(r) that minimizes the functional is the exact ground state density $n_o(r)$

By defining the electron density:

$$n(r) = N \int d^3r_2 ... \int d^3r_N \Psi^*(r_1, r_2, ..., r_N)\Psi(r_1, r_2, ..., r_N) \tag{2.6}$$

The problem is reduced from 3N spatial dimensions, to only 3 spatial dimensions. The ground state density contains all of the information required to solve the Schrödinger equation, and is the basic variable used in DFT. Following from the Hohenberg-Kohn theorems, there is a mapping between the density and external potential. Thus, the many-body wave function is a functional of the density.

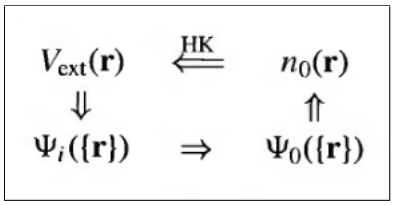

FIGURE 2.1: Theorem 1 of the Hohenberg-Kohn theorems state that the external potential $V_{ext}(r)$ is uniquely determined by the ground state density $n_o(r)$. The external potential can be used to generate all states of the system $\Psi_i(r)$ including the ground state $\Psi_o(r)$ and ground state density $n_o(r)$. The mapping from ground state density to the external potential by the Hohenberg-Kohn theorems complete the loop. Image retrieved from reference [59].

The Kohn-Sham Ansatz

Up to this point, the interacting many-body Schrödinger equation is still too difficult to solve. In the Kohn-Sham Ansatz, the interacting many-body problem is replaced by an auxiliary system of non-interacting independent particles. Here the ground state density for the non-interacting system is mapped to the ground state density for the fully interacting problem (figure 2.2). The accuracy of the model is then limited by the approximations made to the exchange-correlation

term which accounts for the difference between the non-interacting and fully interacting systems.

Returning to the electron-electron interaction potential in equation 2.2:

$$\hat{V}_{e-e} = \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{1}{r_{ij}} \tag{2.7}$$

instead of tracking all interactions between every electron in the system, the jth electron is treated as a point charge in the field of all other electrons, reducing the many electron problem to many one electron problems moving through an average potential generated by the fictitious system of electrons. The local effective potential for which non-interacting particles move in is given by

$$V_s(r) = V_{ext}(r) + V_H[n(r)] + V_{xc}[n(r)] \tag{2.8}$$

Where V_{ext} is the interaction between electrons and nuclei in the system, V_H is the Hartree potential, and V_{xc} is the exchange correlation potential. The Hartree potential (or colomb potential) represents the interactions between one electron and the average potential of all electrons in the system.

$$V_H = \int^{'} \frac{n(r')}{r - r'} dr' \tag{2.9}$$

The exchange-correlation term accounts for all of the many body effects.

$$V_{xc} = \frac{\partial E_{xc}[n(r)]}{\partial n(r)} \tag{2.10}$$

Where

$$E_{xc}[n(r)] = T[n(r)] - T_s[n(r)] + E_{ee}[n(r)] - E_H[n(r)] \tag{2.11}$$

is the difference between the kinetic energies, and the difference between the internal interaction energies of the interacting many-body system and the non-interacting fictitious independent particle system. Here, $T[n(r)]$ represents the true interacting kinetic energy, and $T_s[n(r)]$ is the independent particle kinetic energy of the fictitious system. For the differences between interacting energies, $E_{ee}[n(r)]$ is the electron-electron interaction energy and $E_H[n(r)]$ is the Hartree

energy (the term which replaces electron-electron interactions for the fictitious independent particle system).

Now the Hamiltonian for the non-interacting electrons is a sum of the one-electron Hamiltonians

$$\hat{H} = \sum_{i}^{N} \hat{h}_i \tag{2.12}$$

And the Schrödinger equation can be rewritten for each one electron Hamiltonian:

$$\hat{h}_i \Psi_i(\mathrm{r}) = \epsilon_i \Psi_i(\mathrm{r}) \tag{2.13}$$

$$(-\frac{1}{2}\nabla^2 + V_s((r))\Psi_i(r) = \epsilon_i \Psi_i(r) \tag{2.14}$$

In this representation of the electronic shrödinger equation, the eigenvector, $\Psi_i(r)$, is now the molecular orbital corresponding to electron i. The eigenvalue ϵ_i is the energy corresponding to the molecular orbital $\Psi_i(r)$.

The electron density can be reformulated as well for the auxiliary system. It is given by:

$$n(r) = \sum_{\sigma} \sum_{i=1}^{N^\sigma} |\Psi_i^\sigma(r)|^2 \tag{2.15}$$

Where σ is the spin state of the electron, and N^σ is the total number of spins. Note that $\Psi_i(r)$ depends on one coordinate vector, r, (which corresponds to one electron i) and not the set of coordinate vectors, $\{\mathrm{r}\}$, as it would if it were the fully interacting system. In the Kohn-Sham scheme, the electron density of the noninteracting system is mapped to the electron density of the fully interacting system. This mapping leads to the solution of the fully interacting problem, as the electron density determines all properties of the system. This is visualized in figure 2.2.

Solving the Kohn-Sham Equations

Before solving the Kohn-Sham equations, we should note that the solution thus far contains a part that is known and part that is unknown. The unknown part is the exchange-correlation term which accounts for differences between the fully

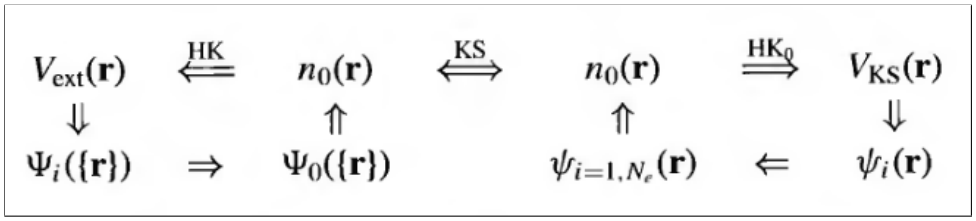

FIGURE 2.2: In the Kohn-Sham ansatz, the electron density of the noninteracting problem is mapped to the electron density of the many-body fully interacting problem. By solving the electron density for the non-interacting problem, all properties for the fully interacting problem are found. Image retrieved from reference [59]

interacting system and the noninteracting system. The known part includes everything else in the expression. The exchange-correlation term has been functionalized using many different approaches. Two common approaches include the Local Density Approximation (LDA) [45] and Generalized Gradient Approximation (GGA) [47]. In LDA, electron density is treated as a uniform gas. In GGA, a correction term is added to the local density approximation in form of a gradient.

Basis Sets

In order to efficiently solve the Kohn-Sham equations on a computer, the molecular orbitals are represented by a set of basis functions. In this ansatz, the molecular orbitals which may or may not be centered at atomic nuclei, are represented by a linear combination of atomic orbitals (LCAO):

$$\Psi_i = \sum_{\mu} c_{\mu}^{i} \Psi_{\mu}(r) \tag{2.16}$$

where $\Psi_{\mu}(r)$ are the atomic orbitals and c_{μ}^{i} are the coefficients which weight the contribution of each atomic orbital to the entire molecular orbital. To obtain the coefficients, the total energy of the system is minimized. These atomic orbitals are often Slater Type Orbitals (STO) [78] or Gaussian Type Orbitals (GTO) [14]. STOs are known to emulate the cusps near the nuclei and exponential decay at long range as seen in the exact wave functions of the hydrogen atom [33]. Evaluating STOs can be costly however. As a result, GTOs become an attractive choice as they are much easier to compute than STOs. Since GTOs are less

accurate than STOs, a linear combination of GTOs can be used to approximate the STOs. It is also common to use a plane wave basis set, where the number of plane waves used are limited by a cutoff energy. In this implementation, a plane wave basis set will often use a *pseudopotential*. The role of pseudopotentials are to replace the effects of tightly bound core electrons with an effective potential that interacts with valence electrons. Through the use of pseudopotentials, the number of electrons considered when solving the Kohn-Sham equations as well as the number of basis functions required are reduced.

Ideally, a finite basis set should approach the complete basis set (CBS) limit. Practically speaking, the basis sets are incomplete by necessity and various extrapolation methods [36, 29, 30] are used to extrapolate to the CBS limit. As noted by [28] such methods do enforce a degree of uncertainty in the accuracy of the basis functions, however, due to the approximate nature of extrapolation.

Self-Consistency Scheme

Once an approximation is chosen for the exchange-correlation term, and an appropriate set of basis functions have been chosen, the Kohn-Sham equations can be solved self-consistently. Here an initial electron density is guessed, which is used to solve the effective potential and the Kohn-Sham Equation to get a single particle orbital $\Psi_i(r)$. The density is then computed from equation 2.15 and compared to the initial guess for the electron density. If these densities are within a specified convergence criteria, then this density is the true ground state density. If the densities differ, then a new guess is used as input and the routine proceeds until the true ground state density is obtained.

When the ground state electron density is found, the total energy can be expressed as:

$$E[n(r)] = \sum_i f(\epsilon_i) \left\langle \Psi_i \middle| \left(-\frac{1}{2}\nabla^2 + \int V_{ext}(r) n(r) dr \right) \middle| \Psi_i \right\rangle + \frac{1}{2} \int \int' \frac{n(r) n(r') dr dr'}{|r - r'|} + E_{xc}[n(r)] + E_{II} \tag{2.17}$$

Where $f(\epsilon_i) \in [0,2]$ is the occupation of i^{th} molecular orbital. This is taken from the Fermi function with factor of 2 for spin [46]

$$f(\epsilon_i) = 2\Big[exp\Big(\frac{\epsilon_i - \mu}{k_B T + 1}\Big)\Big] \tag{2.18}$$

The chemical potential μ is chosen so that $\sum_i f_i = N$ where N is the number of electrons.

2.1.2 Density Functional Tight Binding (DFTB+)

Density Functional Tight Binding (DFTB+) is a tight binding approach based on the Taylor series expansion of the Kohn-Sham energy (equation 2.17) in DFT. It is faster than DFT by approximately 2-3 orders of magnitude [87], which makes it an attractive choice for exploring new materials. Since, DFTB is derived from DFT, it has the same strengths and weaknesses of DFT. The DFTB model has been implemented for up to the 3rd order expansion of the Kohn-Sham energy (DFTB1, DFTB2, and DFTB3 are the 1st, 2nd and 3rd order expansions respectively). Here we will only discuss up to the 2nd order expansion of the Kohn-Sham energy.

In the tight binding description, electrons are assumed to be tightly bound to atoms in the system, so only valence electrons are considered. A minimal basis set is used, meaning that only a single radial basis function is used for each molecular orbital [46]. The parameters (Hubbard values, Hamiltonian matrix elements and overlap matrix) from DFTB are calculated using the Perdew-Burke-Ernzerhof (PBE) [71] functional, and the diatomic repulsion potential from calculations using the Becke, three-parameter, Lee-Yang-Parr (B3LYP) [8] functional [87].

Starting from equation 2.17, the energy will be expanded to a second order fluctuation of the density $n_o(r)$. The reference density $n_o(r)$ is made up of a superposition of neutral atomic densities $n_o = \sum_\alpha n_o^\alpha$. This treatment suggests that the density contains no charge transfer. The minimizing density is defined such that

$$n_{min}(r) = n_o(r) + \delta n_o(r) \tag{2.19}$$

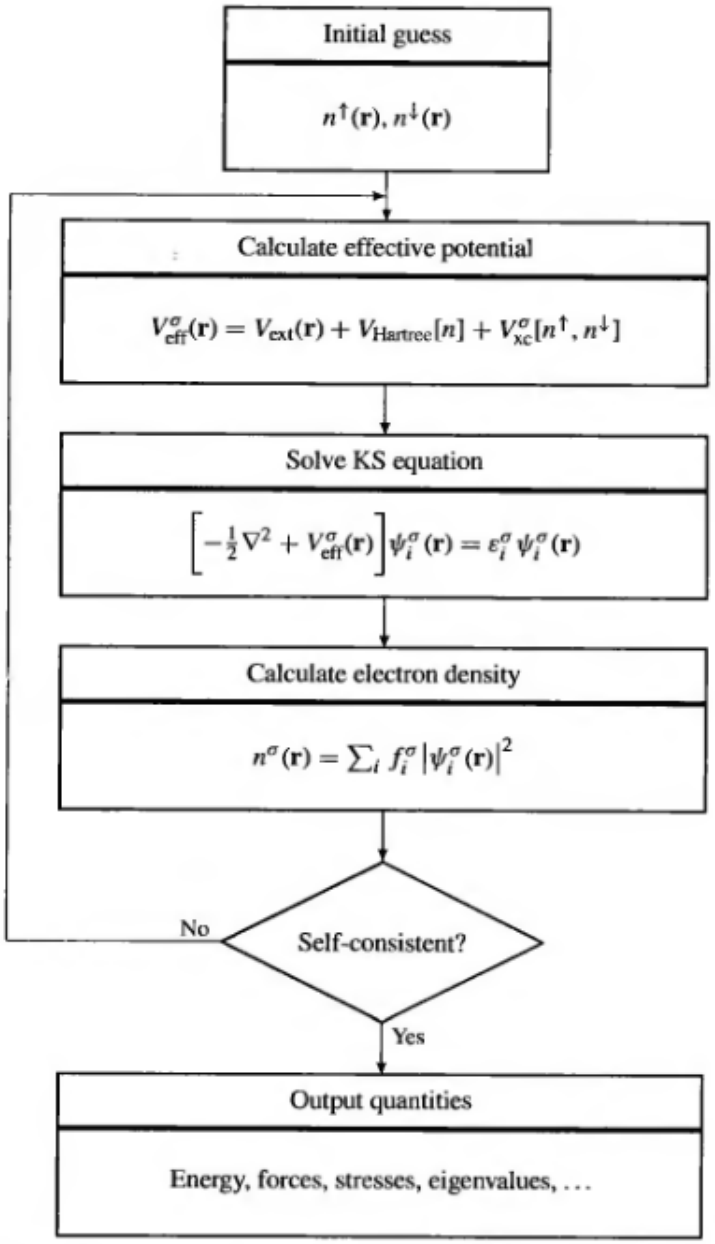

FIGURE 2.3: Kohn-Sham self-consistency scheme: an initial electron density is guessed which is used to solve the effective potential. The Kohn-Sham equation is then solved to get a single particle orbital. From this wavefunction, a new density is computed. If this density is within a specified convergence criteria, it is considered the true ground state density. If not, a new guess is made. Image retrieved from [59].

Where $n_o(r)$ is the reference density and $\delta n_o(r)$ is a small fluctuation [46]. For simplicity, the densities will be rewritten such that $n(r) \rightarrow n$ and $n(r') \rightarrow n'$. Integrals will also be simplified such that $\int d^3r \rightarrow \int$ and $\int' d^3r' \rightarrow \int'$.

Expanding the exchange-correlation energy in a Taylor series to the second order, the following expression is obtained:

$$E_{xc}[n_o + \delta n] = E_{xc}[n_o] + \int \left[\frac{\partial E_{xc}[n]}{\delta n}\right]_{n_o} \delta n + \frac{1}{2}\int' \int \left[\frac{\partial^2 E_{xc}[n]}{\delta n \delta n'}\right]_{n_o, n_o'} \delta n \delta n' \tag{2.20}$$

Returning to the Kohn-Sham energy functional from DFT, the equation can then be rewritten as:

$$\begin{aligned} E[\delta n] \approx \sum_i f_i \langle \Psi_i | H[n_o] | \Psi_i \rangle + \frac{1}{2}\int \int' \left(\frac{\delta^2 E_{xc}[n_o]}{\delta n \delta n'} + \frac{1}{|r - r'|}\right) \delta n \delta n' \\ - \frac{1}{2}\int V_H[n_o] n_o + E_{xc}[n_o] + E_{II} - \int V_{xc}[n_o] n_o \end{aligned} \tag{2.21}$$

Now the Kohn-Sham energy can be separated into first, second and third order terms: $E[n_o + \delta n_o] = E^0[n_o] + E^1[n_o, \delta n_o] + E^2[n_o, \delta n_o^2]$. DFTB1 (non-self consistent DFTB) includes only up to the first order. DFTB2 (Self consistent DFTB) contains up to the 2nd order term of the energy expansion.

The first term in equation 2.21, is the band structure energy, which can be found from summing all occupied eigenstates:

$$E_{BS}[\delta n] = \sum_i f_i \langle \Psi_i | H[n_o] | \Psi_i \rangle = \sum_i f_i \epsilon_i \tag{2.22}$$

The second term makes up the energy from charge fluctuations:

$$E_{coul}[\delta n] = \frac{1}{2}\int \int' \left(\frac{\delta^2 E_{xc}[n_o]}{\delta n \delta n'} + \frac{1}{|r - r'|}\right) \delta n \delta n' \tag{2.23}$$

And the last four terms make up the repulsive energy, which is repulsive mainly due to the dominating ion-ion repulsive interaction term:

$$E_{rep} = -\frac{1}{2}\int V_H[n_o]n_o + E_{xc}[n_o] + E_{II} - \int V_{xc}[n_o]n_o \tag{2.24}$$

The energy functional can then be rephrased:

$$E[\delta n] = E_{BS}[\delta n] + E_{coul}[\delta n] + E_{rep} \tag{2.25}$$

Up to this point, the energy has simply been expanded to the second order and the charge density has been approximated by a superposition of neutral atomic charge densities. The band structure and repulsive energies make up the energy up to the first order expansion (DFTB1). The second order terms are contained in the $E_{coul}[\delta n]$ term, and with this inclusion we obtain SCC-DFTB (DFTB2).

Energy from charge fluctuations

We will start with the second order term of the energy expansion; the energy from charge fluctuations. The expansion of the atomic energy as a function of Δq extra electrons is given by:

$$E(\Delta q) \approx E_o + \left(\frac{\partial E}{\partial \Delta q}\right)\Delta q + \frac{1}{2}\left(\frac{\partial^2 E}{\partial \Delta q^2}\right)\Delta q^2 = E_o - \chi\Delta q + U\Delta q^2 \tag{2.26}$$

Where the electronegativity[3] has the form $\chi \approx \frac{(IE-EA)}{2}$, and replaces the first derivative of energy with respect to Δq extra electrons. The Hubbard energy[4] is $U \approx IE - EA$ (where IE is the ionization energy[5] and EA is the electron affinity[6]) replaces the second derivative of energy with respect to Δq extra electrons.

The charge density fluctuation can be written as: $\delta n = n - n_o$ where $\delta n = \sum_\alpha \delta n_\alpha$ is the superposition of atomic charge density contributions. δn is approximated by the charge fluctuations at atom α, where $\Delta q_\alpha = q_\alpha - q_\alpha^o$ is the difference between the charge and the number of valence electrons on a neutral

[3]Electronegativity is the tendency for an atom to attract a shared pair of electrons in a covalent bond to itself.

[4]The Hubbard energy (chemical hardness) describes how the energy changes when electrons are added or removed.

[5]The ionization energy is the amount of energy required to remove an electron from an atom.

[6]The electron affinity is the amount of energy released when an electron is added to an atom.

atom α, computed with Mulliken charge analysis[7] [65]. The extra electrons on atom α can be written as $\Delta q \approx \int_{V_\alpha} \delta n(r) d^3r$ which can be rewritten as δn atomic contributions $\delta n(r) = \sum_\alpha \Delta q_\alpha \delta n_\alpha(r)$.

By rewriting the charge fluctuation term with δn atomic contributions, the integral becomes

$$E_{coul} = \frac{1}{2}\Delta q_\alpha \Delta q_\beta \int \int' \left(\frac{\delta^2 E_{xc}[n_o]}{\delta n \delta n'} + \frac{1}{|r - r'|} \right) \delta n_\alpha \delta n'_\beta \tag{2.27}$$

For $\alpha = \beta$ this term becomes $E_{coul} = \frac{1}{2} U_\alpha \Delta q_\alpha^2$ where U is the Hubbard energy and is twice the atom absolute hardness η. The Hubbard energy is a part of the atomic parameter set for DFTB. For larger inter-atomic distances, where $\alpha \neq \beta$, the exchange-correlation term goes away leaving only the electrostatic interaction $\left(E_{coul} = \frac{1}{2}\Delta q_\alpha \Delta q_\beta \int \int' \frac{\delta n_\alpha \delta n'_\beta}{|r-r'|} \right)$ between atomic populations Δq_α and Δq_β. To compute the energy due to charge fluctuations, a functional form for the charge density must be assumed.

The function $\gamma_{\alpha\beta}$ approximates the integrand for the charge fluctuation term, and it is rewritten such that

$$E_{coul} = \frac{1}{2} \sum_{\alpha\beta} \gamma_{\alpha\beta}(R_{\alpha\beta}) \Delta q_\alpha \Delta q_\beta \tag{2.28}$$

where

$$\gamma_{\alpha\beta} = \begin{cases} U_\alpha & \alpha = \beta \\ \frac{\delta n_\alpha \delta n'_\beta}{|r-r'|} & \alpha \neq \beta \end{cases} \tag{2.29}$$

Energy from repulsive interactions

The repulsive energy term is approximated as a sum of pair potentials which are fit to a relevant set of molecules.

$$E_{rep} = \sum_{\alpha<\beta} V_{\alpha\beta}(R_{\alpha\beta}) \tag{2.30}$$

[7] Mulliken charge analysis is a method used to estimate the partial atomic charges primarily for the linear combination of atomic orbital molecular orbital method.

This repulsive energy depends only on the reference density n_o. Since the reference density n_o is the superposition of neutral atomic densities, the repulsive energy term does not depend on a specific atomic environment [61]. This means that the potential can be applied to other molecular environments once obtained for the reference system. The repulsive potential is obtained by fitting to a higher level DFT potential, or by fitting to empirical parameters. It contains all core electron effects. These inter-atomic potentials belong to the diatomic parameter set for DFTB.

Band structure energy

The last term remaining in the total energy expression is the band structure energy. As mentioned previously, the repulsive energy contains core electron effects. Now the effects of valence electrons are considered. Since core electrons are considered to be tightly bound, a minimal local basis set is used to consider only valence electrons:

$$\Psi_i = \sum_{\mu} c_{\mu}^{a} \phi_{\mu} \tag{2.31}$$

Where pseudoatomic orbitals ϕ_μ are taken from DFT calculations on the atoms that are involved.

The band structure energy is a first order term that can then be expanded such that

$$E_{BS} = \sum_i f_i \sum_{\mu\nu} c_{\mu}^{a} c_{\nu}^{a} H_{\mu\nu}^{o} = \sum_i f_i \epsilon_i \tag{2.32}$$

where E_{BS} is the sum of occupied Kohn-Sham energies. The Hamiltonian

$$H_{\mu\nu}^{o} = \langle \phi_\mu | H^o | \phi_\nu \rangle \tag{2.33}$$

contains pre-tabulated matrix elements.

Returning to the total energy expression, up to the 2^{nd} order expansion the total energy is:

$$E = \sum_i f_i \sum_{\mu\nu} c_{\mu}^{a} c_{\nu}^{a} H_{\mu\nu}^{o} + \frac{1}{2} \gamma_{\alpha\beta}(R_{\alpha\beta}) \Delta q_\alpha \Delta q_\beta + \sum_{\alpha<\beta} V_{\alpha\beta} \tag{2.34}$$

The eigenvalue problem is now:

$$\sum_{\nu} c_{\nu}^{i} H_{\mu\nu} = \epsilon_i \sum_{\nu} S_{\mu\nu} \tag{2.35}$$

Note that:

$$H_{\mu\nu} = H^{o}_{\mu\nu} + \frac{1}{2} S_{\mu\nu} \sum_{K} \Delta q_K (\gamma_{\alpha K} + \gamma_{\beta K}) \qquad \mu \in \alpha \quad \nu \in \beta \tag{2.36}$$

where $S_{\mu\nu}$ is the overlap matrix

$$S_{\mu\nu} = \langle \phi_\mu | \phi_\nu \rangle \tag{2.37}$$

and $\gamma_{\alpha K}$ and $\gamma_{\beta K}$ are functions used in approximating the charge fluctuation terms for atoms α and β respectively. In the tight binding description the Hamiltonian $H^{o}_{\mu\nu}$ and overlap $S_{\mu\nu}$ matrix elements are pre-tabulated. For the diagonal elements of $H^{o}_{\mu\nu}$, a one-center approximation is made and $H^{o}_{\mu\mu} = \epsilon_\mu$ (ϵ_μ is the Kohn-Sham eigenvalue for a neutral unconfined atom [23]). The non-diagonal terms use the 2-center approximation so $H^{o}_{\mu\nu} = \langle \phi_\mu | H(n^{\alpha}_{o} + n^{\beta}_{o}) | \phi_\nu \rangle$.

The electrostatic potential due to charge fluctuations on α is $\epsilon_\alpha = \sum_K \gamma_{\alpha K} \Delta q_K$. Then

$$H_{\mu\nu} = H^{o}_{\mu\nu} + h^{1}_{\mu\nu} S_{\mu\nu} \tag{2.38}$$

with

$$h^{1}_{\mu\nu} = \frac{1}{2}(\epsilon_\alpha + \epsilon_\beta) \tag{2.39}$$

In this representation, the true Hamiltonian matrix elements are given by the reference Hamiltonian, plus a shift due to charge fluctuations which is the average electrostatic potential around orbitals μ and ν [46].

From here equation 2.35 and 2.36, need to be solved self-consistently. Starting from an initial guess for $\{\Delta q\}$, $h^{1}_{\mu\nu}$ and $H_{\mu\nu}$ are obtained. Solving the eigenvalue problem (equation 2.35), the new coefficients $\{c^{i}_{\mu}\}$ are obtained and used to solve for new $\{\Delta q\}$. If this new $\{\Delta q\}$ is close enough to the initial guess, i.e. within the SCC convergence threshold, the solution is obtained. Otherwise,

a new guess for $\{\Delta q\}$ is chosen and this process repeats iteratively until self-consistency has been achieved. This routine is summarized in figure 2.4. Generally speaking, for DFTB, less self-consistent iterations are required than are needed for a full DFT calculation. However, this self-consistency check makes DFTB2 and DFTB3 about 5-10 times slower than non-self-consistent DFTB1. This is due to the fact that it often takes approximately 5-10 iterations to solve the eigenvalue problem (equation 2.35) [61].

pseudo-atoms

If the atomic orbitals are derived from free atoms, they would be too diffuse. To ensure a compact basis, a confinement potential term is introduced to the Hamiltonian of the Kohn-Sham equation for atomic orbitals. This confining potential takes the form $V_{conf}(r) = \left(\frac{r}{r_o}\right)^2$ where a reasonable choice for r_o is twice the covalent radius ($r_o = 2r_{cov}$) [72]. The compact basis set is then solved from

$$[T + v_{eff}(n) + \left(\frac{r}{r_o}\right)^2]\phi_\nu = \epsilon_\nu \phi_\nu \tag{2.40}$$

where the Hamiltonian is modified with an additional confinement potential. The effective potential is dependent on the electron density for neutral atom α. The density is determined from the Kohn-Sham equation. Another confinement radius r_o^d must be chosen for this initial density [24]. This generates a pseudo-atom. Note that confinement radii r_o and r_o^d are adjustable parameters.

Parameter Sets

For SCC-DFTB there are two types of parameters: atomic and diatomic parameters. The atomic parameters consist of the two confinement radii r_o and r_o^d for the atomic orbitals, the Hubbard parameter used in the charge fluctuation term, and the spin polarization energy (necessary only to compute heats of formation) [32].

The second type of parameters are the diatomic parameters. This term includes all of the pair potentials from E_{rep}. These are typically obtained by fitting to a higher level DFT calculation.

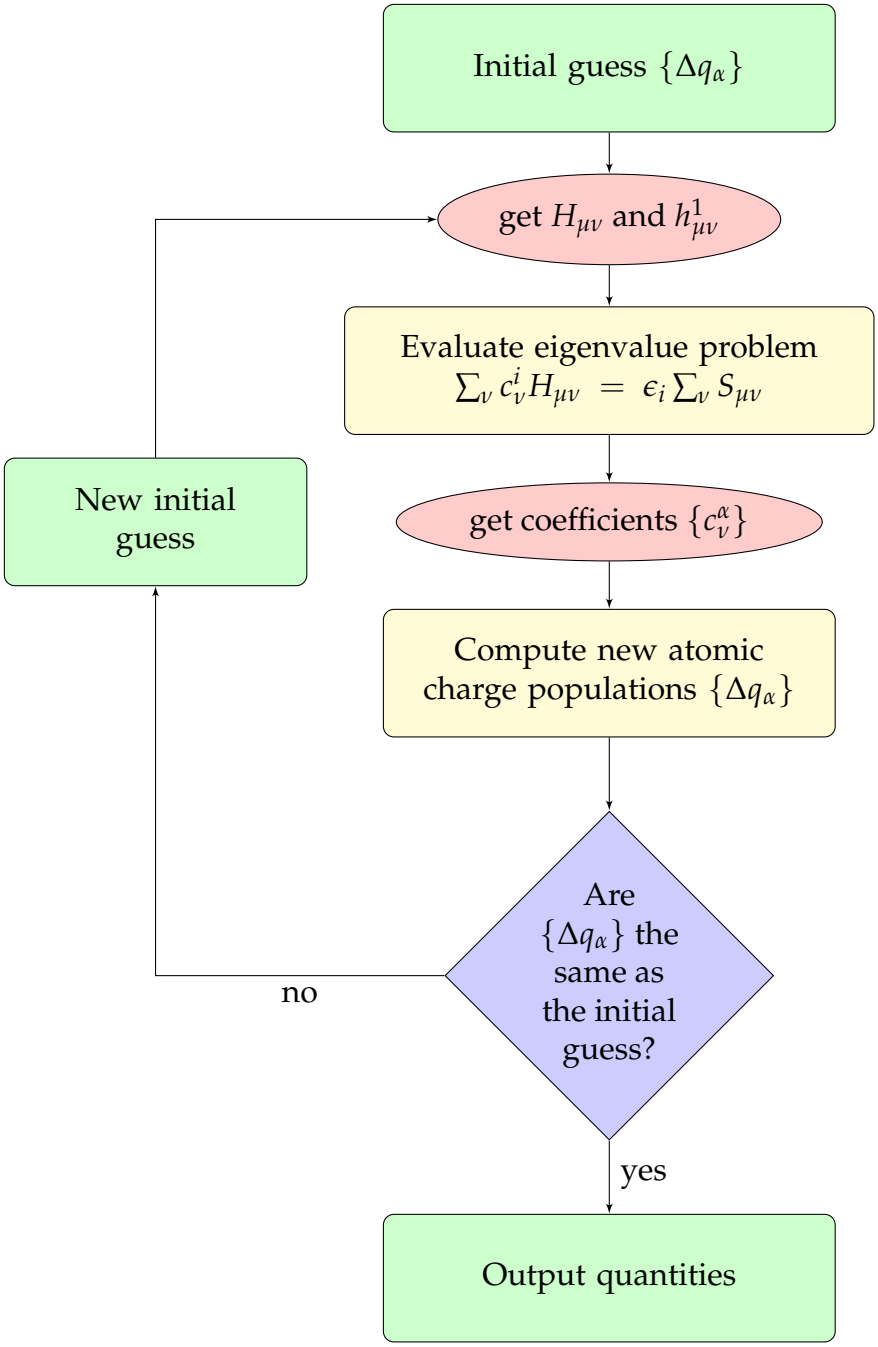

FIGURE 2.4: Self-consistency in SCC-DFTB method. In the Kohn-Sham equations, the charge density was solved self-consistently. Here the atomic charge population is solved self-consistently. It typically takes much fewer iterations to solve than for full DFT.

Implementation Details for DFT and DFTB

When computing the electronic wavefunction for a periodic system in real space, the number of interactions between electrons and ions becomes infinitely large. However, for a periodic system, it is reasonable to assume a periodic potential which a given electron is interacting with.

In a periodic crystal structure, the atomic coordinates can be represented by a set of translations on a repeating basis:

Crystal Structure = Bravais Lattice + Basis [59]

Where the basis is defined by a primitive cell made up of primitive lattice vectors, and the types of atoms. This primitive cell is the smallest repeat unit which can generate an entire crystal through translation operations. A Bravais Lattice is the set of translations $T(n) = n_1a_1 + n_2a_2 + n_3a_3$ (where a_i are lattice vectors in the i^{th} direction and n_i are integers) which can generate a crystal lattice. The Wigner-Seitz cell is the most compact cell centered around a single lattice point in real space.

Since a lattice is periodic by nature, the coordinates of each lattice point can be represented by a periodic function $f(r + T) = f(r)$. Consequently, this periodic function can be mapped onto reciprocal space via Fourier transform. The first Brillouin Zone is defined by the Wigner-Seitz cell in reciprocal space. The Brillouin Zone is the unit cell on the reciprocal lattice with reciprocal lattice vectors. The reciprocal lattice is defined by $G(m) = m_1b_1 + m_2b_2 + m_3b_3$ where b_j are reciprocal lattice vectors and m_j are integer values which translate the reciprocal lattice vectors. The reciprocal lattice vectors are related to the real space lattice vectors by:

$$b_1 = 2\pi\frac{a_2 \times a_3}{a_1 \cdot a_2 \times a_3} \tag{2.41}$$

$$b_2 = 2\pi\frac{a_3 \times a_1}{a_2 \cdot a_3 \times a_1} \tag{2.42}$$

$$b_3 = 2\pi\frac{a_1 \times a_2}{a_3 \cdot a_1 \times a_2} \tag{2.43}$$

The Bloch wave is given by the product of a periodic potential $u_k(r + T_n) = u(r)$ and a plane wave $e^{ik\cdot r}$ such that

$$\Psi(r) = e^{ik\cdot r}u_k(r) \tag{2.44}$$

where k is the wave vector and r is the position and Ψ is the Bloch wavefunction. The Bloch wave allows the summation of electronic interactions over an infinite number of translations to be transformed into an integral over the first Brillouin Zone. This integral is replaced by a weighted sum over a discrete set of wave vectors.

The periodic potential can be described as a Fourier series, $u_k(r) = \sum_G c_k exp(G \cdot r)$. To exactly solve Kohn-Sham equations in a plane-wave basis, the number of basis functions goes to infinity. Since we are interested in the ground state energy, the most important wave-functions have a low kinetic energy $E = \frac{\hbar}{2m}|k+G|^2$. Thus, the basis set can be reduced to include only the low kinetic energy range. The truncation value is called the kinetic energy cut-off $E_{cut} \geq \frac{1}{2}|k+G|^2$. The kinetic energy cut-off is a convergence parameter which should be tested. Here the Brillouin zone is sampled by a set of K-points. Increasing the number of K-Points increases the resolution of the calculation.

K-Point Sampling

Since any k-point in the reciprocal lattice is essentially equivalent given the periodic nature of the lattice ($k' = k + G$), only the integral over the first Brillouin Zone needs to be evaluated. The wave function is sampled at multiple k-points in the Brillouin Zone. In the Monkhorst-Pack algorithm the integral is approximated by a set of equidistant k vectors with an identical weight [63]:

$$k_{n_1,n_2,n_3} = \sum_i^3 \frac{2n_i - N_i - 1}{2N_i} G_i \tag{2.45}$$

where G_i are the primitive vectors.

2.1.3 Running Molecular Dynamics

Molecular dynamics are a way of evolving a set of configurations through time. These simulations are a way of sampling equilibrium states, and sometimes non-equilibrium states [38]. Prior to running MD, an energy minimization can be performed to avoid situations such as atomic overlap (this would lead to high energies in the MD simulations). This can be achieved through energy optimizations where the coordinates and/or lattice vectors are relaxed. During a relaxation the coordinates can change position or lattice vectors change in

size, in whichever way minimizes the energy in the system. MD simulations require the definition of a potential to describe inter-atomic interactions. The derivative of the potential energy with respect to atomic coordinates provides the force. Once the force is obtained, one can integrate the equations of motion and update the positions of atomic nuclei.

2.2 Machine Learning Basics

Machine learning is a method of data analysis which automates model building. This allows computers to 'learn' patterns from data without being explicitly programmed to do so. The goal of a machine learning algorithm is to find a generalizable mapping between input and output variables. Once this mapping is obtained, it can be evaluated as a function to predict output(s) given an input. Machine learning can be categorized into two classes of algorithms; supervised and unsupervised learning.

A key distinction between supervised and unsupervised learning is the availability of feedback. In supervised learning, inputs as well as outputs are provided as training data. In unsupervised learning, the goal is to find patterns within the data without any feedback.

The machine learned potentials explored in this work are produced using the supervised learning method. Here, the network learns a general mapping from the atomic coordinates to energies and forces. Reference data is obtained by generating a large number of molecular configurations and evaluating the energy and forces at those points in configuration space (using one of the aforementioned simulation methods).

In the following, the mechanisms of feed-forward artificial neural networks will be discussed, as well as optimization methods and practices for training neural networks. Finally, the application of these methods to neural network potentials is outlined and the implementation in the Atomic Energy Network (ænet) code is introduced.

2.2.1 Feed-forward Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired by neurons in the brain. A neuron can receive, process and transmit information as a signal to other neurons in

a network. The connection of many neurons creates a *neural network*. In a single neuron, dendrites receive an input signal from other neurons, and the soma or cell body sums these signals. If the accumulation of these signals exceed some threshold, the neuron will fire sending a signal through the axon to the synapses where it can be transmitted to other neurons.

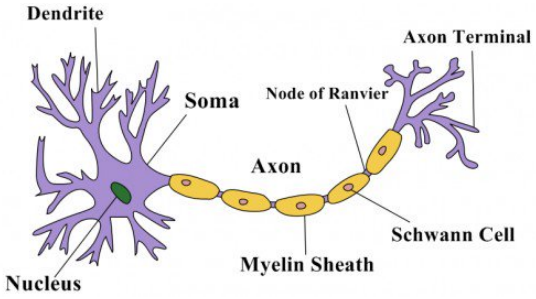

FIGURE 2.5: In a biological neuron, the dendrites receive information through an input signal. Each dendrite passes a signal to the cell body to be summed and activated. If the signal exceeds some threshold, it is propagated through the axon to be transmitted to another neuron. Image retrieved from [43].

Artificial neural networks are simplified models of these biological neurons. The most basic unit of a neural network is an artificial neuron, which is represented graphically as a node. In one artificial neuron, a vector of real valued inputs replace the dendrites of a biological neuron. These input nodes have weighted connections to a node in the next adjacent layer. Each weight specifies how much emphasis to put on the signal from its corresponding input node. As in the soma of a biological neuron, the (weighted) signals are summed and passed through an activation function. In a biological neuron, if a certain threshold has not been exceeded, a signal is not transmitted. If the input signals exceed the threshold, the neuron fires. The firing of the neuron is then best described by a Heaviside step function which produces a binary output. Using a step function for the activation however would result in numerical instabilities from the discontinuity in the step. Also, for the specific task of constructing a function to describe the PES for atomic environments, a continuous range of output values is desired. Therefore, a smooth nonlinear function such as a sigmoid or hyperbolic tangent is often used instead. The role of the activation function is to determine whether or not the signal is transmitted, and at what strength it is transmitted.

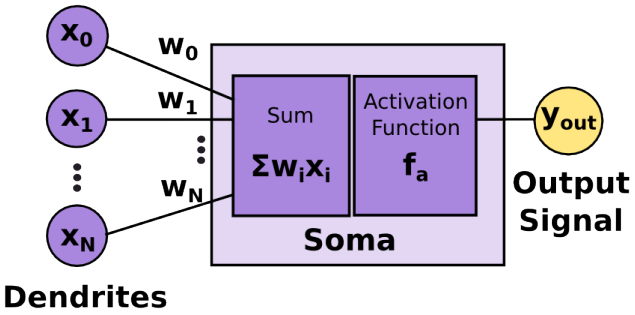

FIGURE 2.6: Representation of a single artificial neuron. The nodes represent inputs or outputs, and connections indicate weights. Image adapted from [17].

A neural network is a collection of artificial neurons connected to one another in various ways. The network is organized into layers, where each layer has a set of nodes. As mentioned previously, each node is a single artificial neuron (figure 2.6). In a fully connected network, all input nodes in one layer are connected to all nodes of the next layer. The connection between nodes are analogous to the synapse where a signal can be transmitted between connected nodes. The network weights assigned to each connection are the fitting parameters of the neural network. They are optimized and adjusted while the neural network is learning.

Feed-forward artificial neural networks are a class of ANN where there is no recurrence i.e. information only propagates forward throughout the network. In a multilayer perceptron neural network (deep feed-forward neural network) there are multiple hidden layers. Hidden layers, made up of hidden units, exist between the input and output layers and do not have any physical meaning. The hidden layers increase the functional flexibility of the neural network allowing the network to learn a more complex function.

Each hidden unit contains an activation function which is determined by the user. Some common activation functions include sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLu) [66] amongst others. The network weights determine the steepness of the activation functions and the inclusion of bias nodes allows the activation function to be shifted. Bias nodes are input nodes that always have the value one. They remain unconnected to the previous layer, and only have connections to the next layer as an input node.

As a simple analogy, let's say that a neural network was learning the equation of a line: $f(x) = w_1 \cdot x + w_b$. The bias node would be analogous to the constant intercept value w_b. The weight from the non-bias unit would be analogous to the slope, w_1.

The activation functions available in ænet are the linear (identify) function,

$$f_{linear}(x) = x \tag{2.46}$$

hyperbolic tangent,

$$f_{tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.47}$$

sigmoid,

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.48}$$

and hyperbolic tangent with linear twisting [50].

$$f_{twist}(x) = (1.7159)\tanh\left(\frac{2}{3}x\right) + ax \tag{2.49}$$

These functions are visualized in figure **??**. The activation function represents the firing rate of the neuron. The linear (or identity) function maps the input signal proportionally to the output signal onto a range of activations that is unbounded (the output values range from $(-\infty, \infty)$). If a network is made up of only linear activations, it can not learn complex functional mappings. Also, no matter how many layers exist between input and output, the multi-layer network is equivalent to a one layer network (with 0 hidden layers). The mapping is only a linear transformation from input to output.

To represent complex functions, non-linear activation functions should be used. A two layer (1 hidden layer) feed-forward neural network using a finite number of neurons and non-linear activation functions has been shown to be a universal function approximator [31, 20, 76]. This means that the network can approximate continuous functions on a compact set of R^N arbitrarily well [19]. Recent work has shown that the universal approximation theorem also holds for unbounded nonlinear activation functions [80]. Note that since a network of linear activations of any depth reduces to one layer, it cannot be a universal

function approximator.

The sigmoid function is a nonlinear activation function with output bound from (0,1). This provides a good analogy to the biological neuron; if the neuron is not firing, the activation is 0. If the neuron is firing at maximum frequency, the activation is 1. The sigmoid function is also a differentiable function which allows for backpropagation (section 2.2.2) during neural network training. One limitation of the sigmoid activation function is that if the input values are too large, the output gets pushed to the tails of the sigmoid at 0 and 1 (0 for large negative inputs, 1 for large positive inputs). This causes the activation output to respond less to changes in input values. As a result, the gradients can become small, slowing the update of network weights; in some cases (for deep networks) the weights may stop updating entirely due to the vanishing gradient problem. Special care also needs to be taken during the initialization step, since if weights are initialized too large prior to training with sigmoid activations, the output will be pushed to the tails of the sigmoid and prevent the network from learning. Another limitation is that sigmoid outputs are not zero-centered. Since the output of a sigmoid is always positive, the gradient on the weights will be all positive or all negative during backpropagation. This can lead to increased difficulty during optimization.

The hyperbolic tangent function is a rescaled sigmoid function that is zero-centered and bound from (-1,1). The tanh function is related to the sigmoid function by: $f_{tanh}(x) = 2(f_{sigmoid}(2x)) - 1$. Since it is zero-centered, the network optimization tends to be easier. The neurons saturate at the tails of tanh, however, so vanishing gradients can still be an issue.

ænet includes the tanh with linear twisting function, which was originally proposed by LeCun et al, as an improvement to the tanh function [49]. The rescaling of tanh and the addition of a small linear term is meant to help avoid saturation at the tails of the tanh function. The twist function is recommended for general purposes in ænet [3].

The functional form of the neural network is determined by the number of layers and the number of neurons. Weights are fitting parameters, which are essentially coefficients for linear combinations. The value of the output for the

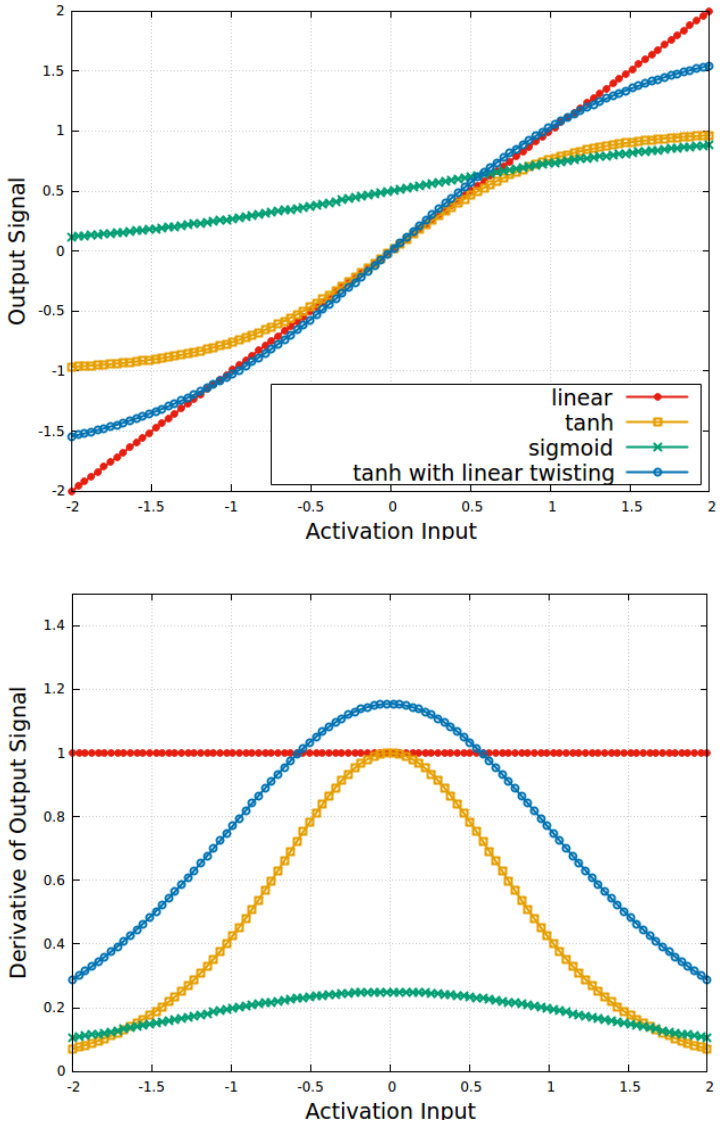

FIGURE 2.7: Activation functions available in ænet and their derivatives. Image adapted from [3] **Top:** Activation function outputs. Notice that the sigmoid and tanh function are bound on (0,1) and (-1,1) respectively. Whereas the linear and tanh with linear twisting functions are unbounded. **Bottom:** Activation function derivatives. Notice that small valued outputs for the sigmoid function can lead to the network updating weights more slowly during training while using a backpropagation algorithm (section 2.2.2).

i^{th} neuron in layer j (for $j = 1, ..., N-1$ layers), is given by:

$$x_{ij}(x_{k,j-1}) = f_a^{ij}(\sum_{k=1} w_{i,k}^{j-1} x_{k,j-1} + b_{ij}) \tag{2.50}$$

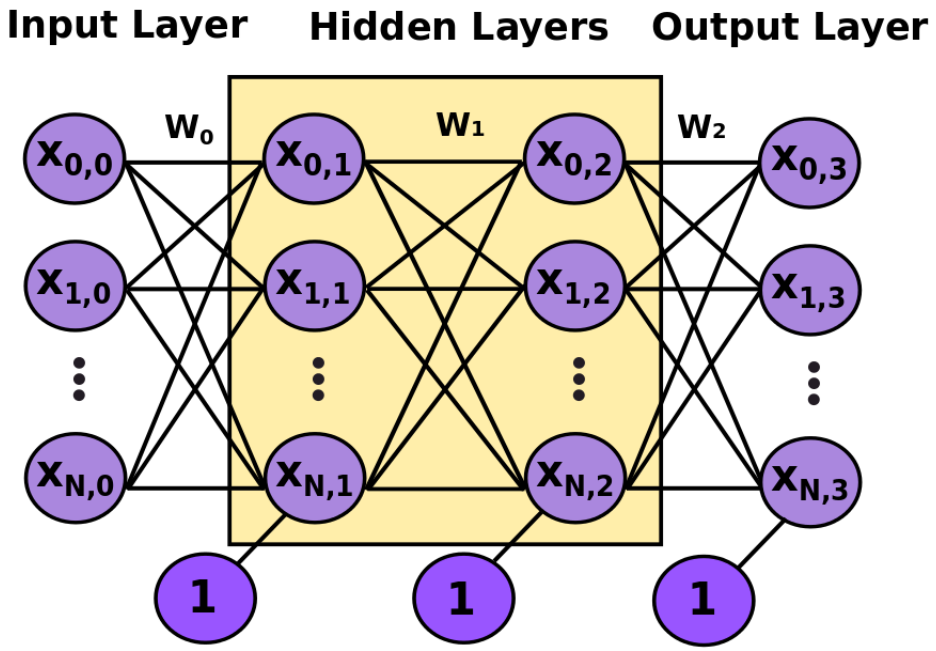

FIGURE 2.8: Fully connected artificial neural network with two hidden layers and N nodes. The set of weights for each layer is given by W. Neurons with a value of 1 indicate bias nodes.

The input signals from the previous layer are given by the set $x_{k,j-1}$ where k is the index for each neuron in previous layer $j-1$. The weight of the signal is given by $w^j_{i,k}$ and connects neuron i in layer j, with neuron k in the previous layer j-1. The weight is multiplied by neuron $x_{k,j}$ of the previous layer. A bias node b_{ij} which is connected only to layer j (not to previous layer j-1), is added to the sum of the weighted signals. This serves the purpose of shifting the linear combination. The sum of the weighted signals and bias node is used as the argument for the activation function f^{ij}_a and produces an output signal (creating an output node).

The can be rewritten in a vectorized format such that

$$X_j(X_{j-1}) = f^j_a(W_j X_{j-1} + b_j) \tag{2.51}$$

where $j = 1, ..., N-1$ for N layers. Note that the bias nodes have a weight unconnected to the previous layer with an input signal of one.

The number of weights in the neural network is given by

$$N_w = \sum_{k=1}^{M+1} \left(N_{k-1} \cdot N_k + N_k \right) \tag{2.52}$$

Where M is the total number of hidden layers, and N_k are the number of neurons in layer k.

The functional form of the neural network in figure 2.8 is given by

$$F_{i,3} = f_i^3 \left\{ b_i^3 + \sum_{k=1}^{N} w_{k,i}^2 \cdot f_k^2 \left[b_k^2 + \sum_{j=1}^{N} w_{j,k}^1 \cdot f_j^1 \left(b_j^1 + \sum_{i=1}^{N} a_{i,j}^0 \cdot x_{i,0} \right) \right] \right\} \tag{2.53}$$

During forward propagation, information passes through the network from input layer to output layer once. A cost function can be defined which provides a way to compare the output to the ground truth or target values. This is essentially computing an error. Through back propagation, this error is passed back through the network to update the weights.

2.2.2 Backpropagation

Backpropagation refers to a method where the calculated errors are propagated back through the network layers. This allows for the weights to be adjusted to minimize the error between network output and the target value [74]. Given an error function, the gradient of the error function is taken with respect to the neural network weights. The error or loss function is a measure of the difference between the output of the network and the ground truth value. Since the gradient of the final layer of weights is calculated first, and then carried through to the first layer, the error is propagated backward through the network.

The goal of backpropagation is to minimize the error ε with respect to a set of optimal weights $w_{j,k}^l$

$$\frac{\partial \varepsilon}{\partial w_{jk}^l} \tag{2.54}$$

where w_{jk}^l is the weight connecting neuron j in layer l, with neuron k of previous layer $l-1$. The cost, or error, function may be given by something like the mean

squared error:

$$\varepsilon = \frac{1}{2}(a^L - \hat{y})^2 \tag{2.55}$$

where a^L is the activation output in the final layer (the output of the entire neural network) and $\hat{y}$ is the target value. The output of each node in the neural network can be given by:

$$a_j^l = f_a(w_{jk}^j a_k^{l-1} + b_j^l) = f_a(z_j^l) \tag{2.56}$$

To simplify the expression, the weighted activation input is expressed as $z_j^l = w_{jk}^l a_k^{l-1} + b_j^l$.

The cost function can be decomposed into individual error contributions such that:

$$\frac{\partial \varepsilon}{\partial w_{ik}^j} = \frac{1}{N} \sum_{m=1}^{N} \frac{\partial \epsilon_m}{\partial w_{ik}^j} \tag{2.57}$$

This is the average over the error contributions from each training example ($m \in N$).

The error of neuron j in layer l is given by:

$$\delta_j^l = \frac{\partial \varepsilon}{\partial z_j^l} \tag{2.58}$$

This error shows how perturbing the activation input effects the cost function ε.

Since the error at each neuron within the networks depends on the error in the last layer of the network, the error in the output layer must be computed first. It is given by the following expression:

$$\delta_j^L = \frac{\partial \varepsilon}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \varepsilon}{\partial a_j^L} f_a'(z_j^L) \tag{2.59}$$

The $\frac{\partial \varepsilon}{\partial a_j^L}$ term represents how much the cost changes with respect to the final activation output. For the mean squared error cost function, this simplifies to $\frac{\partial \varepsilon}{\partial a_j^L} = (a^L - \hat{y})$. The $\frac{\partial a_j^L}{\partial z_j^L}$ term represents how much the activation changes with respect to the weighted activation input.

In a vectorized notation, the error in the output layer can be expressed as:

$$\delta_j^L = \nabla_a \varepsilon \odot f_a'(z^L) \tag{2.60}$$

where $\nabla_a \varepsilon$ is a vector of partial derivatives of the cost function with respect to activation outputs.

To get the errors in the hidden layers, the error must be passed backwards from the output layer to the hidden layers. The error in layer l depends on the errors in the proceeding layers of the network eg. layer l+1, l+2, to the final layer L.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f_a'(z^l) \tag{2.61}$$

This equation depends on 2.60, and with these two equations, the error can be backpropagated through the network.

Now we can compute the total contribution of the change in cost function with respect to network weights and biases. This equation is given by:

$$\frac{\partial \varepsilon}{\partial b_j^l} = \delta_j^l x \tag{2.62}$$

This means that the error is equivalent to the change in cost function with respect to bias weights.

Finally, the expression for how the cost function changes with respect to any weight in the network is given by:

$$\frac{\partial \varepsilon}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.63}$$

When $a_k^{l-1}\delta_j^l$ is a large value, the weights will rapidly change during optimization. Likewise if it is small, the weights will not update very much during optimization. If an activation function like sigmoid or tanh is used and inputs get pushed to the the tails, the activation function derivative changes very slowly. Therefore small errors are obtained and propagated through the network. In deep neural networks, this may lead to the vanishing gradient problem. This

happens when the error shrinks exponentially with the number of layers, preventing the weights from being updated beyond the first few layers. This problem occurs primarily when using deep neural networks with many layers.

Putting everything together, the backpropagation algorithm can be implemented by following these steps:

- Generate training examples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$ and initialize weights.
- Implement forward propagation to generate network outputs a^L, and compute activations a_j^L and their weighted inputs z^L.
- Compute the error in the last layer δ^L using equation 2.60.
- Backpropagate the errors δ^l through the network using equation 2.61.
- Output the gradient of the cost function with respect to weights and biases using equations 2.62 and 2.63.
- Average over the all training examples to compute the total derivative of the cost function with respect to weights and biases.
- Update weights and biases by moving in the negative direction of the gradient of the cost function.

2.2.3 Online vs. Batch Training Methods

There are various different methods for evaluating the total error function ε. When all the samples in the training set are used to generate the error function, *batch* training methods are being used. This means that before the weights are updated, all samples must be accounted for in the error function. As a result, batch learning can be computationally expensive for large datasets.

Alternatively, *online* training methods can be used. In online training, data from the training set is shown sequentially to the network, and weights are updated accordingly. For example, one sample from the training set can be used to generate an approximate error function and the weights will be adjusted based on this sample alone. This allows for progress to be made on the optimization at every sample in the training set instead of updating the weights only after the error of all samples is computed.

Another method which can be used to reduce the computational cost of a batch training method is to use *mini-batching*. Mini-batching is a method which uses a subset of the full batch size of training data to estimate the error function.

2.2.4 Optimization Methods

As previously discussed, training a neural network requires the minimization of a loss function. There are three different optimization schemes included in ænet.

Gradient descent is a first order optimization method. To find minima, at each iteration the negative gradient of the function is taken at its current point (in this case the loss function). Next the weight parameters are updated with respect to the negative gradient of the loss function:

$$w^{(I+1)} = w^{(I)} + -\gamma \nabla \varepsilon \tag{2.64}$$

where I is the iteration, w are the weights, ε is the error function and γ is a hyperparameter [8] called the learning rate. For online training, the weight updates are given by:

$$\Delta w_{\alpha,n}^{I+1} = -\gamma \frac{\partial}{\partial w_\alpha} \left(\frac{1}{2} e_n^2 \right) = -\gamma e_n \frac{\partial a^L}{\partial w_\alpha} \tag{2.65}$$

Selecting a learning rate that is too small will lead to slow convergence to a solution, whereas chosing a learning rate that is too large can prevent convergence to a minimum or even cause the solution to diverge.

Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a quasi-newton method and is an approximation to the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. In this case, the weights are updated using the inverse Hessian matrix, H. In this context the Hessian matrix is a matrix of second derivatives of the network function with respect to the weights:

$$\Delta w^{I+1} = -(H^{(I)})^{-1} \nabla \varepsilon^I \tag{2.66}$$

[8] A hyperparameter is a parameter whose value is set prior to training. This distinguishes it from parameters that are determined during training.

The difference between L-BFGS and BFGS is that in the L-BFGS method, instead of storing all the values of the inverse Hessian matrix, only some vectors are stored. L-BFGS is useful when there are a large number of weight parameters [3].

Levenberg-Maquardt is a method used to solve non-linear least squares problems. In this method, the weights are updated using the Jacobian $J = \frac{\partial \mathcal{N}}{\partial w}$ where $\mathcal{N}$ is the network function and w are the weights. The weights are updated by:

$$\Delta w^{I+1} = -(J^{T,(I)} J^{(I)} + \lambda I)^{-1} J^{T,(I)} e^{(I)} \tag{2.67}$$

where e is the error of each sample in the reference set, and λ is the reciprocal learning rate.

2.2.5 Model Validation

To test whether the model is generalizable to an unseen dataset, it is essential to use validation methods. To validate the model, a set of data is reserved apart from the training data. The error is evaluated on data in the training set as well as the testing data set. If the error on the testing data diverges away from the error on the training data, it is likely that overfitting has occurred. Overfitting occurs when the network fits to noise as well as training data. If the training error continues to decrease while the testing error increases, the model may do a good job at predicting the output of examples it has already seen, but it will not be able to extrapolate well to data outside of the training set.

2.2.6 Machine Learning in Atomistic Simulations

Neural Network Potentials (NNPs) can describe complicated potential energy surfaces that arise for:

- interaction types such as covalent, ionic, or metallic bonding and dispersion interactions [11]
- complex reaction or transition pathways [11]
- unusual atomic environments arising from amorphous systems, or structures which emerge from phase transitions. [11]

A neural network can generate a function of arbitrary form. This is particularly useful when the potential energy surface is very difficult to solve analytically and a physically inspired functional form will thus, not suffice.

2.2.7 Behler-Parinello Symmetry Basis Functions

Prior to Behler-Parinello symmetry functions, implementations of machine learning to atomistic simulations directly used Cartesian coordinates $\mathcal{R}_i$ as inputs for Artificial Neural Networks (ANNs) or fit to a specific number of degrees of freedom [53]. ANNs were trained to predict the structural energy $E(\sigma)$ given a set of atomic configurations $\sigma = \mathcal{R}_i$, i.e. $E(\sigma) = E^{ANN}(\sigma) = \mathcal{N}(R_i)$. This method does not generalize well. If structural energies depend on the Cartesian coordinates of atomic configurations, the potentials become highly specialized [3]. One would not be able to accurately predict the energy on a system with a fewer number of atoms for instance.

One solution to this issue is to represent the structural energy as a function of *local atomic environment*. In this case, $E(\sigma) \approx \sum_i^{atoms} E_i(\sigma_i)$ where σ_i is the set of coordinates for a local structural environment, and E_i is the energy contribution from that local structural environment. The total energy of the system would then be the sum of the energy contributions.

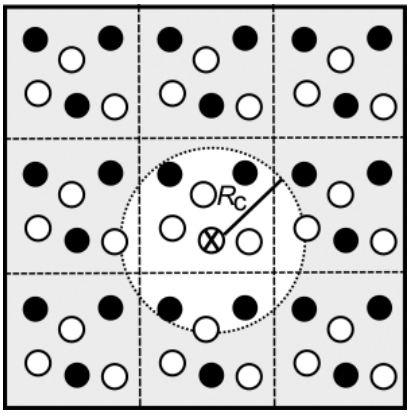

FIGURE 2.9: Local structural environment of an atom including periodic images. Interactions between atoms within the cutoff radius and the centered atom are included. Image retrieved from [11].

Another desirable feature is invariance to translations, rotations, and invariance to permutations of equivalent atoms in the system. For instance, if the hydrogen atoms on a water molecule are swapped, the potential energy should not change (the OH bond length is equivalent and the molecule is symmetric about the oxygen atom)[11].

Invariance to translations, rotations and exchange of equivalent atoms can be ensured by projecting the Cartesian coordinates onto a set of symmetry basis functions developed by Behler and Parinello [10].

The 2 classes of symmetry functions are the radial and angular symmetry functions [11]. Here radial symmetery functions will take on the following notation:

$$G_i^r(\sigma_i) = \sum_{j \neq i}^{neighbours} g^r(R_{ij}) \quad where \quad R_{ij} = \left| R_j - R_i \right| \tag{2.68}$$

And for angular functions:

$$G_i^a(\sigma_i) = \sum_{k \neq j \neq i}^{neighbours} g^a(\theta_{ijk}) \quad where \quad \theta_{ijk} = \angle(R_j - R_i, R_k - R_i) \tag{2.69}$$

Radial symmetry functions are a set of basis functions, centered at atom i, that model the radial distribution of neighbours within the cutoff radius R_c. A simple example of a radial symmetry function is:

$$G_i^1 = \sum_{j=1}^{N_{atom}} f_c(R_{ij}) \tag{2.70}$$

Where the cutoff function may be given by:

$$f_c(R_{ij}) = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_c} \right) + 1 \right] & R_{ij} \leq R_c \\ 0.0 & R_{ij} > R_c \end{cases} \tag{2.71}$$

This is the radial symmetry function around atom i. R_{ij} represents the distance between atom i and neighbouring atoms j. In this case, the cutoff radius is set by the user and assumed to be a set distance away from the central atom. Since equation 2.70 will assume the same cutoff radius for all neighbours, there

would need to be multiple sets of G_i^1 functions with different cutoff radii, if different atomic species are interacting with atom i in the atomic environment. For example, consider a case where the atomic environment is centered around atom A, and atom A is bonded to atom B and atom C (where atoms B and C are different species). If the same cutoff radius is used for atom B and C, but atom C interacts with atom A at distances larger than the cutoff radius, the description of the interaction potential would artificially be zero where it should not be. Instead, one could superimpose a set of basis functions, some of which describe the environment of B within its own cutoff, and some which describe the environment of C with larger cutoff radius. In any case, it is still a good idea to ensure that cutoff radii are not too short to accurately model the atomic environment. Alternatively, one could multiply by an exponential

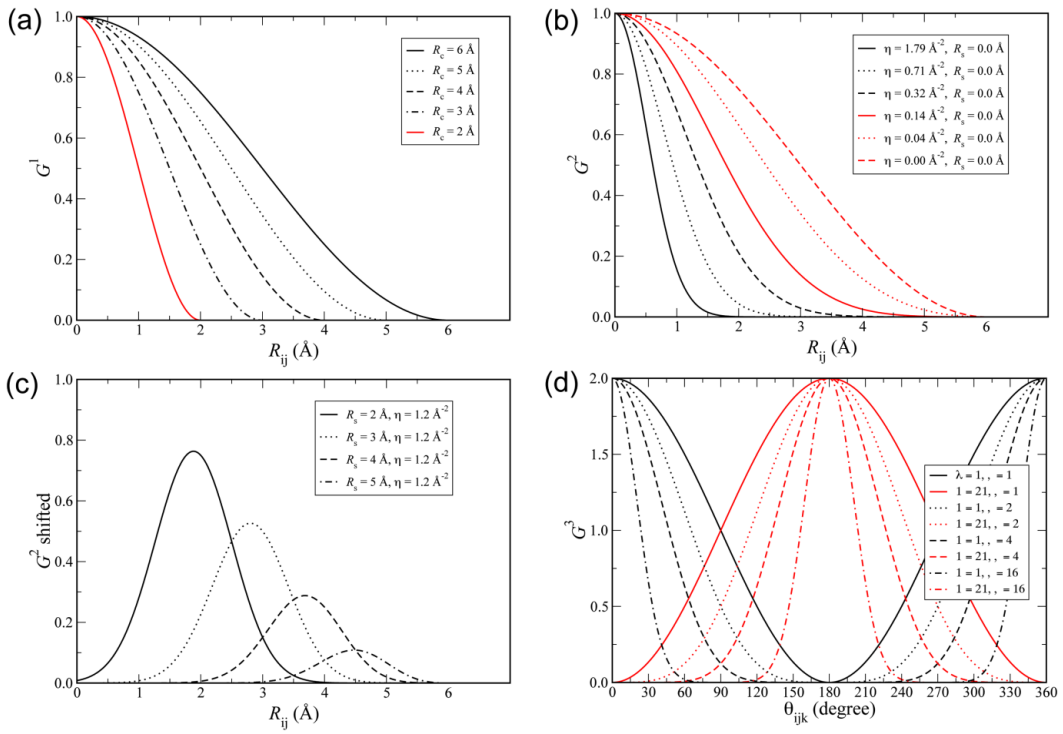

FIGURE 2.10: a) Shows a simple symmetry function using only the cutoff functions. Various values for cutoff radii are used to control how far from the central atom the atomic environment extends. b) Shows the change in basis function by varying the η parameter. c) R_s is varied. This shifts the gaussian to different radii. d) Angular symmetry functions. Image retrieved from [11].

function $\exp^{-\eta(R_{ij}-R_s)^2}$ to control the decay of the interaction within one user

defined cutoff radius:

$$G_i^2 = \sum_{j=1}^{N_{atom}} \exp^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}) \tag{2.72}$$

In this situation, only the η parameter is varied to change the radial extension of the symmetry function. This allows for greater control over where the functions should decay since there is only one hard cutoff at the boundary of the atomic environment. These functions are plotted in figure 2.10 where values of R_c, η and R_s are varied in part (a), (b) and (c) respectively.

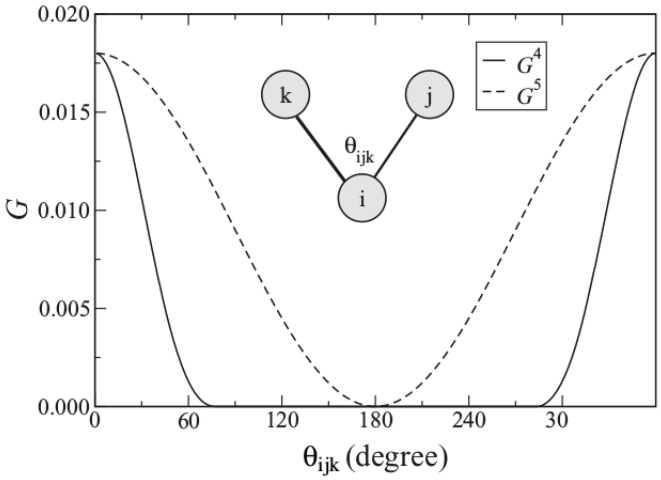

FIGURE 2.11: Angular symmetry functions formed between atom i, j, and k. Image retrieved from [11]

The angular symmetry functions, are functions that are formed between atom i and its two neighbouring atoms (j and k) such that they form an angle θ_{ijk} as seen in 2.11. The cosine of θ_{ijk} can be used to model the angular symmetry functions since the potential is periodic with respect to the angle. The angular functions are then multiplied by Gaussians for all three interatomic distances between the 3 atoms to ensure the function approaches zero if any of the distance between atom pairs becomes larger than the cutoff radius [11]:

$$G_i^3 = 2^{(1-\zeta)} \sum_{j \neq i} \sum_{k \neq i,j} \left[\left(1 + \lambda \cdot \cos(\theta_{ijk})^{\zeta} \cdot \exp^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{jk}) \cdot f_c(R_{jk}) \right) \right] \tag{2.73}$$

where $2^{(1-\zeta)}$ is a normalization factor and λ is a parameter with values varying from -1 to 1. The λ parameter can invert the cosine function.

Another option for angular symmetry function is:

$$G_i^4 = 2^{(1-\zeta)} \sum_{j \neq i} \sum_{k \neq i,j} \left[\left(1 + \lambda \cdot \cos(\theta_{ijk})^{\zeta} \cdot \exp^{-\eta(R_{ij}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{jk}) \right) \right] \tag{2.74}$$

In this case the Gaussian and cutoff function between atoms j and k are removed from the G_i^3 function. By doing this the G_i^4 function is larger than the G_i^3 function since the terms that were removed had a magnitude less than 1. The function also has access to a larger set of angles since the maximum distance between j and k inside the cutoff sphere is $2 \cdot R_c$[11].

Once the Cartesian coordinates are transformed using the symmetry functions, they can be used as an input for a neural network. The transformed coordinates describe all of the atomic environments in a reference structure. Given an atomic structural environment a subnet will learn an atomic energy contribution. When all of the energy contributions are learned from each atomic environment, they are summed to produce a total energy for a structure. This may be different from a typical architecture due to the fact that each input for the main network feeds into a subnetwork.

2.2.8 Atomic Energy Network (ænet)

ænet is an open source implementation of the Behler-Parinello approach. First the atomic fingerprints are generated by projecting the Cartesian coordinates onto a set of symmetry functions. Structural fingerprint files must be included for each atomic species in the dataset. These files include various parameters to fit the set of basis functions. Since symmetry functions are evaluated for each combination of atomic species (eg. A-A, B-B, A-B for binary combinations), the

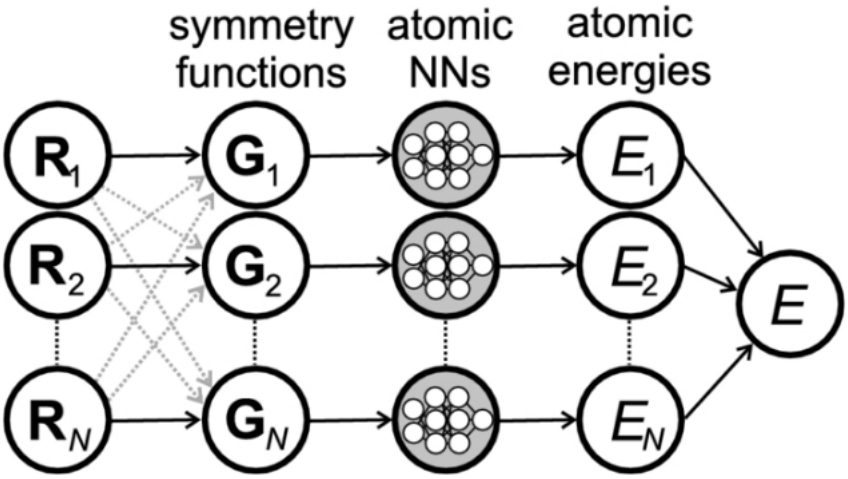

FIGURE 2.12: Behler-Parinello network architecture. Here inputs are mapped onto a set of symmetry functions. The symmetry functions feed into subnets which each output an atomic energy contribution. Atomic energies are summed in the output layer to generate the total energy. Image retrieved from [9]

number of basis functions scale with the number of species [3].

Usage:

1. **generate.x:** To use ænet, one should construct a set of reference data. The input files are given in .xsf format, which includes atomic positions, forces and lattice vectors. The lattice vectors define how to translate the basis to repeat the crystal structure periodically.

   The next step is to define the symmetry functions and symmetry function parameters for each atom type in *structural fingerprinting* files. The structural fingerprint defines the basis setups for each atom type. Each atom has its own setup to ensure that atom types can be distinguished from one another.

   In the main input file (generate.in), the user inputs the atomic energies of all atomic species, and links all reference data and fingerprinting files. When generate.x is executed, the reference data is projected onto an invariant basis. A binary file containing the transformed coordinates is output.

2. **train.x:** The next step is to train a network on the transformed coordinates. The user can define the network architecture, choice of symmetry functions, optimization schemes, and testing fractions. At the end of each step, an ANN potential is output.

3. **predict.x**: lastly, the ANN potential can be used to predict atomic energies. Here a set of unseen structures should be used to test how well the ANN potential can predict the energy. This code will predict the energies and forces on atoms.

Chapter 3

Methods

3.1 Data generation

All materials were constructed using the Materials Project [42, 68] which is an extensive database of material structures and their properties. The structures obtained were relaxed and used as inputs for MD using DFTB+ [1]. Three different parameter sets were used to describe the interatomic potentials: mio-1-1 [25], matsci-0-3 [54] and pbc-0-3 [44]. For all 2D materials and their 3D analogues, NVT simulations were implemented at $T = \{300K, 500K, 750K, 1000K\}$.

3.1.1 2D and 3D systems

To generate a dataset for 2D Silica, bulk SiO_2, graphene, diamond, hexagonal boron nitride (hBN) and cubic boron nitride (cBN), structures were sampled from their respective MD trajectories. A simulation time step of 0.273 fs was used to be consistent with Ryczko et al. [75]. MD trajectories were sampled every 273 fs to generate reference data. There were 108 atoms in the SiO_2 systems, and 128 atoms in the graphene, diamond, hBN and cBN systems. The Brillouin zone was sampled by a 2.0x2.0x1.0 K-Point mesh centered around gamma point 1.0 [1]. For some of the runs, the data obtained from temperature sampling was augmented by scaling the lattice vectors and coordinates by +/- 10% around the original structures.

[1] The gamma point is the center of the Brillouin zone.

3.1.2 TiO_2

The ænet software was released with an example TiO_2 dataset, which was used for benchmarking. This data set consists of 5 polymorphs (rutile, anatase, brookite, columbite and baddeleyite) [3] which were obtained by distorting ideal rutile, anatase and brookite and running DFT calculations and MD trajectories. To distort the structures, the lattice constants were scaled +/-10% around ground state, the crystals were gradually tilted by applying volume-conserving monoclinic strain and the structures were compressed in one crystal dimension [3]. Supercell structures with oxygen vacancies were added to this augmented dataset. Also, structures were iteratively added to the reference data from short MD trajectories at various temperatures (not specified by the authors) using a preliminary NNP.

3.1.3 Run Summary

Material	Ensemble	Parameter Set	Temperatures
2D Silica/Bulk SiO_2	NVT	pbc-0-3 [44]	300K, 500K, 750K, 1000K
hBN/cBN	NVT	matsci-0-3 [54]	300K, 500K, 750K, 1000K
Graphene/Diamond	NVT	mio-1-1 [25]	300K, 500K, 750K, 1000K

TABLE 3.1: Above are basic simulation info for each system.

Material	#atoms/#configs	# atomic environments	MD Sampling Frequency
2D Silica/Bulk SiO_2	$\frac{108 \text{ atoms}}{2000 \text{ configs}}$	216000	273 fs
hBN/cBN	$\frac{128 \text{ atoms}}{2000 \text{ configs}}$	256000	273 fs
Graphene/Diamond	$\frac{128 \text{ atoms}}{2000 \text{ configs}}$	256000	273 fs

TABLE 3.2: Above is info for sampling from each MD trajectory

3.2 Workflow

The general workflow for constructing reference data and obtaining NNPs is shown in figure 3.1. To generate input data, DFTB+ was used to run MD. The initial geometries were obtained from the Materials Project [42]. Structures were

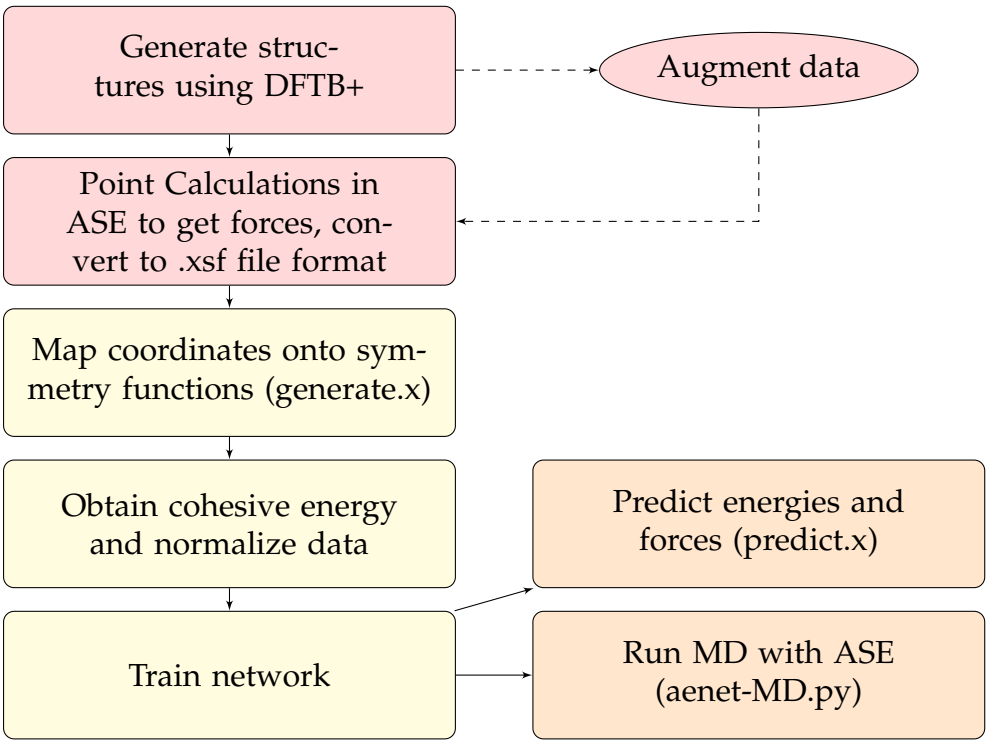

FIGURE 3.1: Workflow established for using ænet.

sampled from the MD trajectory to be included in the reference data. At this point, there is an option of augmenting the data. Data augmentation is further discussed in Section 4. Since DFTB+ does not print out the forces during MD until the last step, a point calculation is taken using the DFTB calculator in Atomic Simulation Environment[2] (ASE) [48]. The structures then need to be converted to .xsf file format for use in aenet.

Once the reference dataset is ready, the coordinates must be mapped onto a set of symmetry functions. The symmetry function parameters and type are specified in *atomic fingerprint* files. These files specify the setup for the basis functions used for each atomic environment. Each atomic species has it's own structural fingerprint, but for the sake of simplicity, in all runs included in this work the symmetry functions and parameters are identical between atomic species.

Since ænet trains on cohesive energies, the atomic energy (energy carried by

[2]ASE is a collection of tools in python that are used for setting up, modifying, running and visualizing atomistic simulations. Various calculators are available to interface ASE with different codes.

an atom) of each atomic species must be included. This is done by calculating the energy of a single atom in a vacuum using DFTB+. The cohesive energy is the difference between the total energy in the system and the sum of all atomic energies. Training on the cohesive energy makes it easier to train as there is a smaller spread of energies. Since this is the primary reason the network trains on cohesive energy, it is not entirely necessary to use the actual atomic energy value. By using another value however, the training may proceed more slowly and we can no longer interpret the network as training on cohesive energies.

Data is normalized using the method described by Montavon et al [64]. The neural network architecture and optimizer is then specified, and the nerual network trains until specified to stop. Neural network potentials are printed out at every epoch (one pass where the network has seen all samples in a dataset), so early stopping is possible. Once a NNP is obtained, it is ready for use in MD, or it can be used to predict energies and forces from a list of structures directly. The ænetLib library contains routines to parse the ANN potential files for energy and force evaluation. An implementation of the ASE calculator is linked to this library so that the forces and potentials can be evaluated during MD.

3.3 Performance Metrics

Loss Curves

At each training iteration the mean absolute error (MAE) is taken between the cohesive atomic energies and the target energies. The MAE is plotted on a log scale to enhance the differences at very low error values. The loss is plotted over the number of epochs. The loss curve will typically decrease until the network starts to overfit. At this point, the testing error will start to diverge away from the training error. In general, only the testing error will be plotted on the loss curve, as it indicates how well the network is predicting against unseen structures. The target error used throughout this work is 5 [meV/atom] to be consistent with Artrith et al. [3]. This is about an order of magnitude smaller than the thermal energy of the system (kT=25.85 [meV]-86.17 [meV]).

Predicted vs. True

To test how a NNP can predict energy and forces on structures outside the training and testing data, the force and energy predictions of a separate dataset are taken. To see the similarity of these predicted values to the target values, the predicted and true energies are plotted against one another, where a one-to-one mapping is a perfect fit.

Pearson Correlation Coefficient

To describe how well the target quantities are being predicted by the network, the Pearson correlation coefficient is computed. The Pearson correlation coefficient is a measure of linear correlation between two variables. The values of the coefficient range between -1, 0 and 1, where -1 is total negative correlation, 0 is no correlation, and 1 is total positive correlation. The Pearson correlation coefficient for a sample is given by:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{n\sum_{i=1}^{n}(x_i - \sum x_i)^2}\sqrt{n\sum_{i=1}^{n}(y_i - \sum y_i)^2}} \tag{3.1}$$

where n is the size of the sample, x_i and y_i are points from each dataset, $\overline{x}$ and $\overline{y}$ are the mean of each dataset.

Squared Error

In some cases, to obtain a clearer idea of how well the network is predicting on a given set of structures, the squared error is taken between the predicted and target quantity.

3.4 Convergence Studies

3.4.1 Cutoff Radius

The cutoff radius defines how many atoms are included in each atomic environment. If the cutoff radius is too small, interactions that occur at larger distances go unaccounted for, resulting in a less accurate potential. Behler et al.

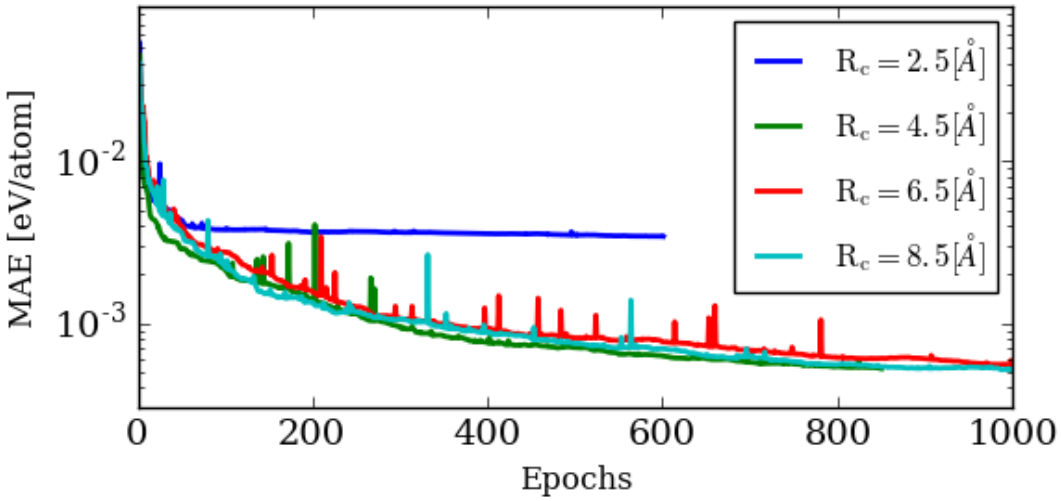

FIGURE 3.2: Cutoff radius convergence for the 2D silica dataset. A 70-10-10-1 tanh network architecture is optimized with l-BFGS

[11], recommend using a cutoff radius between 6 and 10 Å. Using a large cutoff radius increases the computational demand, as more atoms are included within the cutoff sphere. In the following test, the convergence of cutoff radii are tested for the non-augmented 2D Silica system. 4 cutoff radii were used, $R_c \in \{2.5\text{ Å}, 4.5\text{ Å}, 6.5\text{ Å}, 8.5\text{ Å}\}$, with a 70-10-10-1 network architecture (optimized using l-BFGS). The peaks in the loss curve can be attributed to overshooting an acceptable step size during optimization (the step size is approximated by performing a one-dimensional optimization at each iteration).

The smallest cutoff radius tested $R_c = 2.5$ Å converged to the highest error, indicating that it is too small to capture relevant interactions. A cutoff radius of as low as $R_c = 4.5$ Å appeared to be sufficient to capture all interactions. The loss converged to roughly the same values for anything greater than $R_c = 4.5$ Å

3.4.2 Training Set Size

Determining a sufficient training set size can optimize the computational efficiency of the neural network. More training examples in the reference data can result in a network that takes longer to train. Of course, the size of the training set depends on the how varied the reference data is. A network training

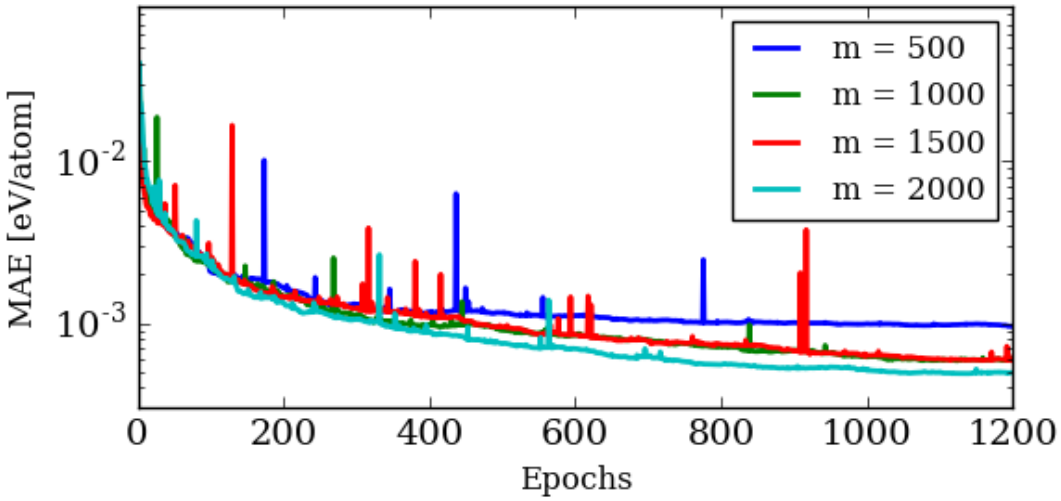

FIGURE 3.3: Training set convergence for the 2D silica dataset. A 70-10-10-1 tanh network architecture is optimized with the l-BFGS optimizer. Reference datasets have sizes of $m \in 500, 1000, 1500, 2000$. From this reference data, 85% is reserved for the training set, and 15% is reserved for a testing set.

on 2000 examples of very similar structures will train much better than a network training on 2000 examples of diverse structures that give rise to a larger spread in energies. The network training on similar structures, however, will be less general than the network training on a diverse set of structures. In this test, a 70-10-10-1 architecture network was optimized using l-BFGS. The non-augmented 2D Silica data is used for the reference dataset.

In this test, the loss decreases as the number of training examples increase. Using a reference dataset of 500 structures resulted in an error that was much higher relative to the others. In the following tests, a reference dataset of 2000 structures will be used since it converged to the lowest error.

3.4.3 Time Between Snapshots

Too much correlation within the dataset can lead to over-fitting in the network. To generate data, snapshots from various MD trajectories were uniformly sampled. In this situation reference data that has too much correlation may result in the neural network learning the specific MD trajectory, and not being able to generalize to others. In this experiment, the time between snapshots sampled

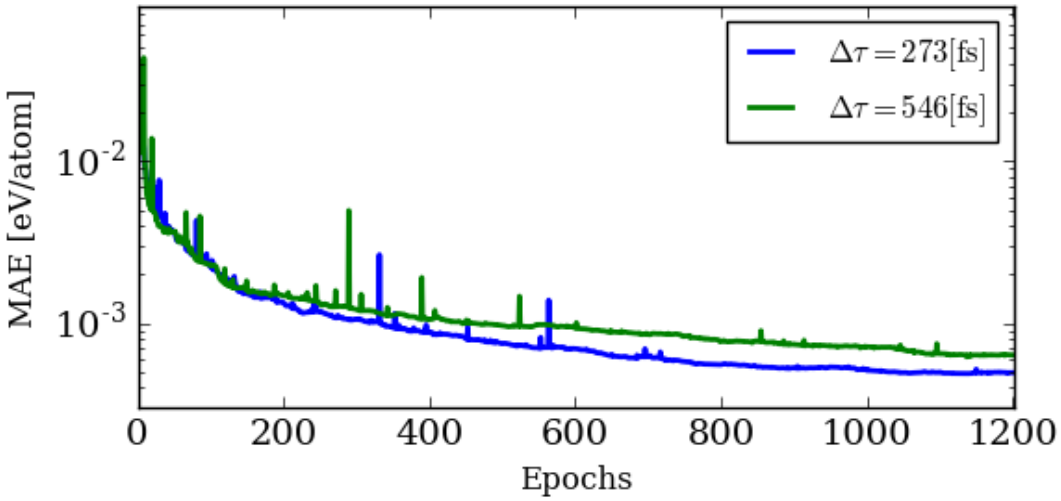

FIGURE 3.4: Convergence of networks using different sampling windows. A 70-10-10-1 tanh network architecture is optimized using the l-BFGS method.

from the trajectory was doubled from $\Delta\tau = 273$ fs to $\Delta\tau = 546$ fs. There was a slight increase in error for the structures sampled less frequently from the trajectory. This implies that there is some correlation in the data sampled every 273 fs.

3.4.4 Optimizers and Sensitivity to Random Seeds

For the augmented graphene reference dataset, the 3 optimizers available in ænet were used to optimize the network multiple times to determine the sensitivity to initial random seeds. A 26-10-10-1 network architecture with tanh activation functions was trained over 5 different random seeds. As discussed by Artrith et al. [3], in general the online gradient descent optimizer is the least expensive to evaluate, the Levenberg-Marquardt optimizer is efficient for small datasets but expensive for large datasets, and the l-BFGS optimizer is efficient for large datasets as it parallelizes well. Both the online gradient descent and Levenberg-Marquardt methods require the definition of a learning rate, whereas the l-BFGS method does not.

The l-BFGS optimizer was found to converge consistently to a similar error

between runs (figure 3.5). The l-BFGS optimizer also does not require optimizing the learning rate parameter so it is ideal for running a large number of experiments.

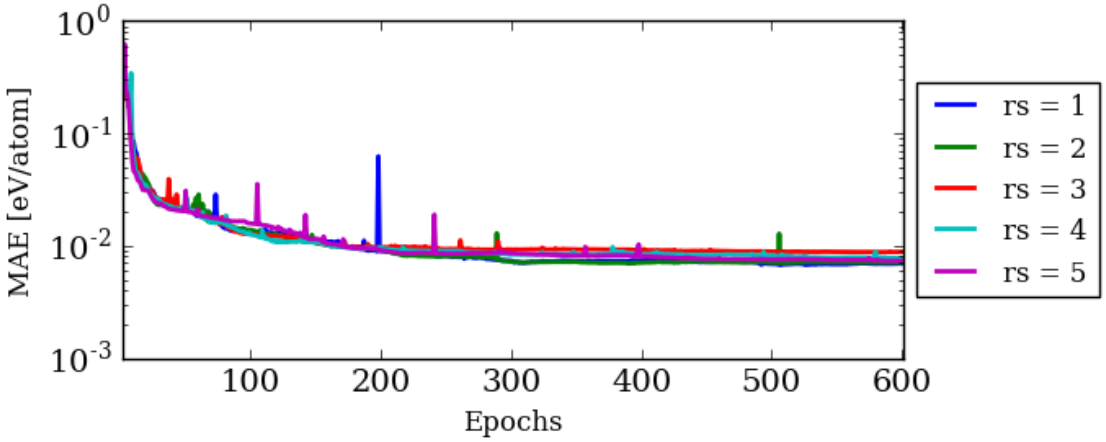

FIGURE 3.5: Convergence of the l-BFGS optimizer over 5 random seeds. A 26-10-10-1 tanh network architecture is optimized using the graphene dataset.

The LM optimizer was implemented with an initial learning rate of 0.1, 3 iterations per optimization to adjust the learning rate, a factor to adjust the learning rate of 5.0, and a convergence threshold of 0.001. The batch size (number of training points used to evaluate the error function in an iteration) was set to the maximum number of training points. Compared to l-BFGS the LM network had significantly more variation with respect to random seed. In figure 3.6, random seed 1 is shown to converge to a local minimum, and then get kicked out of the minimum to a lower minimum at 400 epochs. The second random seed however converged to a local minimum and then started diverging away after about 425 epochs. This convergence behaviour is less stable than the l-BFGS method, but it is possible that over many random seeds that it will converge to a lower error than l-BFGS.

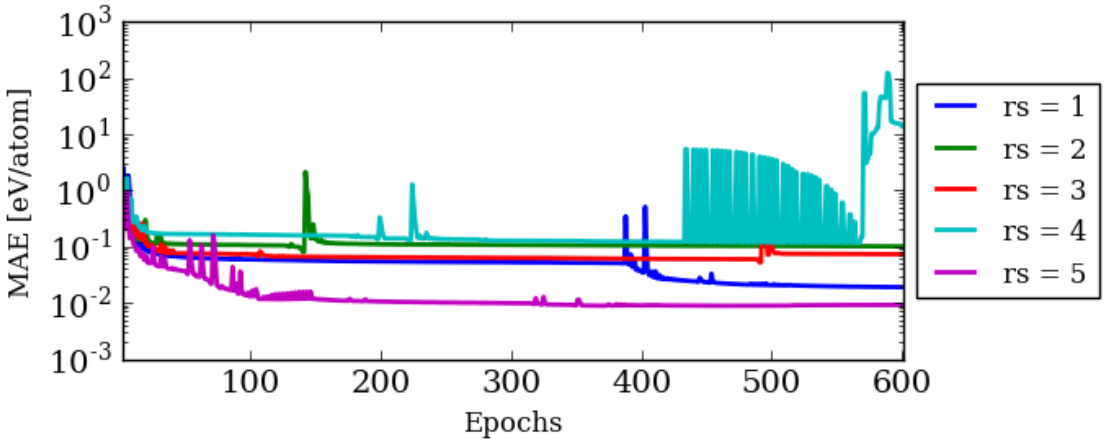

FIGURE 3.6: Convergence of the Levenberg-Marquardt optimizer over 5 random seeds. A 26-10-10-1 tanh network architecture is optimized using the graphene dataset.

For the online gradient descent optimizer, the learning rate was chosen to be $\gamma = 5.0E-7$. This controls how much the weights are adjusted with respect to the gradient of the loss function. The momentum parameter is used to control fluctuations. In this run it was set to a value of $\alpha = 0.25$. The online gradient descent optimizer consistently performed less well than the l-BFGS and LM optimizers (figure 3.7), which is a result that is consistent with the optimization of the TiO_2 example dataset provided by ænet [3].

While the LM optimizer was able to converge to a lower error than the l-BFGS optimizer given multiple runs, it was shown to have less stable convergence behaviour. To achieve a low error using the LM or gradient descent optimizers, the learning rate must be optimized. Choosing a learning rate that is too high can lead to overshooting during optimization. This can lead to the network not converging or diverging. Choosing a learning rate that is too small can lead to a more reliable optimization, but it comes at a high computational cost. It would take more experiments for the LM and gradient descent optimizer to adjust the learning rate parameter as well control for the sensitivity to random seeds. Therefore, for most experiments, the l-BFGS optimizer is used. This choice in optimizer allows for more reliable comparisons. The LM optimizer is generally only implemented once a network is shown to converge to a high error, and it may be helpful to use LM to achieve lower convergence.

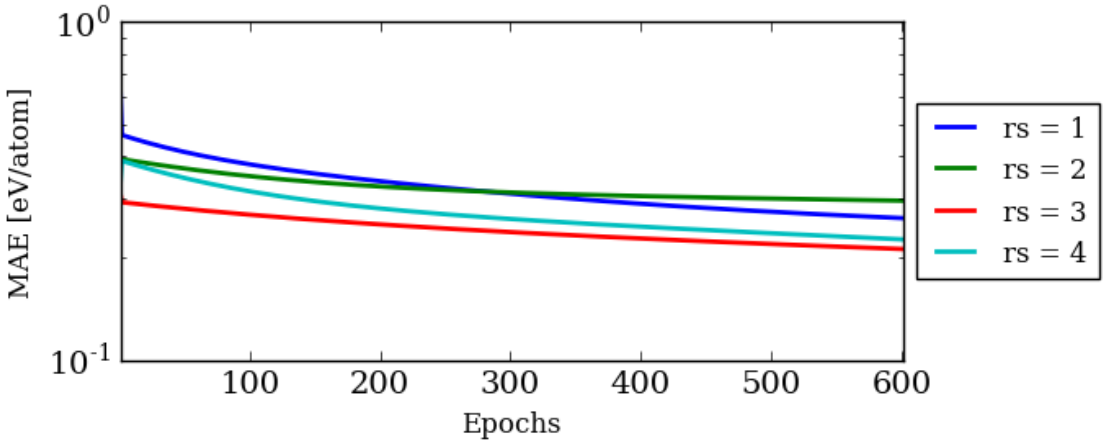

FIGURE 3.7: Convergence of the Online Gradient Descent optimizer over 5 random seeds. A 26-10-10-1 tanh network architecture is optimized using the graphene dataset.

Chapter 4

Network Activations

Activation functions play the role of mapping input signals on to a new domain. Nonlinear activation functions are generally required for a network to learn a nonlinear mapping because a network with only linear activation functions is equivalent to linear regression. In this chapter, we will test the performance of various activation functions available in ænet. We will see that networks using only linear activation functions are easier to train, and under some circumstances sufficient. Lastly, we will look the role of data augmentation in the complexity of the problem we aim to solve.

4.1 2D Silica

Recall that the 2D Silica dataset was generated by combining structures from 4 NVT simulations run at T=300K, 500K, 750K, and 1000K. In total 2000 structures (500 from each MD trajectory) were included in the reference data set. This dataset was split into 85% training data and 15% testing data. For all the results obtained in section 4.1, the l-BFGS optimizer and the backpropagation method were used to minimize the cost function. For all loss curves, only the testing error is plotted, as it is a better indication of how the network is able to predict energies for structures it has not seen yet.

In this section, we will discuss the performance of neural networks using each activation function on the 2D Silica data set. We will explain using a toy model why the linear regression network performs, at first glance, surprisingly well on this dataset. We will also investigate where a linear regression model has limited prediction capabilities, and where a nonlinear network becomes advantageous to use.

4.1.1 Performance of Activations Available in ænet

To determine which activation function works best for 2D silica, a 70-10-10-1 (8 radial and 62 angular basis function input) network architecture was trained for 600 epochs. Networks using each activation function available in ænet (linear, sigmoid, tanh or twist activation functions) were compared (figure 4.1). Note that each network used only one of each type of activation function eg. a tanh network would have tanh activations in all hidden units of the network. The only exception to this is that in the final layer a linear activation function is always used since a larger spread of energies can be mapped to the outputs (they are not confined to (-1,1) or (0,1)).

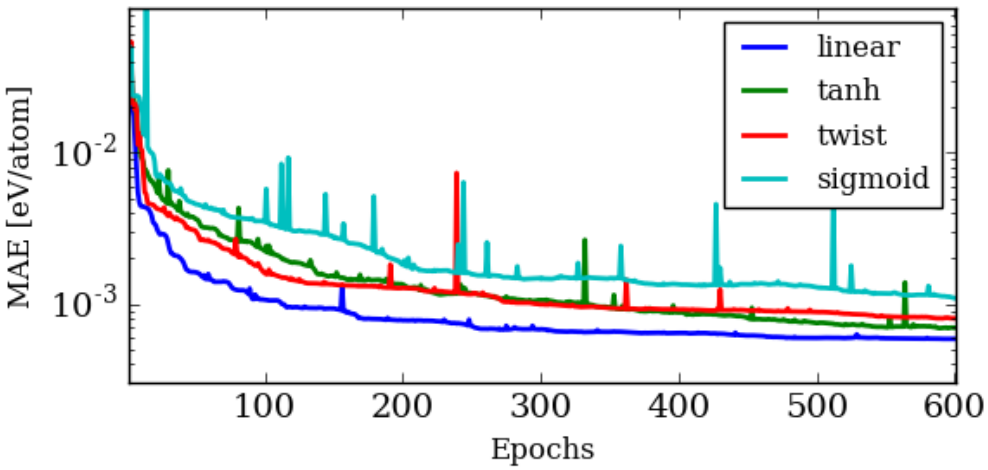

FIGURE 4.1: 70-10-10-1 network optimized with l-BFGS using linear, sigmoid, tanh, and tanh with linear twisting activation functions. The 2D silica dataset is used.

Sigmoid (logistic) function

In figure 4.1, we see that, not surprisingly, the sigmoid network was the slowest to converge and converged to a higher error than networks using other activations. As mentioned previously, the derivative of the sigmoid function is small compared to tanh, and this causes the weight updates to be very small. As mentioned in other works [35, 34], the sigmoid function tends to lead to neurons saturating at the tails for any large positive or negative input values. The sigmoid function also has a non-zero mean, which can lead to the network optimization requiring a larger number of iterations. As discussed in reference

[49], since the sigmoid function always produces a positive output, all of the weight updates have the same sign during backpropagation. The weights are always increasing or decreasing as a group during weight updates. This leads to a zigzagging in the optimization that is slow and inefficient. It should also be noted that if the inputs to the network are too small and close to zero, it may result in small weight updates (slowing optimization). It is ideal to ensure that weights are initialized to values that are not too small. Otherwise, they can shrink inputs even more and slow weight optimization.

Hyperbolic tangent (tanh) function
Along with the twist function, the tanh function performs second best to the linear activation function at 600 epochs, but still appears to be converging. The tanh function converges much more slowly than the linear function. The tanh function has two regions where small derivatives near the tails lead to small weight updates. This situation occurs when the inputs are largely positive or negative. In figure 4.2, the tanh network is run for longer and found to converge to a slightly lower value than the linear function. We will return to this after discussing the results for the remaining activation functions.

Tanh with linear twisting (twist) function
The twist (tanh with linear twisting) function is a scaled tanh function [50] with a small linear term added. It is meant to be more efficient to evaluate than tanh [49] and help avoid the flatter regions near the tails of tanh. These regions have small derivatives causing the weights to be updated very slowly during backpropagation if the inputs are largely negative or positive. In the twist function, the linear term added to the tanh is intended to circumvent this issue. In this experiment, however, we did not find that the twist function performed significantly differently than the tanh function.

Linear (identity) function
Perhaps the most interesting result is that the network of linear activation functions converged faster than networks using nonlinear activation functions and also led to the lowest error. Using a linear activation not only accelerates the time spent training a neural network, but it also results in a network function

that is quicker to evaluate than networks activated by nonlinear functions. After compressing the network to a linear regression model it would require more floating point operations (FLOPs) to compute the nonlinear transformation of inputs than just a weighted sum output from a nonlinear network.

To show that a network made up of linear activations compresses to a single layer, let us compress a network of 3 layers. Say that the network has inputs $X^{(1)}$, biases $b^{(l)}$ and linear activation function, $f_a^{(l)}(x) = x$, in layer l. The output of the network can be expressed as a nested function, since the activation outputs are equivalently the inputs for the following layer. Given a linear activation, $f_a^l(W^{(l)}X^{(l)}) = (W^{(l)}X^{(l)})$, the network function for a 3 layer network is expressed as:

$$\begin{aligned} a^3 &= f_a^{(3)}(W^{(3)}f_a^{(2)}(W^{(2)}f_a^{(1)}(W^{(1)}X^{(1)} + b^{(1)}) + b^{(2)}) + b^{(3)}) \\ &= f_a^{(3)}(W^{(3)}f_a^{(2)}(W^{(2)}X^{(2)} + c^{(2)}) + b^{(3)}) \\ &= f_a^{(3)}(W^{(3)}X^{(3)} + c^{(3)}) \\ &= W^{(3)}X^{(3)} + c^{(3)} \end{aligned}$$

Where $X^{(2)} = W^{(1)}X^{(1)}$, $X^{(3)} = W^{(2)}X^{(2)}$ and $c^{(l)}$ is the addition of b constants from the previous layer with the current layer. Since the activation input is mapped directly to the output from the previous layer, the network collapses to a single layer. This layer is simply a linear combination of the inputs plus a constant (bias) term.

Training over more epochs it becomes apparent that the nonlinear tanh function can achieve an error that is 17% lower than the linear function (figure 4.2). It takes about twice as many epochs, however, to achieve convergence. It would still be interesting to determine whether the linear network could produce an NNP capable of predicting well on unseen structures.

To test the performance of the tanh and linear network, the NNP produced by each network was used to predict energies of 1000 structures that were not included in the training data. This independent data consisted of structures equally sampled from each MD trajectory at different temperatures. Note that these are the same MD trajectories from which the training set was sampled from. The independent data is sampled from structures taken from later portions of the trajectory. In figure 4.3, the energies predicted by the network on the

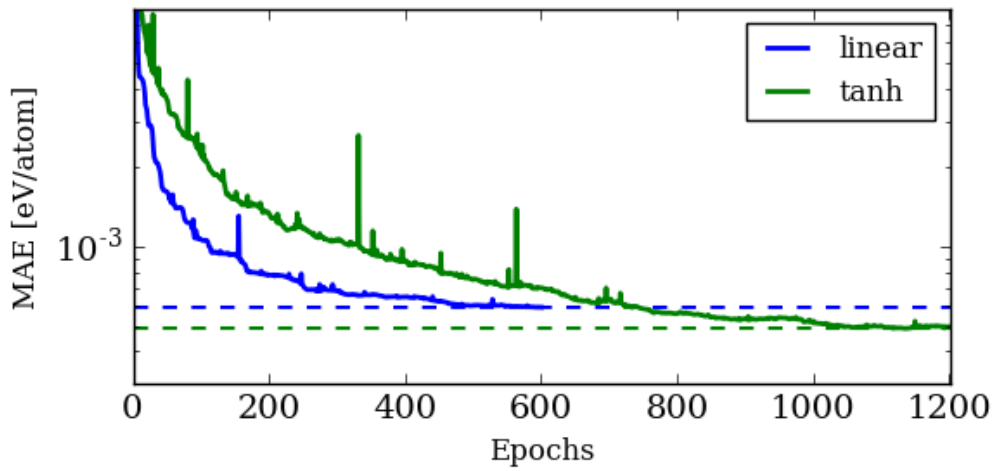

FIGURE 4.2: The network using a tanh activation was run for 600 more epochs. The error converges lower than the linear network, but takes twice as many epochs to converge.

independent testing data were plotted against the actual energies of the independent testing data. A perfect fit would mean that the predicted energies have a one-to-one mapping to the actual energies. In figure 4.3, it is apparent that the linearly activated network performs just as well as the network activated by tanh. This is an interesting result, because if only linear activation functions are needed to produce a predictive potential, the network function can be calculated more quickly. But what is really happening in the network?

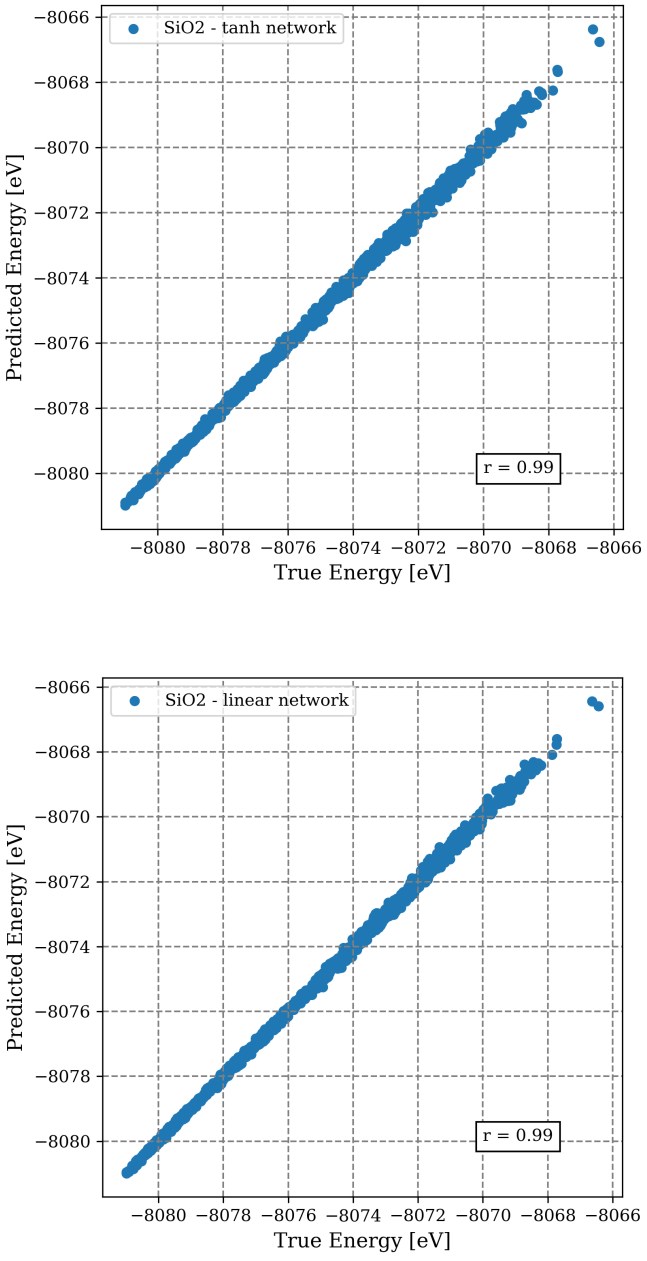

FIGURE 4.3: **Top:** Energies predicted by a 70-10-10-1 network using tanh activation functions are plotted against the true energy values. **Bottom:** Energies predicted by a 70-10-10-1 network using linear activation functions are plotted against the true energy values.

4.1.2 Toy model: Pair potential H_2

In the previous experiment, we discussed how this model, that is essentially linear regression, does a surprisingly good job at generating a well performing NNP. Typically, a neural network requires nonlinear activation functions to generate nonlinear mappings from input data to output data. In the ænet approach, the inputs are projected onto a set of symmetry basis functions prior to network optimization. A linear regression network in ænet is doing a linear combination of basis functions. This means that a linear network in ænet is actually capable of nonlinear mappings. Let us further examine the capabilities and performance of these linear networks.

In these experiments, we will use a toy model to better understand how linear and nonlinear networks perform in ænet. For simplicity, a pair potential of a neutral H_2 molecule was chosen. To generate training data, hydrogen atoms were manually placed at fixed distances apart and the corresponding energies were computed. The distances and corresponding energies were uniformly sampled from this set of configurations. Since the pair potential is a function of distance only between two particles, the system only has one degree of freedom. Therefore only the radial symmetry functions are required to map coordinates to potential energy surface. To generate data independent of the training and testing data, the energies were more densely sampled at various distances. The resulting training set size was 85, and the testing set size was 15. There were 500 samples in the independent testing set.

Performance for Linear and Tanh Networks with respect to Number of Radial Basis Functions

In this first test, we will look at how the number of radial basis functions affects the quality of the neural network fit. The number of radial basis functions is increased from $N = 1, 2, 3, 4, 8$ (where N is the number of radial basis functions) on a N-10-10-1 network architecture. Recall that the G^2 symmetry functions have the following functional form:

$$G^2 = \exp^{-\eta(R_{ij}-R_s)^2} f_c(R_{ij})$$

where $f_c(R_{ij})$ is the cutoff function, used to ensure a smooth cutoff at R_c.

$$f_c(R_{ij}) = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right] & R_{ij} \leq R_c \\ 0.0 & R_{ij} > R_c \end{cases}$$

The parameters used for η were sampled between $\eta = 0.003214$ to $\eta = 2.74972$ (the default range of η values). The R_s value was kept at zero, and an R_c cutoff of 6.500 Å was used. This test is implemented for both linear and tanh activation networks. The approximation of the pair potential by the linear and tanh networks are compared.

In figure 4.4, the true DFTB energies are plotted as a function of distance along with the neural network predictions of the energies at those distances. One may notice that at a distance of about 2.7 Å there is a jump in the energy. This artifact is the result of maintaining a fixed spin polarization during energy calculations for data generation. At smaller interatomic distances, the fixed spins come into close contact, and by Pauli exclusion principle they cannot occupy the same energy state. This results in fluctuations that would normally not be observed if the spins of the electrons were permitted to flip.

In this experiment, the number of radial basis functions are increased from 1 radial basis function to 4 radial basis functions. In figure 4.4, we see that the network with nonlinear activations outperforms the network with linear activations. We see that contrary to the expectation that the linear network will perform just as well as the tanh network, The linear network can not approximate the curve with fewer than 4 radial basis functions. However, even with only one RBF, the nonlinear network is able to learn the function to a very close approximation. Here we have changed the problem in two ways. Firstly, the number of basis functions have been decreased to a smaller number. Secondly, the target function contains highly nonlinear parts. The 2D silica dataset was sampled around equilibrium only. We will return to these points in section 4.2.2.

In figure 4.4, we see that there are no clear changes as the number of radial basis functions are increased from 1 to 4. Keep in mind that the network has an architecture of N-10-10-1 which provides more capacity for the network to learn nonlinear fits. In section 4.1.2, we will look at how decreasing the network capacity also decreases the accuracy of the fit for the nonlinear tanh network.

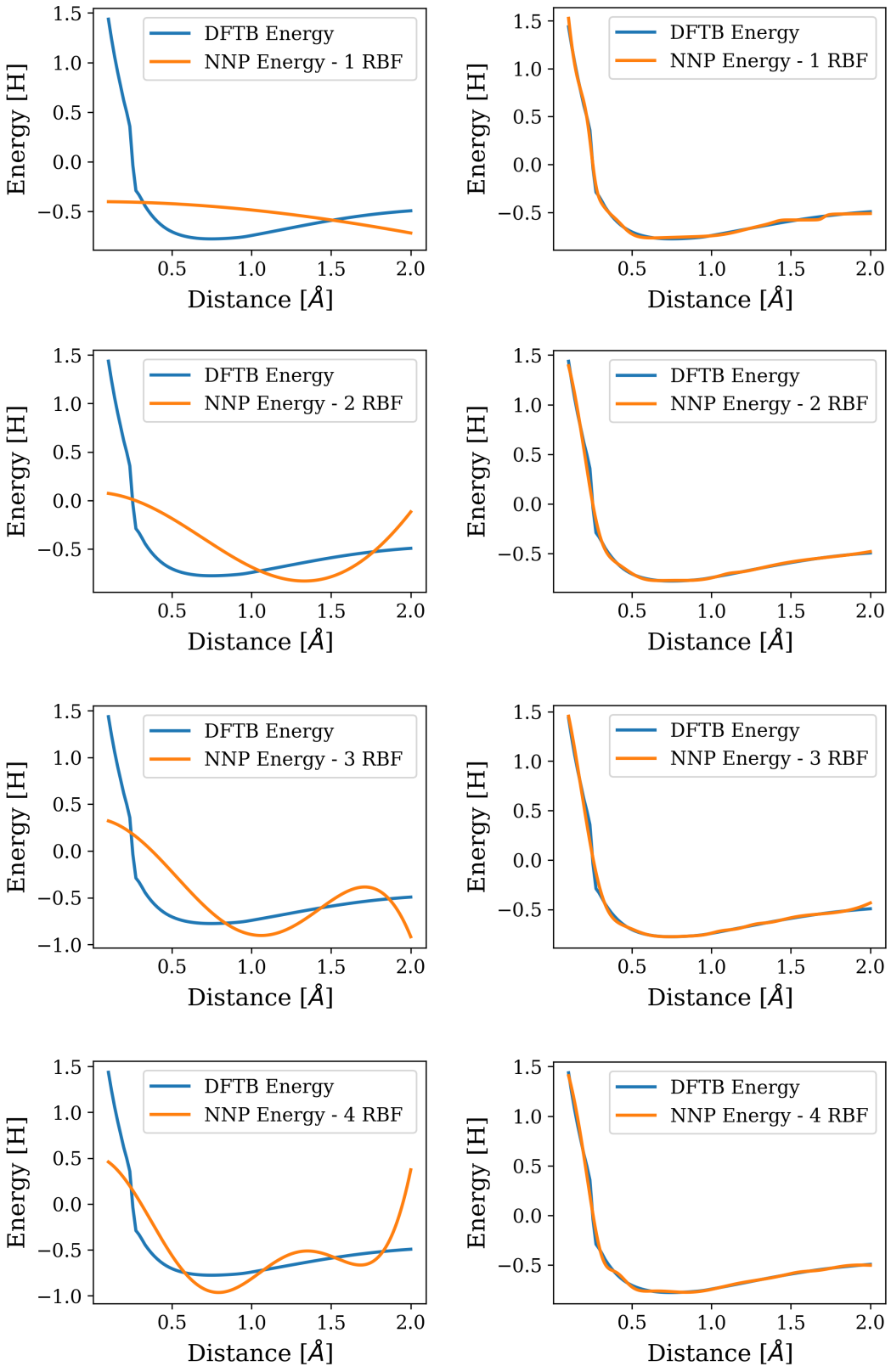

FIGURE 4.4: Energies predicted by the neural network are plotted with the true energies for each number of radial basis function. For the linearly activated network (left), increasing the number of RBFs for the linear network increases the capacity for fitting the nonlinear pair potential curve. The neural network with tanh activations (right) is capable of fitting the nonlinear activation function with only 1 RBF.

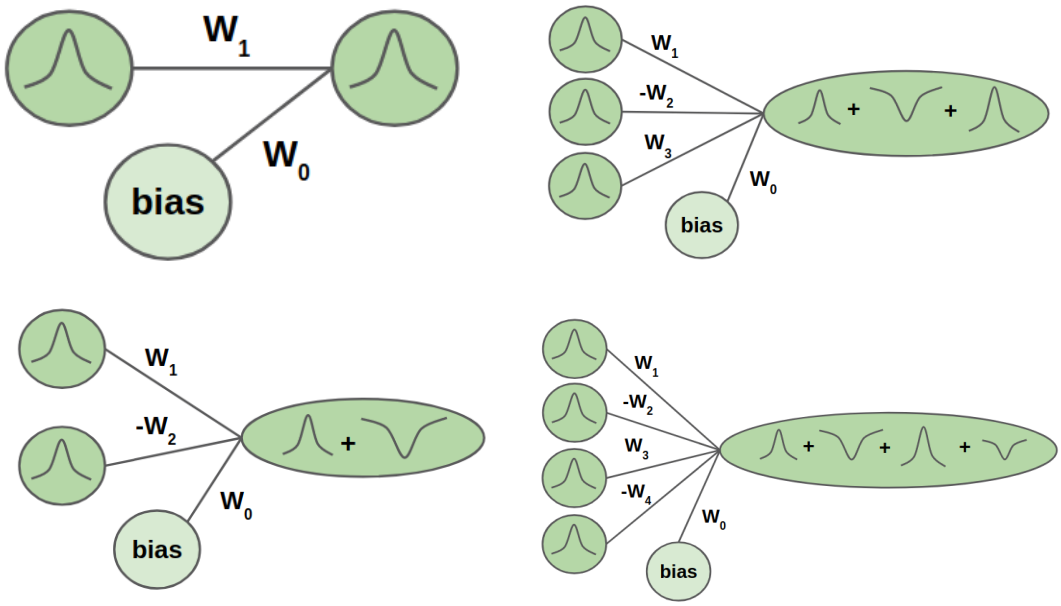

FIGURE 4.5: For the linear network, a linear transformation is applied to each neuron. Adding the gaussian-like functions together allows the network to learn a nonlinear function.

Figure 4.5, provides a graphical representation for what is happening within the linear network. The input nodes output a signal of a gaussian-like functions. The gaussian-like functions are scaled by the network weights, and if the weight is negative, it is flipped. Since the activation function is linear, the sum of the weighted inputs are mapped directly to themselves. So the weighted gaussian-like functions are simply added together.

Analyzing figure 4.4, we can see that for the network with linear activations, the number of radial basis functions determine how well it will be able to approximate a function. For one radial basis function, we see that the network can only predict a monotonically decreasing Gaussian-like function (recall that the Gaussian is scaled by a cosine cutoff function). This is because the output of the neural network is the Gaussian-like function scaled by the network weights and bias:

$$y^{out} = 2We^{-\eta(r_{ij})^2} + W_0 \tag{4.1}$$

where the linear combination is multiplied by 2, since the distance from atom 1 to atom 2 is equivalent to the distance from atom 2 to atom 1. In figure 4.5,

we can see that the network with 1 RBF can only be mapped with a positive or negative weight to the output. Therefore, the network can not learn anything more than an upright or inverted gaussian-like function.

When 2 RBFs are input into the network, it becomes possible to predict a non-monotonic function with one turning point. This is possible because one of the network weights is positive and the other is negative. The addition of two inputs with alternate signs for the weights is shown graphically in figure 4.5. If the weights were all positive, or all negative, the network would only be able to predict a monotonically increasing or decreasing function. As more RBFs are included into the network, the network becomes better able to approximate the nonlinear function.

The linear network with only radial basis functions as input is almost equivalent to an 'RBF network'. An RBF network architecture typically consists of one input layer, one hidden layer with RBF activations and one output layer with linear activations. The network function then looks like

$$y(x) = \sum_{i}^{N} w_i \phi(||x - c_i||) + b_i \tag{4.2}$$

Where w_i are the weights at neuron i, x is the input vector, c_i is the center vector at neuron i, b_i is the bias and ϕ is the radial basis function activation. An RBF network also learns a linear combination of radial basis functions. The only difference between this network and a linear network in ænet with inputs mapped onto radial basis functions is that the parameters in ænet are not optimized during learning. This may make it more difficult for a network to learn unless RBF parameters are densely sampled over a large range. The sensitivity of the network fit to η parameters is discussed in the following section (section 4.1.2).

RBF networks with one hidden layer are capable of being universal approximators on a compact subset $\mathbb{R}^N$ [70]. This suggests that in principle, if enough basis functions are used in the input layer in ænet, a linear network is sufficient to learn the PES of the training data. The decision to use more basis functions with linear activations or fewer basis functions with nonlinear activations is essentially choosing whether to use a wide (shallow[1]) or deep[2] neural network. In

[1] A shallow neural network has one hidden layer.
[2] A deep neural network has multiple hidden layers.

[60], Mhaskar et al discuss how deep networks can often approximate functions using significantly fewer parameters.

RBF Sensitivity to η Parameters

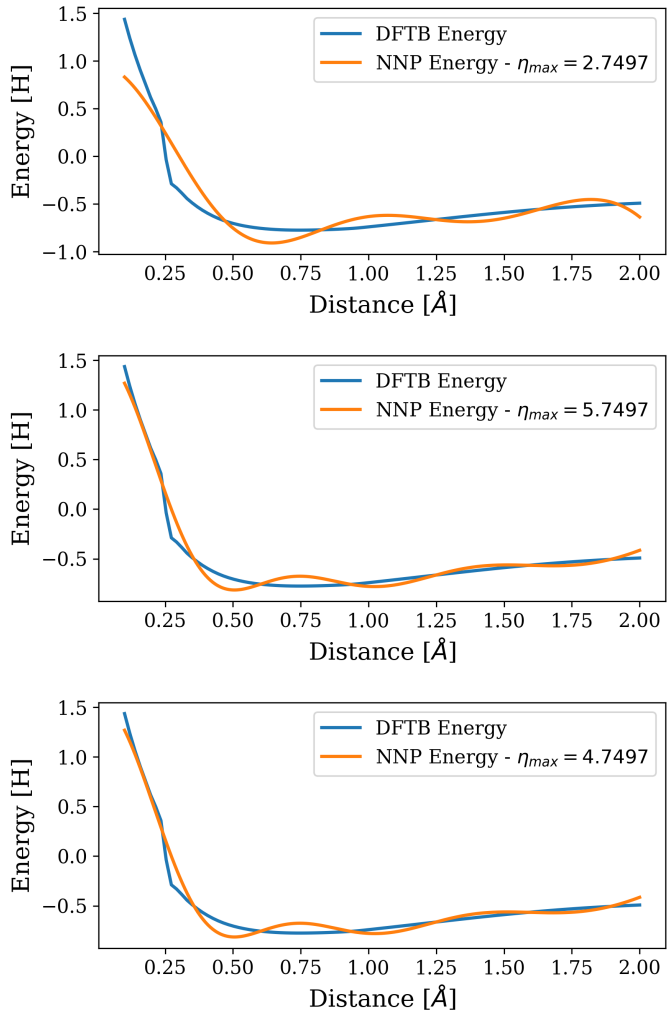

FIGURE 4.6: The network potential generated with different η values is used to predict the true DFTB energies for the pair-potential curve. Each network has a 30-20-20-1 network architecture and uses linear activation functions. Higher η values correspond to smaller interatomic separations. **Top:** η values up to $\eta_{max} = 2.7497$ are used. **Middle:** η values up to $\eta_{max} = 4.7497$ are used. **Bottom:** η values up to $\eta_{max} = 5.7497$ are used.

One significant difference between the RBF networks and the ænet model, is that the parameters for basis functions in ænet are predetermined. RBF networks learn basis function parameters as well as their coefficients. Recall that the width of the radial basis symmetry function used in these experiments is controlled by the η value. Larger η values correspond to compressed symmetry functions, and smaller η values correspond to stretched symmetry functions. To test how the range of η values effect the function approximation, 30 RBFs were tested for different ranges of η. Since the linear network does not approximate the curve well for small interatomic separations, larger η values were more densely sampled. In figure 4.6, a 30-20-20-1 network with linear activations was trained for varying η values. 3 curves were generated shifting the highest η values from $\eta_{max} = 2.7497$ up to $\eta_{max} = 5.7497$. Shifts to higher η values increase the sensitivity of the RBF around smaller interatomic distances.

As the η values increase, the network is better able to approximate the high energy regime. This highlights a limitation of treating ænet like an RBF network (for linearly activated networks). Since the parameters of the basis functions cannot be learned, a very large number of parameters over a wide range of values would be required to closely approximate the function. The linear network in ænet lacks the flexibility of a true RBF network.

NN Capacity Required to Learn NNPs

If nonlinear activations are used in the network instead, how much network capacity is required for a good fit? In section 4.1.2, the tanh networks consistently estimated the curve very closely. This is possibly due to overfitting, since the network capacity was 2 layers with 10 neurons each. In the test illustrated in figure 4.7, a 1 layer network of N nodes with 1 radial basis function input was tested, where the number of nodes are $N \in \{1, 2, 3, 4, 8\}$. A network with 1 RBF and 1 neuron did not have enough capacity to fit the curve. It did, however, get the general trend right (figure 4.7). Increasing to 2 neurons did not change the fit very much. It was not until a 3 neuron network was used that the network closely approximated the curve with small fluctuations. This indicates that even small tanh networks are sufficient to fit the pair potential curve.

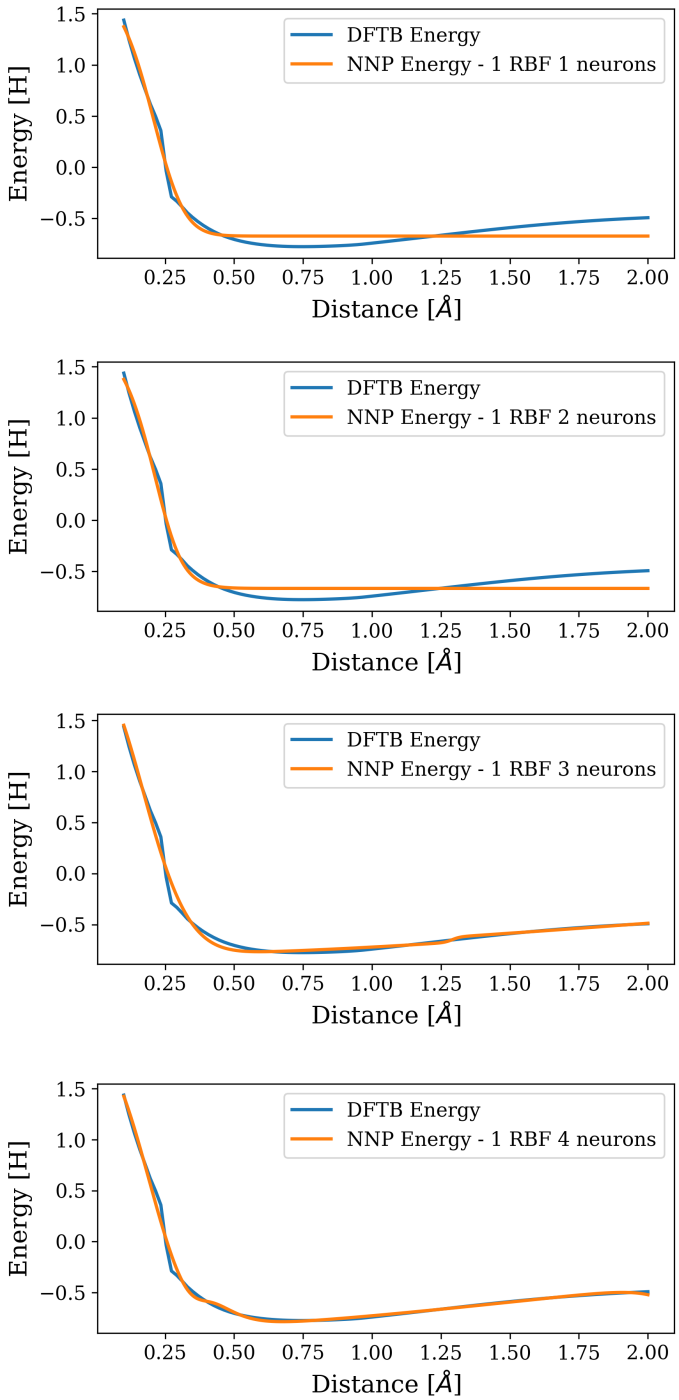

FIGURE 4.7: Network potentials trained on various architectures are used to predict the pair-potential curve. The number of neurons are varied from 1-4 in the hidden layer. **Top left:** 1-1-1 tanh network. **Top right:** 1-3-1 tanh network. **Bottom left:** 1-4-1 tanh network. **Bottom right:** 1-8-1 tanh network.

4.2 Graphene

To test how what we've learned on the simple pair potential generalizes, we will look at graphene. Recall that the graphene dataset has 500 structures sampled from MD trajectories at 4 different temperatures (300K, 500K, 750K, and 1000K). In this experiment, the neural network potential was generated using a N-10-10-1 (where N is the number of symmetry functions) linear or tanh network and optimized with the l-BFGS optimizer. In general 26 symmetry functions are used unless otherwise specified.

4.2.1 Role of Angular Basis Functions

To examine how the addition of angular basis functions affects the ability for the network to learn we will train a network in the presence and absence of angular basis functions. This will provide insight into how much of a role angular basis functions play in constructing the potential energy surface. Here, the number of radial basis functions are reduced to one. The number of angular basis functions will either be 18 (to be consistent with other runs in this chapter), or reduced to 0 (to remove any dependence on angular basis functions). In figure 4.8, the loss for each network activated by either a tanh or linear activation function is shown. Here we see that for the network using angular basis function inputs, the linear network converged to a lower error (0.9 [meV/atom]) than the tanh network (1.3 [meV/atom]). We saw in section 4.1.2 that the number of radial basis functions increase the quality of the fit for linear networks for atoms that have a nonlinear radial dependence. In this experiment, only one radial basis function is being used, and the network is still performing slightly better than the tanh network. This result suggests that the angular basis functions are compensating for the reduction in radial basis functions, allowing the linear network converge to a lower error than the tanh network.

Now we will look at what happens when the angular basis functions are removed. In figure 4.9, we see that both the tanh network and the linear network converge to a higher error than when angular basis functions are included. We see that the linear network, however, quickly converges to a much higher error than before (13.8 [meV/atom]). Whereas the tanh network convergence increases only to 8.5 [meV/atom]. This is still close to our target accuracy of 5 [meV/atom] and is acheived on a network that is using only one radial basis

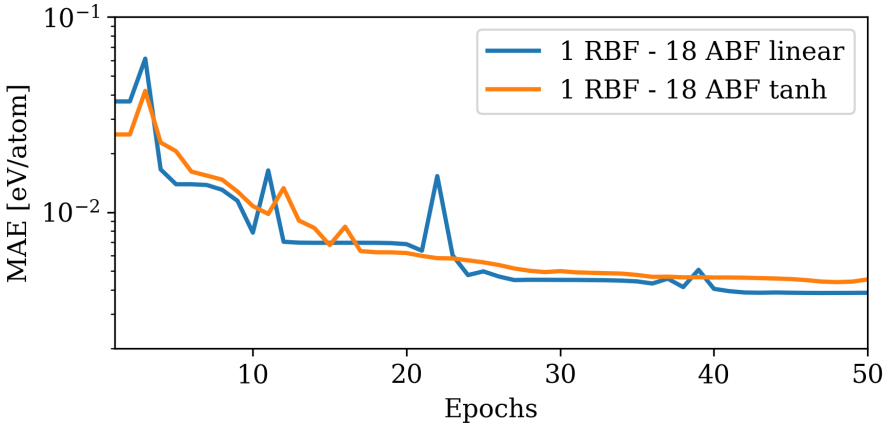

FIGURE 4.8: The graphene dataset is used to train 19-10-10-1 linear and tanh networks where there is one radial basis function and 18 angular basis functions.

function as input. This result is consistent with previous tests on the pair potential model (section 4.1.2), where the tanh network performed much better than the linear network for a fewer number of radial basis functions.

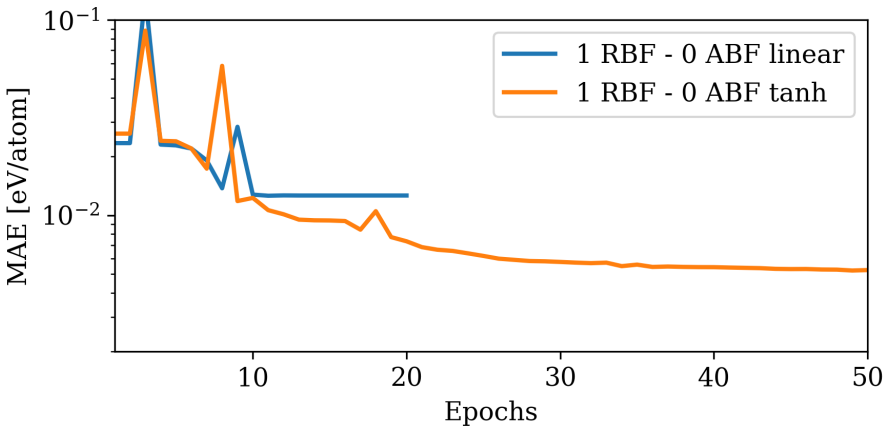

FIGURE 4.9: The graphene dataset is used to train a network with only one radial basis function (and no angular basis functions). Linear and tanh activation functions are used. The linear network gets stuck in a local minima and converges at a higher error.

4.2.2 Sampling and Data Augmentation

Let us return to the question of why the linear network performs better than the tanh network on graphene for an equivalent number of basis functions. In the pair potential experiments, the two hydrogen atoms were brought very close together, and then pulled apart. This created a nonlinear interaction potential where the energy was highest at close distances, decreased to a minimum at the ideal bond length, and then increased back to zero as they were pulled apart.

In a typical MD simulation, the most frequently visited configurations would occur near energy minima, as higher energies occur less frequently. While a range of temperatures (300K-1000K) were sampled to generate graphene structures, configurations are less likely to access high energy regions of the potential energy surface from temperature sampling alone. If the configurations do not explore the high energy regime, then the PES is incompletely sampled. As a result, a neural network potential trained on the incomplete sample does not generalize to these energies.

Of course, in supervised learning, the neural network has limited capacity to predict well on examples that are very different than what was provided in the reference data. If a NNP is trained on a vast amount of data that samples low energy configurations, and a rare event occurs, the NNP will not approximate as well to high energy events. Ideally, the neural network would have access to a dataset that samples a large amount of configuration space, but sampling all of phase space would be computationally intractable.

The curvature of the potential energy surface is low near energy minima relative to the rest of the curve. If the potential energy surface is sampled only near the minima, the potential energy surface will be easier to learn. This is due to the fact that more nonlinear regions of the potential energy surface are ignored. Hence, a linear combination of radial basis functions and angular basis functions may be sufficient to predict data sampled from MD trajectories. A more robust potential however, needs to train on a larger sampling of configuration space. This can be accomplished through data augmentation.

To include structures in the data set that explore higher energies, the unit cell was scaled by up to $+/-10\%$ around structures obtained from MD. The set of scaling factors used to multiply coordinates and lattice vector were SF$\in \{-0.10, -0.05, 0.00, 0.05, 0.10\}$ (where SF is the scaling factor). Scaling the data

increases the complexity of the function to be learned by the network.

To determine how scaling affects the ability for the network to learn, different combinations of the amount of augmented data to the amount of non-augmented data to include in a reference dataset were explored. Previously all of the data was non-augmented, and came entirely from MD trajectories. The ratio of the amount of augmented data to the amount of non-augmented data in the reference data set is varied from 100:0, 50:50, 25:75 and 0:100. The scaled data is expected to have the most nonlinear mapping from coordinates to potential energy surface.

In figure 4.10, the loss curves are plotted for each ratio of augmented to non-augmented data. To be more specific, the total dataset consisting of 25%, 50% and 100% augmented data is combined with 75%, 50% and 0% non-augmented data. The losses for each combination are plotted alongside the loss of a dataset consisting of 100% MD trajectory sampled structures for reference.

When 25% of the data is augmented, the network converges to 5.4 [meV/atom]. Augmenting half the data increases the error convergence to 6.8 [meV/atom]. Augmenting all the data nearly doubles the error of augmenting a quarter of the data. When 100% of the data is augmented the error converges to 10.6 [meV/atom]. Here we see that scaling the data significantly increases the difficulty for the network to learn a mapping from coordinates to PES. This is due to the fact that there is a larger spread of energies, and an increase in nonlinearity of the PES.

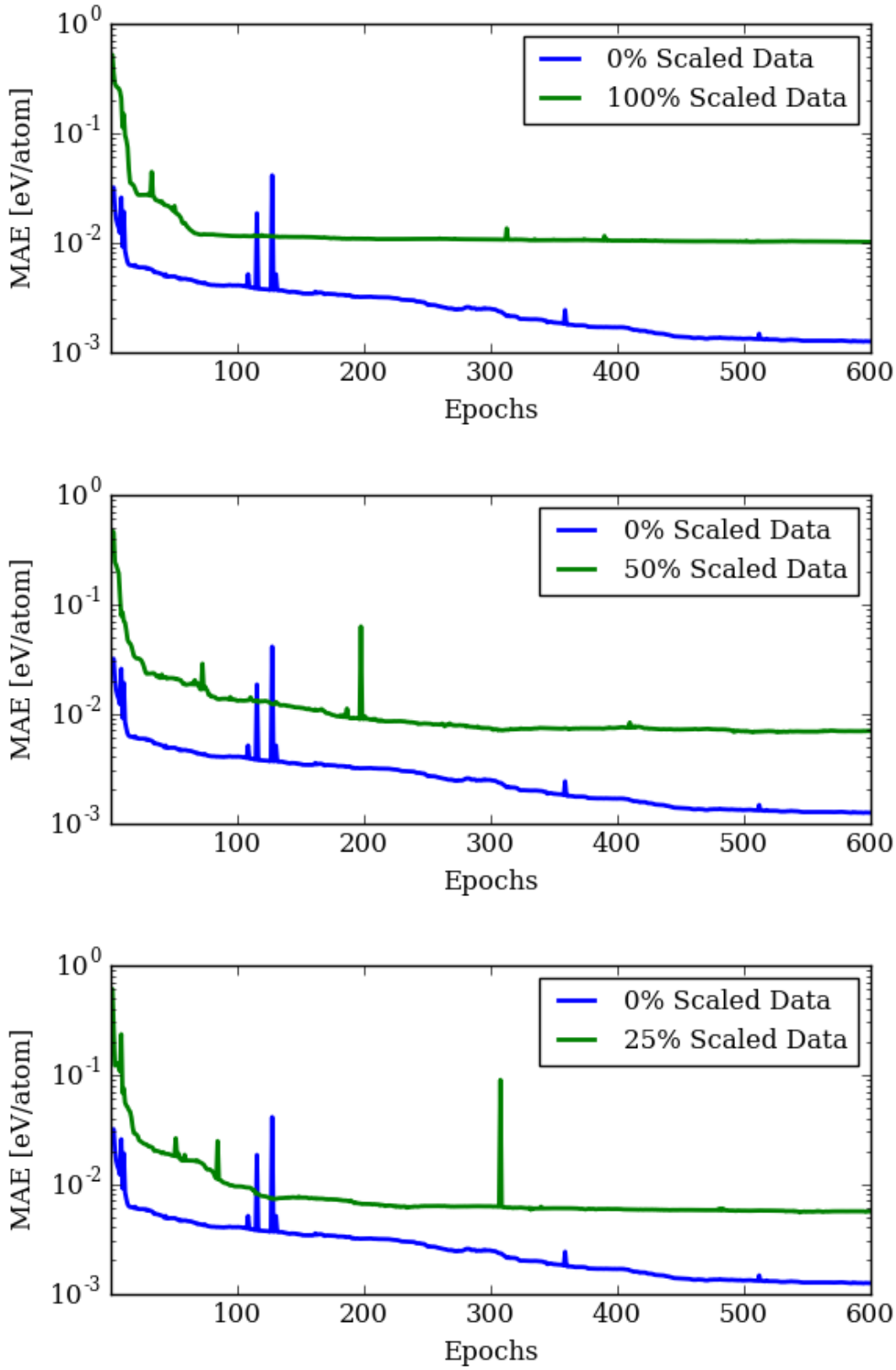

FIGURE 4.10: Loss curves are plotted for various percentages of augmented graphene data. If 0% of the data is scaled, the dataset consists entirely of structures from MD trajectories. If 100% of the data is scaled, the dataset consists of all augmented structures. Percentages in between 0% and 100% indicate a combination of the two.

4.3 Titanium Dioxide (TiO_2)

In this section we will look at a dataset that has been augmented in multiple ways and compare to our graphene dataset. This dataset, based on Titanium Dioxide (TiO_2) structures, was provided as an example made publicly available with the ænet software. We will compare the spread of energies in this dataset to the spread of energies in the graphene dataset. From here, we will determine a rough estimate for how much of the graphene dataset should be augmented and how much should come from MD trajectories to compare to the example dataset provided by ænet.

While the dataset contained 7694 structures (obtained from DFT) in total, the number of atoms in each structure varied from 6-95 atoms. In total, there are 165229 atomic environments in the TiO_2 dataset. In the graphene dataset, there are 256000, meaning that there are 90771 more atomic environments to train on than in the TiO_2 dataset.

The TiO_2 reference data was augmented in 3 different ways [3]:

1. scaling the lattice constants to a range of +/- 10% around ground state
2. gradually tilting the crystal structures
3. stretching and compressing in one crystal dimension

To simplify analysis, since we have only been looking at how changes in radial distances of atoms affect the networks ability to learn a neural network potential, only the first augmentation step is taken into account when implementing data augmentation. The other two augmentation steps would change the angular distributions, and since we have not looked into how angular basis functions play a role in ænet, these steps are excluded. In a first comparison of the TiO_2 dataset to the graphene dataset, we will look at how networks with linear and nonlinear activation functions converge during training on TiO_2. This will give us an idea of how data augmentation increases the complexity of the potential energy surface. For this test, the loss is computed as the number of radial basis functions are increased. In figure 4.11, we see a 63-10-10-1 (with one radial basis function and 62 angular basis function) network optimized with the l-BFGS optimizer using either the linear or tanh activation functions. At one radial basis function, we see that tanh does significantly better than the linear

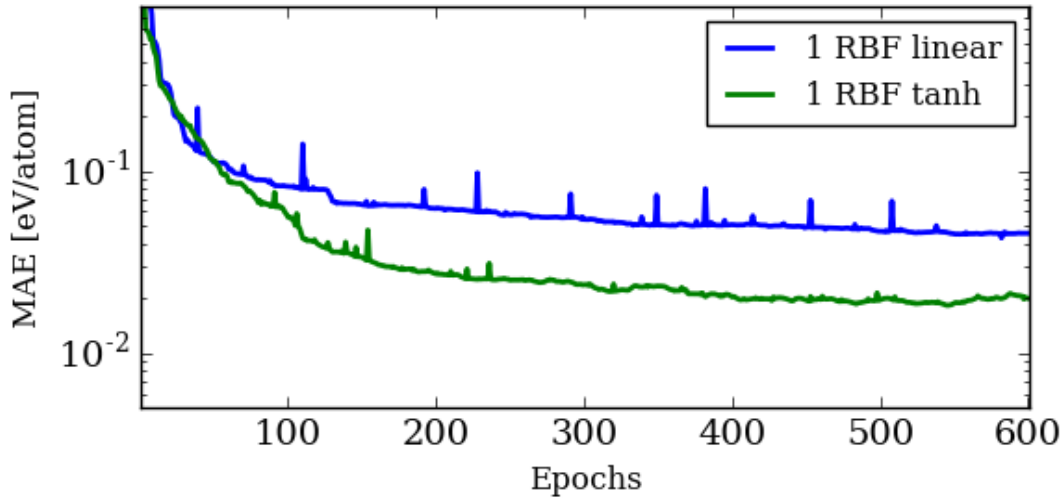

FIGURE 4.11: 63-10-10-1 networks (one RBF) using either linear or tanh activation functions are optimized with l-BFGS. The TiO_2 reference data is used.

network. Here tanh network converges to a testing error of $\sim$ 15 [meV/atom], and the linear network convergest to a testing error of $\sim$ 38 [meV/atom] (where 5 [meV/atom] is the target error [3]). We saw that in the pair potential experiments (section 4.1.2), the tanh network was better able to learn a nonlinear mapping given fewer radial basis function inputs than the linear network.

As we increase the number of radial basis functions from 1 to 8, we see that the linear network gets consistently better as the number of radial basis functions increase (figure 4.12). This is consistent with what we learned in section 4.1.2, where increasing the number of radial basis functions improves the linear network's capacity for learning a nonlinear mapping from input data to output data. At 8 radial basis functions (figure 4.13), the linear network converges to $\sim$ 17.5 [meV/atom] which is 20.5 [meV/atom] lower than when only one radial basis function is used. For the tanh network, the error converges to 8.5 [meV/atom] which is 6.5 [meV/atom] lower than the error at 1 radial basis function. The tanh network, however, takes longer to converge.

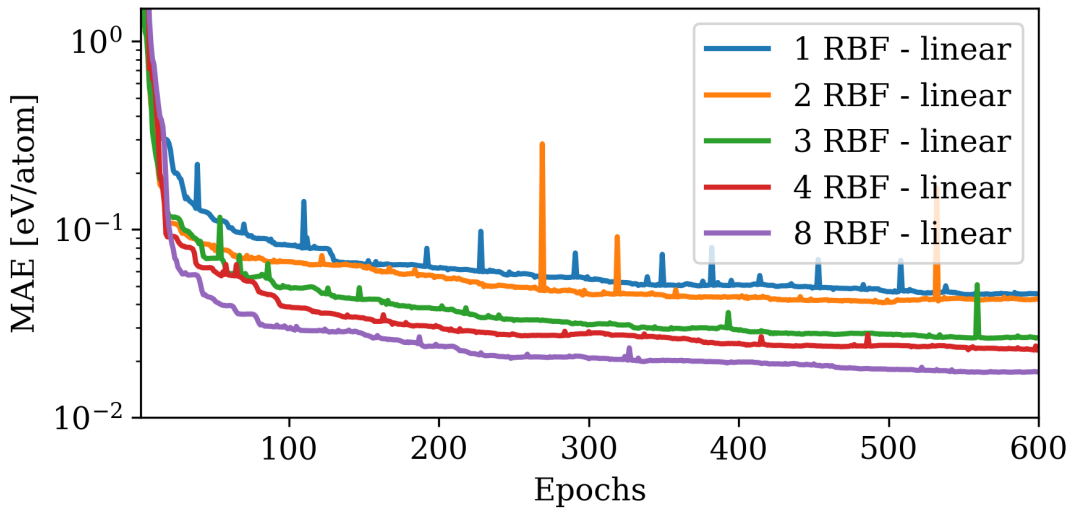

FIGURE 4.12: N-10-10-1 network (increasing RBFs from 1 to 8) using only linear activation functions and optimized with l-BFGS. The TiO_2 reference dataset is used.

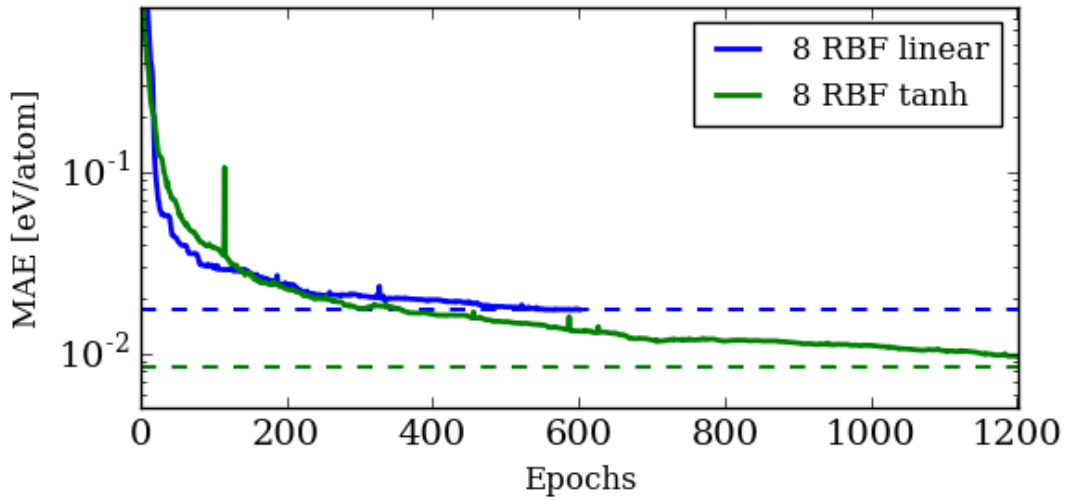

FIGURE 4.13: 70-10-10-1 networks (8 RBFs) with either tanh or linear activation functions are optimized with the l-BFGS optimizer. The TiO_2 reference dataset is used.

4.3.1 Data Augmentation

Since ænet trains on the cohesive energy per atom, to compare my data to the TiO_2 dataset (containing augmented structures), I compared the distribution of cohesive energy per atom for each structure in each data set. The cohesive energy per atom is given by:

$$E_{coh/atom} = \frac{E_{coh}}{N_{Tot}} = \frac{1}{N_{Tot}}\left(E_{sys} - \left(\sum_{atom}^{N_{species}} E_{atom}N_{atom}\right)\right) \tag{4.3}$$

Where N_{Tot} is the total number of atoms in the system, E_{sys} is the total energy of the system, E_{atom} is the atomic energy of each atomic species, N_{atom} is the number of atoms that belong to each atomic species (eg. $N_{Si} = 36$ and $N_O = 72$ if there are 36 silicon atoms and 72 oxygen atoms in the system) and $N_{species}$ is the total number of atomic species in the system (eg. if we are looking at 2D silica, the summation will be over 2 species: oxygen and silicon). E_{atom} is determined by computing the energy of an isolated atom of a given species.

To get a general idea of how much to scale MD data, the $E_{coh/atom}$ spread of unscaled MD data was compared to the TiO_2 dataset. The distribution of $E_{coh/atom}$ for the TiO_2 data set is visualized in figure 4.14. While most of the data is between -10 and -5 eV/atom, some strongly distorted structures give rise to energies as high as 20 eV/atom. Since we are looking mostly at how changing the number of radial basis functions affect learning, the graphene dataset will only be augmented by expanding or compressing the system by a scaling factor between +/- 10% of the original coordinates. This should keep the focus mainly to changes in radial distance from reference atoms, and how radial basis functions can enhance or reduce the quality of the fit.

To get a rough estimate of how much scaled data should be included in graphene, the range of energies in the graphene dataset was compared to that in the TiO_2 dataset. The spread of energies for graphene was measured by its range, which is the difference of maximum and minimum cohesive energy per atom. Because TiO_2 and graphene have different energy minima in each respective dataset, we compared the overlap between the two datasets as follows.

In graphene, the total spread in energies arise from thermal fluctuations. We will assume that the energies in the TiO_2 dataset corresponding to this range also arise from thermal fluctuations. The range of energies in the graphene

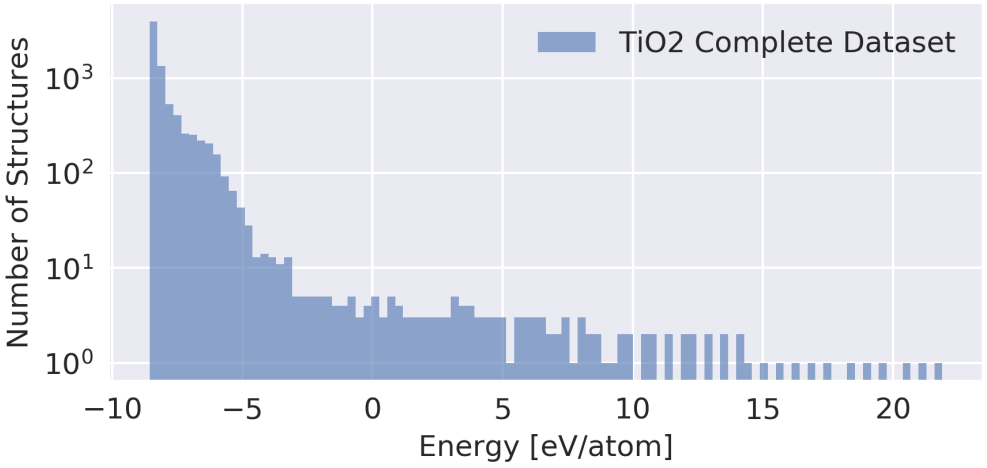

FIGURE 4.14: Distribution of cohesive energies per atom in the complete TiO2 dataset.

dataset is:

$$R_G = \max(E_{coh/atom}) - \min(E_{coh/atom}) \tag{4.4}$$

We were interested in how many of the TiO_2 structures were within R_G of the minimum energy in the TiO_2 dataset. In other words, what fraction of the TiO_2 dataset have energies below R_G in TiO_2. The range R_G in TiO_2 is given by,

$$R_G \text{ in } TiO_2 = \min(E_{coh/atom}) + R_G \tag{4.5}$$

Of the 7815 structures in the TiO_2 dataset, only 1446 fell within this energy range. This represents only 18.5% of the TiO_2 dataset.

Since the ænet example produced a potential that was suitable for MD, a similar ratio of lower energy data (obtained from MD) to higher energy data (obtained from compression and stretching the coordinates) will be used for training in the remaining tests. Approximately 20% of structures will be sampled from the MD trajectory, and 80% of structures will be obtained from scaled data.

Recall that previously only 5 scaling factors were drawn uniformly from the interval (-0.1,0.1) for stretching or compressing coordinates and lattice vectors.

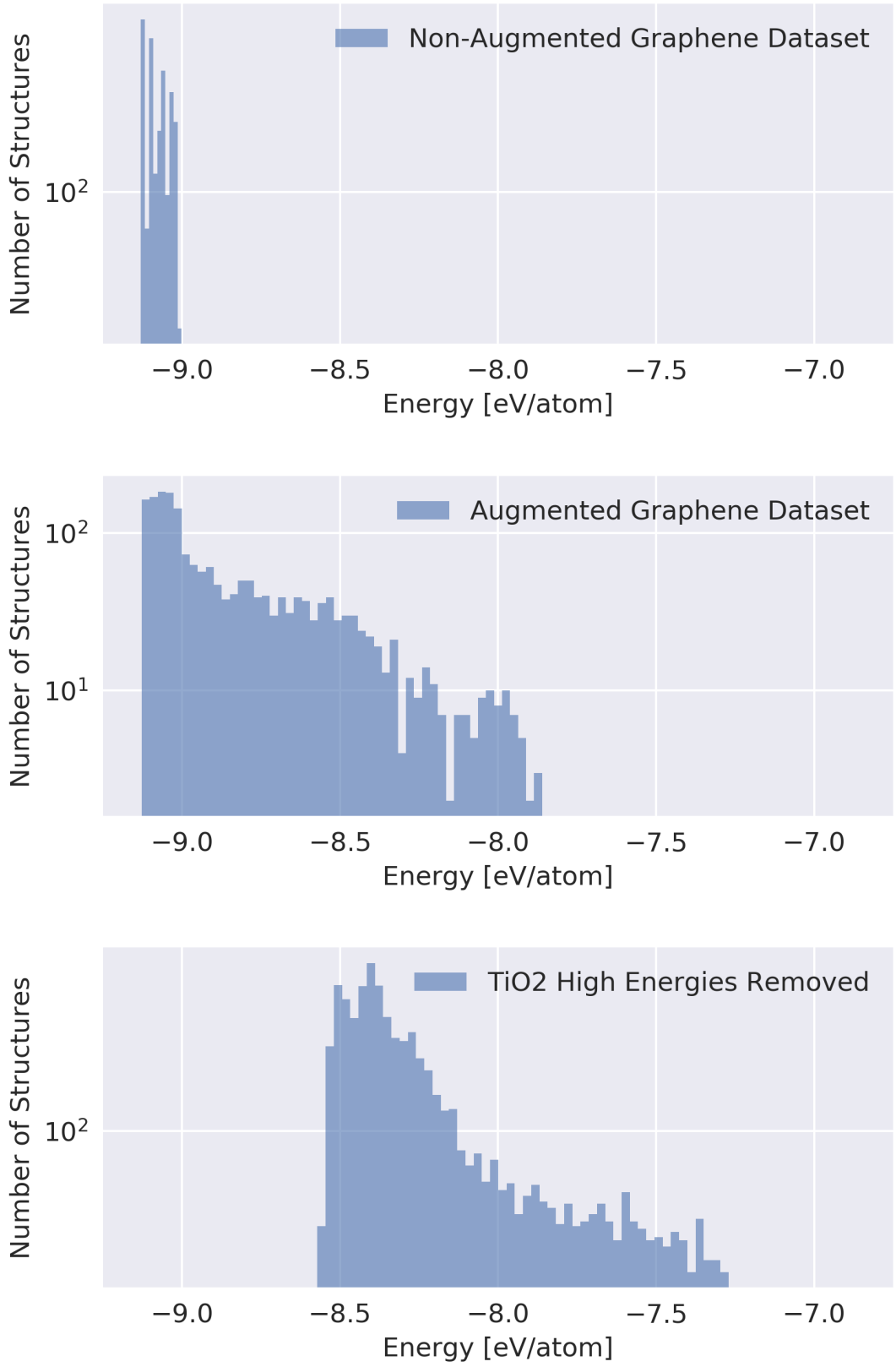

FIGURE 4.15: The distribution of cohesive energies per atom in the **Top:** non-augmented graphene dataset (structures sampled entirely from MD). **Middle:** augmented graphene dataset (structures sampled from the stretched or compressed dataset) **Bottom:** TiO_2 dataset where energies that are higher than the range of graphene in TiO_2 are removed.

In the following runs, each structure in the dataset (retrieved from MD trajectories at 300K, 500K, 750K and 1000K) will be stretched or compressed by 10 scaling factors drawn randomly from (-0.1,0.1). Instead of having energies clustered around values corresponding to these preset scaling factors, the random sampling of scaling factors should help smooth the distribution of energies. We can see the distributions of energies for the non-augmented graphene dataset, the augmented graphene dataset and the R_G in TiO_2 dataset (with higher energies removed) in figure 4.15.

To generate an augmented dataset for graphene, the stretched and compressed structures will be randomly sampled to make up 80% of the training data. The remaining 20% of the reference data will be randomly sampled from various MD trajectories. To summarize, datasets including data augmentation will be generated in the following steps:

1. Run various MD trajectories at different temperatures and write structures every n steps to generate structures in the unscaled dataset.
2. Scale all the coordinates and lattice vectors by 10 scaling factors which are drawn randomly from -0.1 to 0.1.
3. 80% of the training data is randomly sampled from the set of augmented structures.
4. 20% of the training data is randomly sampled from the set of MD trajectory structures.

This algorithm will provide a way to generate a graphene dataset that is roughly comparable to the example TiO_2 dataset provided by ænet. This will serve as a starting point for developing a more robust potential. We will return to this in chapter 5.

Chapter Summary

- Linear networks in ænet are similar to one layer RBF networks, but basis functions have fixed parameters.
- tanh activation networks still learn effectively for low number of RBF (when restricting input to only include radial basis functions) compared to linear activation networks.

- Parameters selected for radial basis indicate how sensitive the radial basis functions will be over corresponding length scales.
- Strictly sampling MD trajectories around equilibrium lead to less complex potential energy surfaces.
- Important to scale data to explore PES more completely. This will inevitably lead to more a more difficult problem.

Chapter 5

Bulk and 2D Materials

5.1 Materials

In this chapter we will explore how the ænet algorithm performs on different material compositions and structures. First we will discuss how the potential energy surfaces differ between materials. From here we will apply what we've learned in the previous chapter (Chapter 4) and sample an augmented dataset for graphene to explore a more complex potential energy surface.

5.1.1 2D Silica and Bulk SiO_2

In this experiment, the 2D silica bilayer was compared to its 3D counterpart, bulk SiO_2. To get a clearer picure of how the potential energy surface of the two materials differ, only structures obtained from temperature sampling are included in the reference dataset. This is to avoid the added complexity introduced by stretching and compressing structures. Temperature sampling allows us to explore near-equilibrium states of the potential energy surface, so these neural network potentials will be able to better predict frequently visited configurations. In the following set of runs, a 70-10-10-1 neural network architecture with tanh activation functions is optimized with the l-BFGS optimizer. There are 2000 structures in each reference dataset (sampled from MD trajectories). The reference data is spit into 85% training data and 15% testing data. A cutoff radius of $R_c = 6.5$ is used.

After training each type of structure, the bulk silica was found to converge to a lower error than the 2D silica (figure 5.15). It is possible that this may be the result of greater symmetry in the bulk system. Each atom in the bulk material

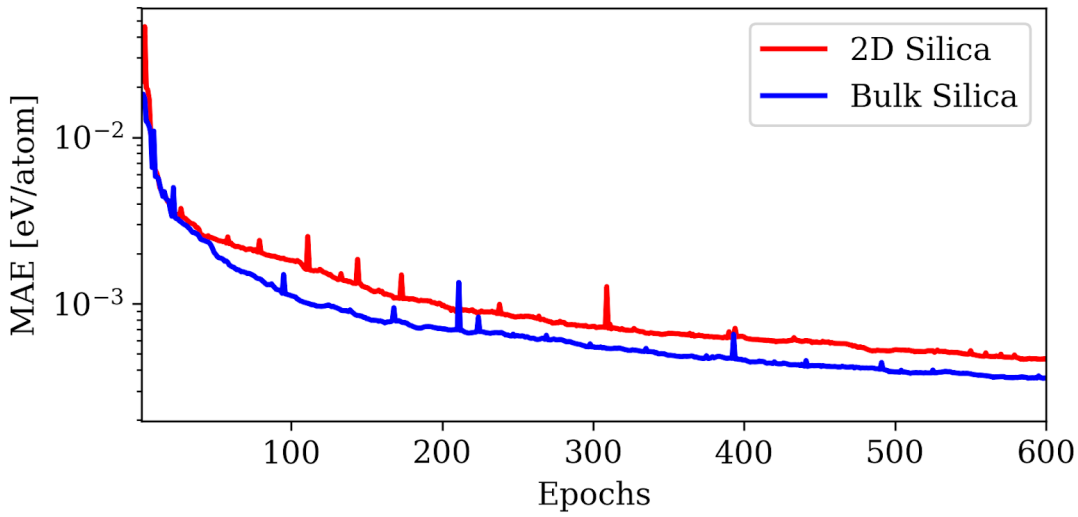

FIGURE 5.1: Loss curves for 2D and bulk silica (datasets are sampled from MD trajectories). A network architecture of 70-10-10-1 is optimized using l-BFGS.

has repeating neighbouring atoms in each direction, whereas the bilayer silica has vacuum on two edges.

The results in chapter 4, suggest that more nonlinearity increases the complexity of the potential energy surface making it more difficult for ænet to converge. In this chapter, to further explore whether the nonlinearity of the potential energy surface is responsible for the difficulty for the network to learn, data is augmented by stretching and compressing and the neural network potentials are used to make predictions on these structural energies. Here the cohesive energy per atom for each structure was determined as a function of scaling factor for stretching or compressing around the ground state. First, a geometry relaxation was performed on each structure where lattice vectors were allowed to change. Once the structures were minimized, a range of scaling factors on the interval of (-0.1,0.1) were chosen to scale the coordinates and lattice vectors. For negative scaling factors, the structures are compressed up to 10% of the ideal (minimized) structure. Likewise, positive scaling factors indicate that the coordinates and lattice vectors are being expanded by up to 10% around the ideal structure.

These scaling experiments are inspired by tests using the pair potential (chapter 4). For the pair potential, atoms that are really close have a rapidly increasing

energy, and when they get pulled away from the ideal bond length the energy increases until there is no longer an interaction at all. Here we will push materials outside the range of configurations for which the neural network was trained. We note that this is a more complicated many-body system, so the comparison is not direct.

Once the dataset was generated, the neural network potential (trained on reference data which sampled MD trajectories) was used to predict the energies of all the scaled structures. For brevity, we will refer to datasets that consist only of structures sampled from MD trajectories as dataset A. The neural networks were expected to predict best around a scaling factor of zero, which corresponds to the ideal structure of the material. This is because MD simulations sample around equilibrium.

In figure 5.2, the cohesive energy per atom is plotted for each scaling factor used to expand or compress the structures. To provide an idea of where the neural network is likely to learn best, the mean and maximum values of the cohesive energy per atom in the reference data set are plotted for each material (figure 5.2). In each experiment, the neural network potential was able to predict the energy curve very closely around equilibrium, and within the energy range upon which the network was trained. From observing the red line indicating the maximum value of cohesive energy per atom in the reference data set, we see that the neural network is able to extrapolate fairly well outside of the values for which it was trained. This indicates that the network is learning the derivatives, since the shape of the curve is preserved for a small range of scaling factors beyond the range of energies for which the network was trained. It eventually, begins to predict poorly as the energies continue to increase.

Note that during MD, the lattice vectors are fixed, so the cohesive energy with respect to the scaling factor curve is not a direct indication of the structures that are explored during MD. With fixed lattice vectors, volume is fixed, so the expansion of one bond must come at the compression of others. Stretching or compressing the coordinates and lattice vectors for augmentation, however, enforces the bond lengths to change as a group.

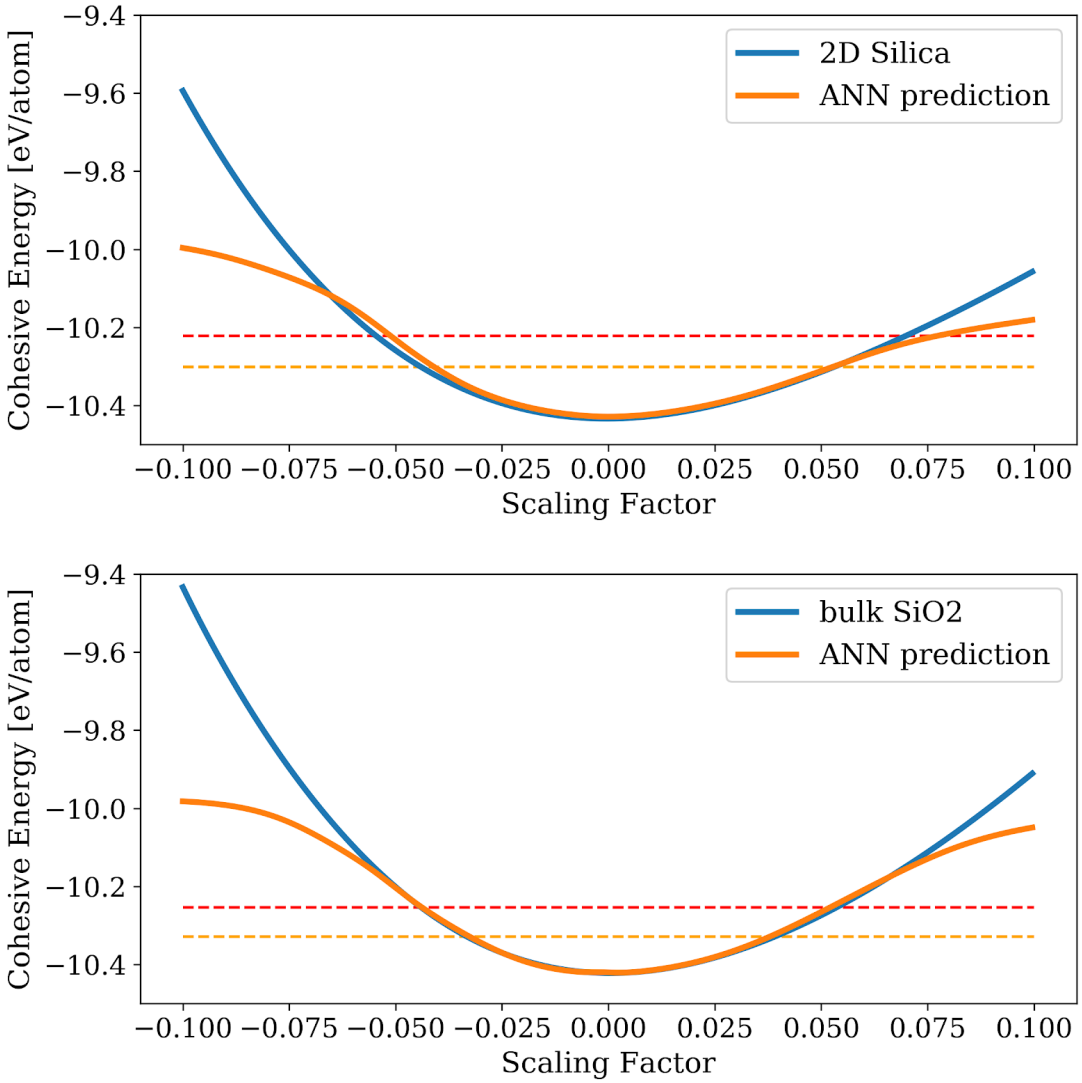

FIGURE 5.2: Cohesive energy per atom as a function of scaling factor. The red horizontal line corresponds to the maximum value of cohesive energy per atom in the training set. The orange horizontal line corresponds to the mean value of cohesive energy per atom. The true cohesive energies per atom are plotted with the ANN prediction of the energies at these points. **Top:** 2D silica energies. **Bottom:** Bulk silica

High pressure MD sampling

In the previous experiment, we implemented temperature sampling around equilibrium. Now we will look at what happens if we train a neural network

potential on MD data that was forced out of equilibrium. In the following experiment, we train on an MD trajectory that samples high pressure 2D silica by maintaining fixed lattice vectors that compress the system by 4.5%. In figure 5.3, we see that the loss curve for the 2D silica sampled near equilibrium converged to a lower value than the loss curve for the 2D silica sampled away from equilibrium at a high pressure. This indicates that it is more difficult for the network to train on data sampled out of equilibrium.

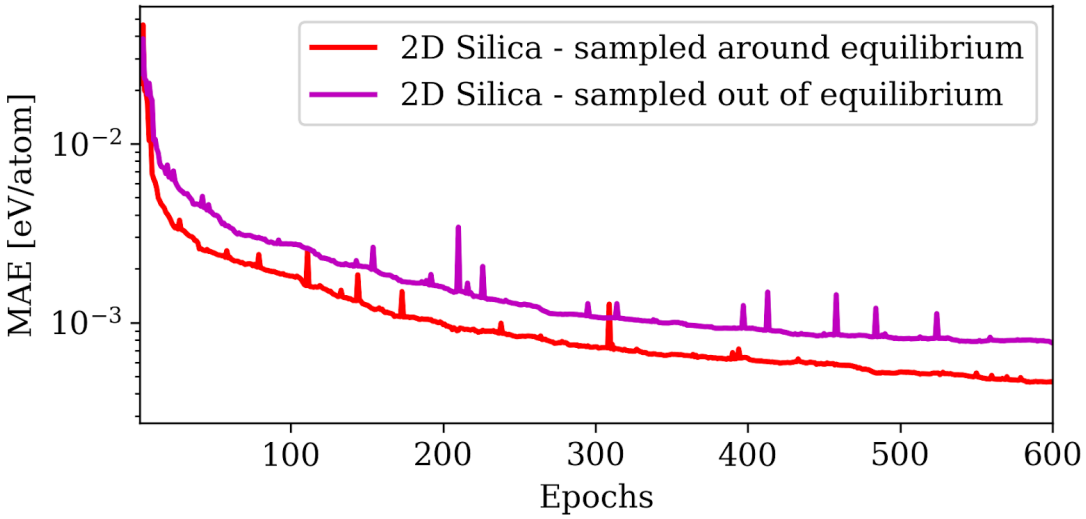

FIGURE 5.3: Loss curves for 2D silica where data is sampled near equilibrium, or away from equilibrium (high pressure). A network architecture of 70-10-10-1 is optimized using l-BFGS.

In figure 5.4, we compare the predictions of the neural network potentials trained on MD data sampled near and away from equilibrium. For the high pressure 2D silica fit, the neural network potential is biased to predict well on compressed structures. This is expected, as the network was trained on compressed structures. When predictions are made on structures augmented by higher scaling factors, the fit is significantly less accurate than the fit of the network trained on data near equilibrium. We conclude that training on data sampled away from equilibrium makes it more difficult for the network to generalize to other parts of the curve. This is likely due to a more complex mapping between structures and energies.

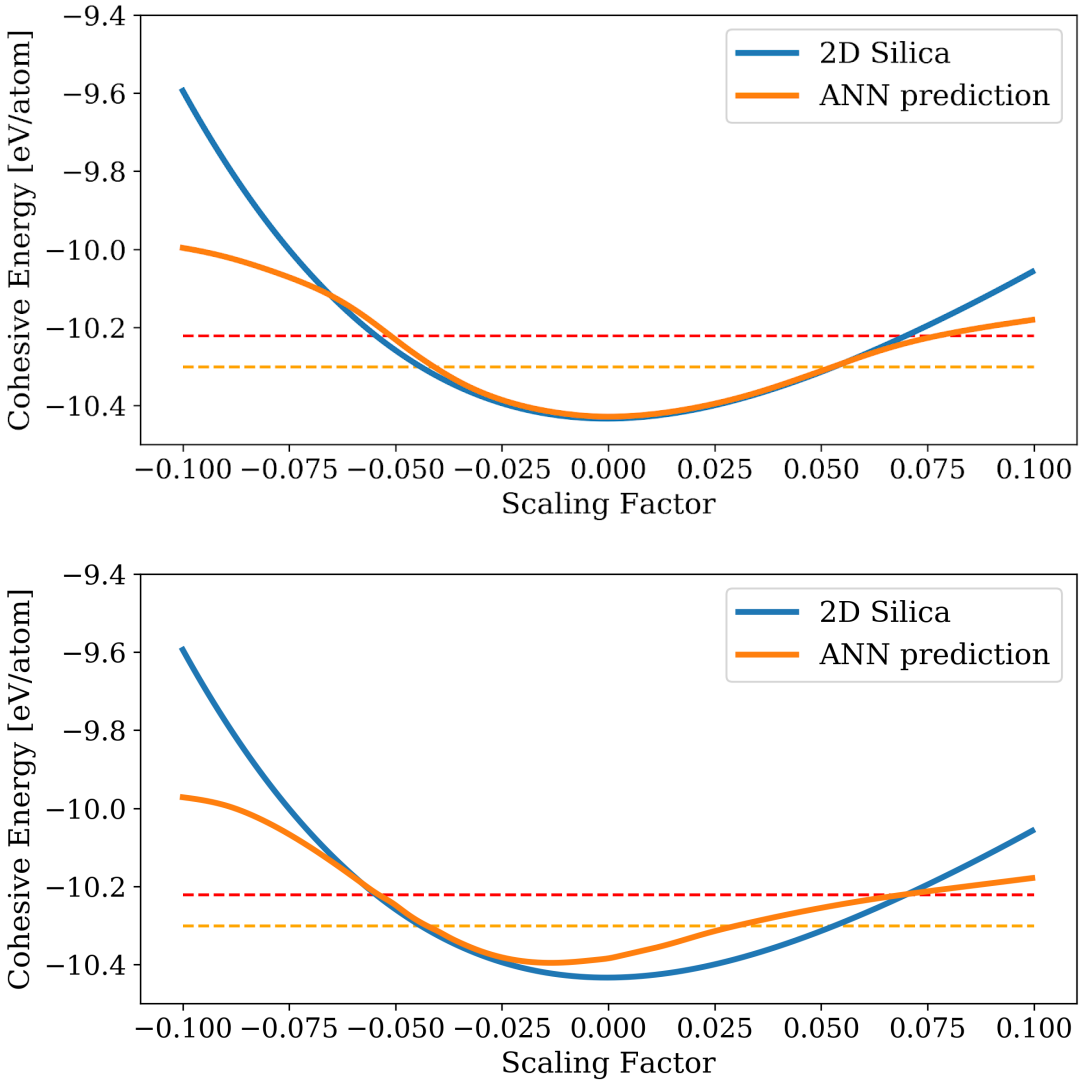

FIGURE 5.4: Cohesive energy per atom as a function of scaling factor. The red horizontal line corresponds to the maximum value of cohesive energy per atom in the training set. The orange horizontal line corresponds to the mean value of cohesive energy per atom. The true cohesive energies per atom are plotted with the ANN prediction of the energies at these points. **Top:** 2D silica energies. Notice that the network potential makes biased predictions. This is due to MD sampling around fixed lattice vectors corresponding to this region. **Bottom:** Bulk silica

5.1.2 hBN and cBN

In this experiment, hexagonal boron nitride is compared to cubic boron nitride. For these runs, lattice vectors were optimized as well as geometries. The same procudure for training as outlined in for the SiO_2 runs (section 5.1.1) was used here.

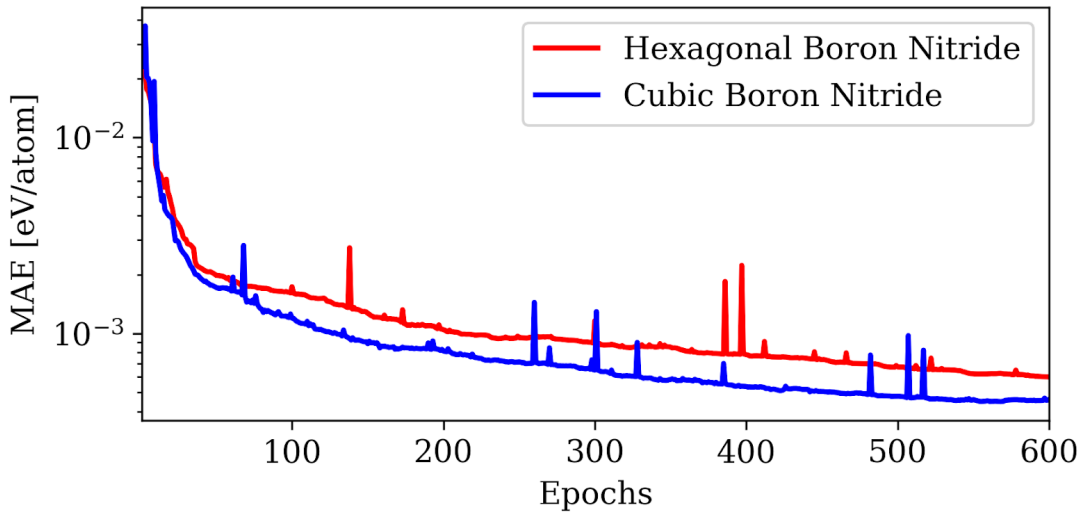

FIGURE 5.5: Loss curves for hexagonal boron nitride and cubic boron nitride (datasets are sampled from MD trajectories). A network architecture of 70-10-10-1 is optimized using l-BFGS.

The loss curves for the boron nitride system indicate that the bulk material converges to a lower loss than the 2D material (figure 5.5), however, the difference is less exaggerated than for the SiO_2 experiments. Overall, the loss curves for the boron nitride systems are lower than the loss curves for the high pressure SiO_2 systems. The difference in loss curves between the high pressure silica and boron nitride is consistent with the idea that a dataset generated near equilibrium with smaller forces is easier to train than a dataset generated away from equilibrium.

When predicting the cohesive energy per atom as a function of scaling factor for the boron nitride systems, the hexagonal boron nitride fit appears to be better over a larger range of scaling factors. This seems to contradict the results of the loss curves. However, there is a larger spread of energies in the hexagonal boron nitride reference data set than for the cubic boron nitride reference

data set. For hexagonal boron nitride, the range in cohesive energies are 20 meV/atom larger than for the cubic boron nitride reference data. In general it can be more difficult to train over a larger energy range.

We note that cubic boron nitride might have less radial deviation from equilibrium state than hexagonal boron nitride when thermal energy is added to the system. It is possible that thermally distorting cubic boron nitride did not lead to structures that are similar to the stretched and compressed augmented data. Therefore, the network might learn the equilibrium structures well, but fail to make correctly on the augmented dataset.

Overall, cubic boron nitride may have trained better due to training over a slightly smaller spread of energies and exploring fewer configurations that have features similar to the augmented dataset.

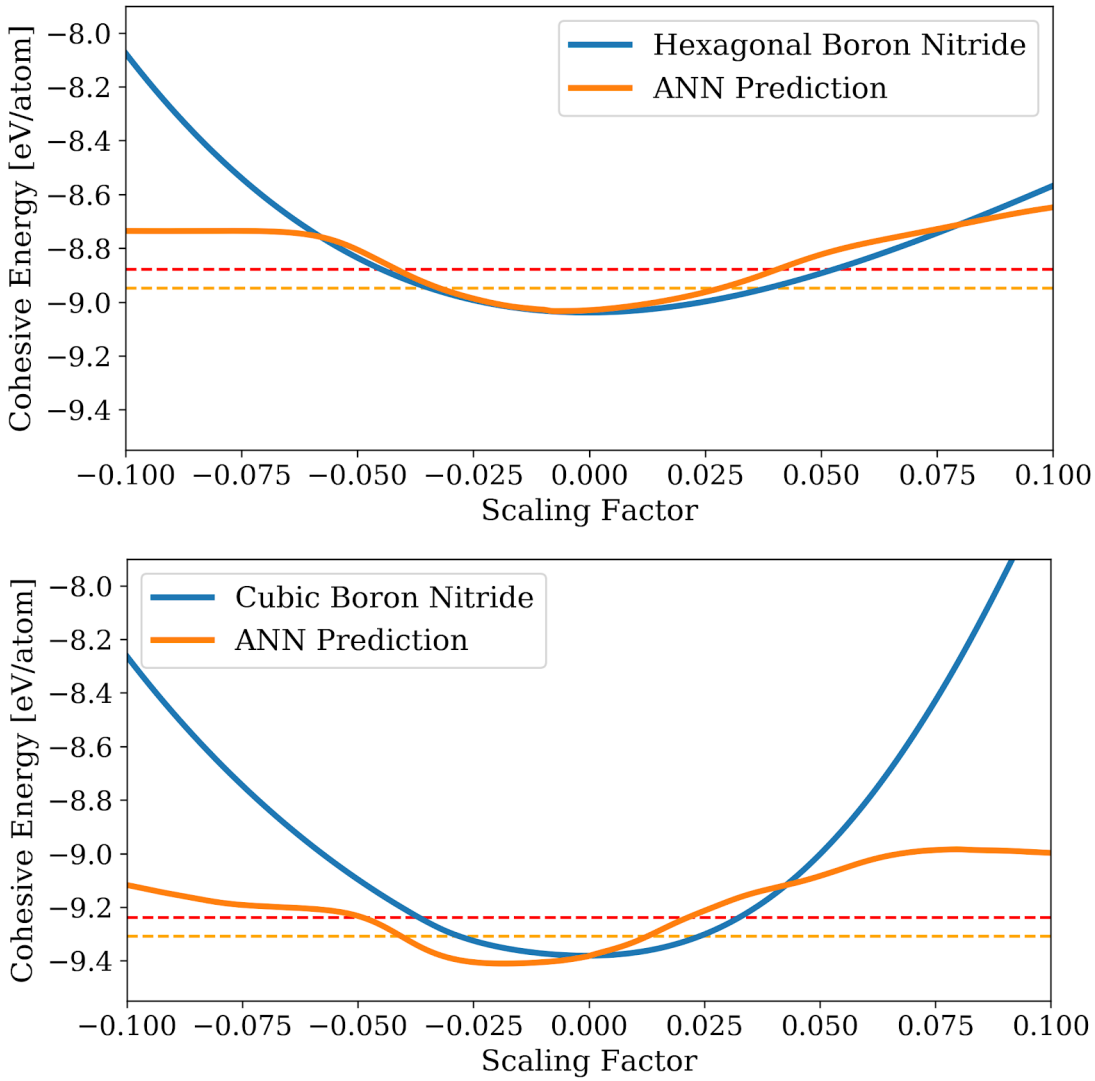

FIGURE 5.6: Cohesive energy per atom as a function of scaling factor (used to expand or compress structures) and the network potential predictions of these energies. The red dashed line indicates the maximum energy included in the reference data set. The orange dashed line indicates the average value of the energy included in the reference data set. **Top:** Hexagonal boron nitride. **Bottom:** Cubic boron nitride.

5.1.3 Graphene and Diamond

Here graphene and diamond are compared to one another. To generate training data, the coordinates and lattice vectors were relaxed. The same procedure for training as outlined in the previous two sections is also used here.

The loss curves (figure 5.7) of graphene and diamond both converged to

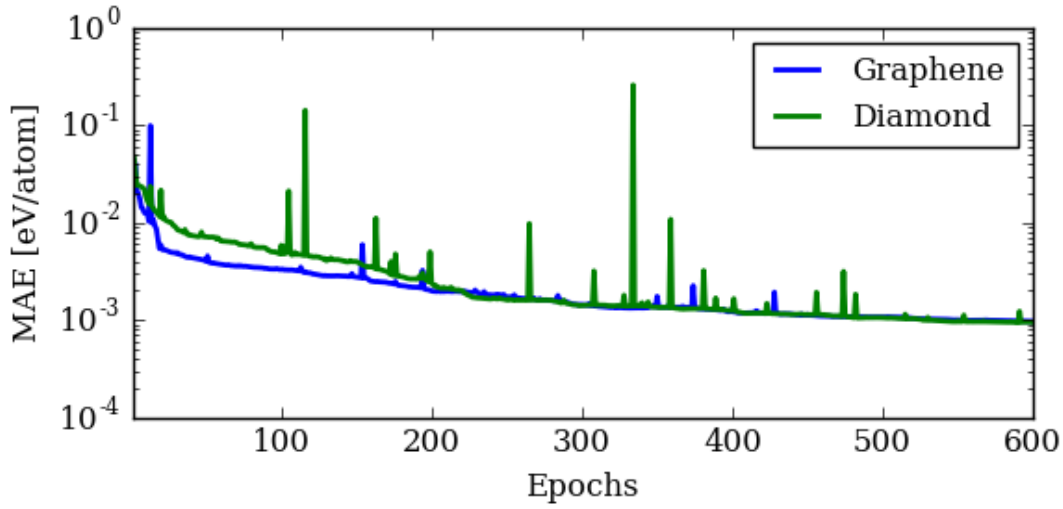

FIGURE 5.7: Loss curves for graphene and diamond (datasets are sampled from MD trajectories). A network architecture of 26-10-10-1 is optimized using l-BFGS.

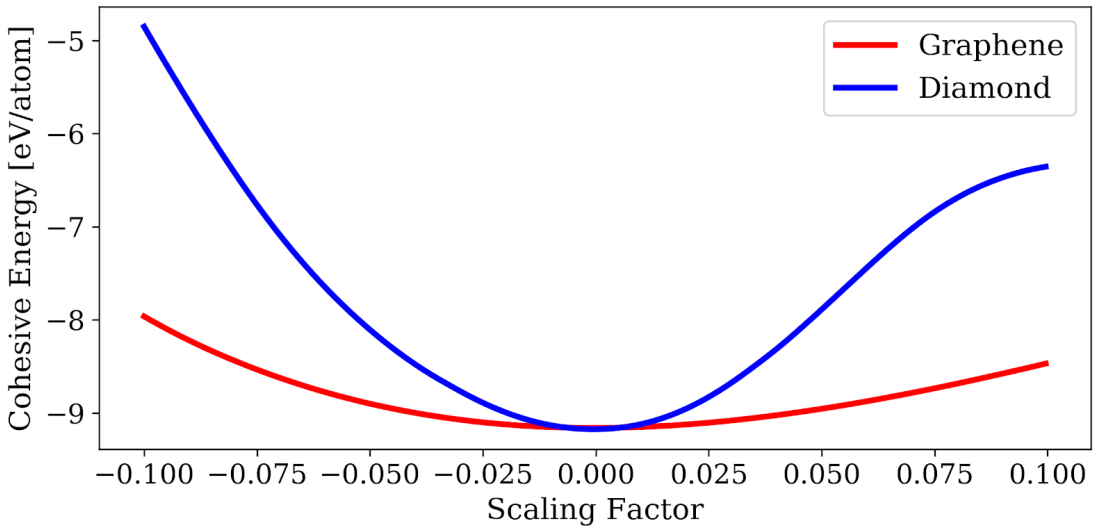

FIGURE 5.8: Cohesive energy per atom as a function of scaling factor for graphene and diamond.

the same error, though the loss for graphene converged more quickly than for diamond. When comparing curves of cohesive energy per atom as a function of scaling factor (figure 5.8), diamond was found to be significantly more non-linear than graphene. The convergence for diamond was likely slower due to

this high degree of nonlinearity. We note that the loss curve for diamond and graphene converged to the highest error of all the systems investigated. In the following section, we will look at how sampling more nonlinear regions of the curve affects the fit. When looking at the network predictions on diamond and graphene (figure 5.9), we see that a very small amount of the cohesive energy curve is fit. As expected, the network predicts best near equilibrium.

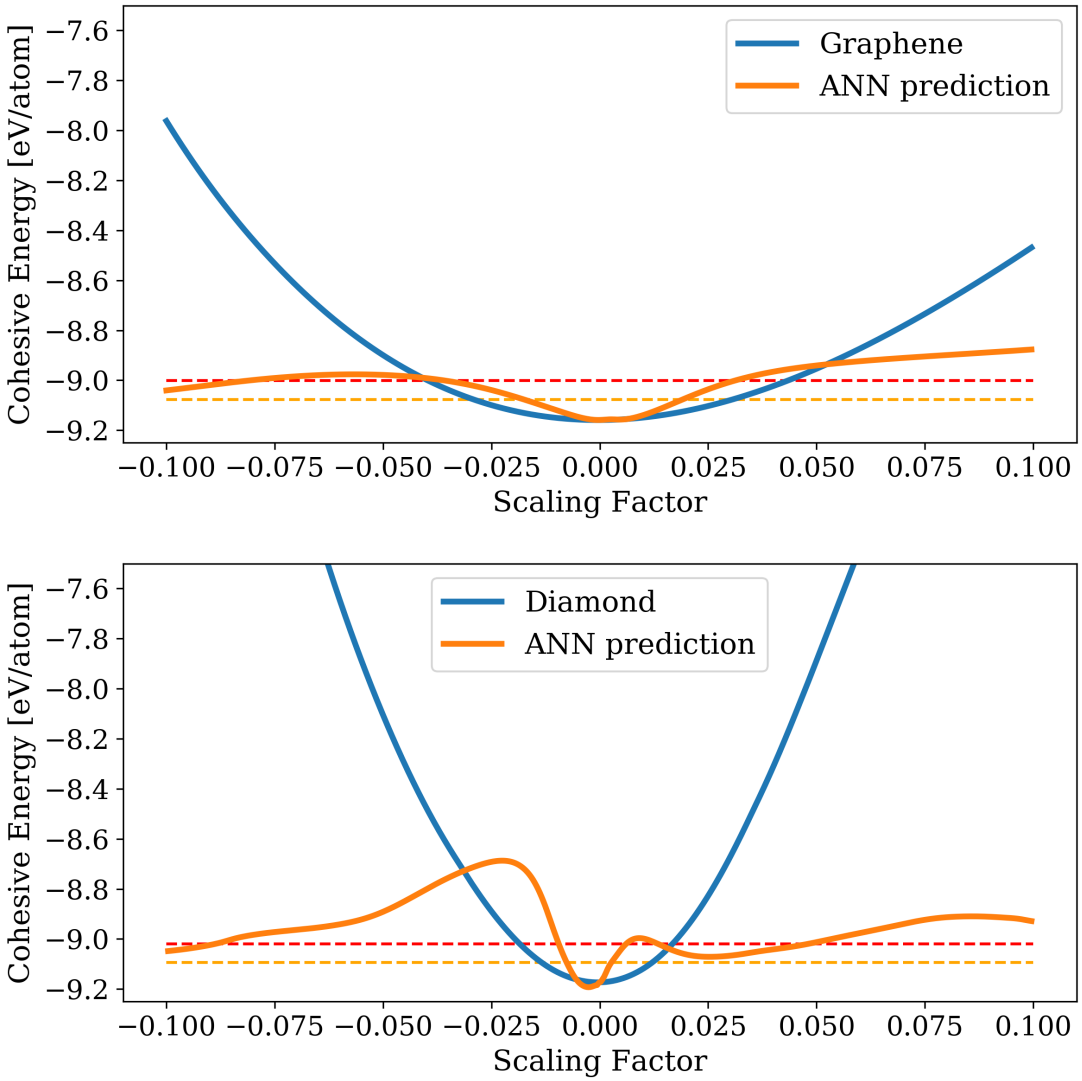

FIGURE 5.9: Network potential predictions on the graphene and diamond datasets. The dashed red line indicates the maximum value of cohesive energy in the dataset. The orange dashed line indicates the mean value of cohesive energy in the dataset. **Top:** graphene **Bottom:** diamond.

5.2 Impact of data augmentation

In this next experiment, we will look at the impact of data augmentation on the complexity of the potential energy surface of graphene. For these runs, we will generate an additional dataset which we will refer to as dataset B. This dataset will consist of augmented and non-augmented data as described in chapter 4. The ideal structure is scaled up to +/-10% around its original coordinates and lattice vectors. Then this distribution of stretched and compressed structures is randomly sampled to make up 80% of the training data. The other 20% of the reference data will come from the non-augmented dataset i.e. structures sampled from MD trajectories at various temperatures. In total 5000 structures (including augmented data and non-augmented data) were included in the reference dataset. This dataset was split into 85% training data and 15% testing data.

In figure 5.10, the network potential trained on dataset B is compared to the network potential trained on dataset A. Both potentials were used to predict structures that were scaled +/-10% around ground state. We can see that, not surprisingly, the potential trained on dataset B is able to predict the curve better than the potential trained on dataset A. Many of the structures that were generated using this range of scaling factors are explicitly represented in this training dataset. The network potential trained on dataset A generally only predicts the structures correctly around the ground state.

It is interesting to take notice of the average and maximum energy included in each dataset. In the top plot of figure 5.10, the average cohesive energies for each reference dataset are plotted. In the bottom, the maximum energy in each reference dataset are plotted. By observing the maximum energy in each dataset, we see that the neural network is able to generally fit the shape of the curve up to this point. The potential trained on dataset A does not fit very close to the curve anywhere besides zero scaling factor. This potential only does well for the ideal case. We should note however, that this is not a perfect test since we are predicting structures with varying lattice vectors by a neural network trained on data with fixed lattice vectors. The potential does particularly poorly when trying to fit highly compressed structures, where the cohesive energy per atom rapidly changes with respect to scaling factor. The potential trained on dataset B has learned the general shape of the curve. However, it does more

poorly where there is a steeper derivative.

Augmented dataset performance

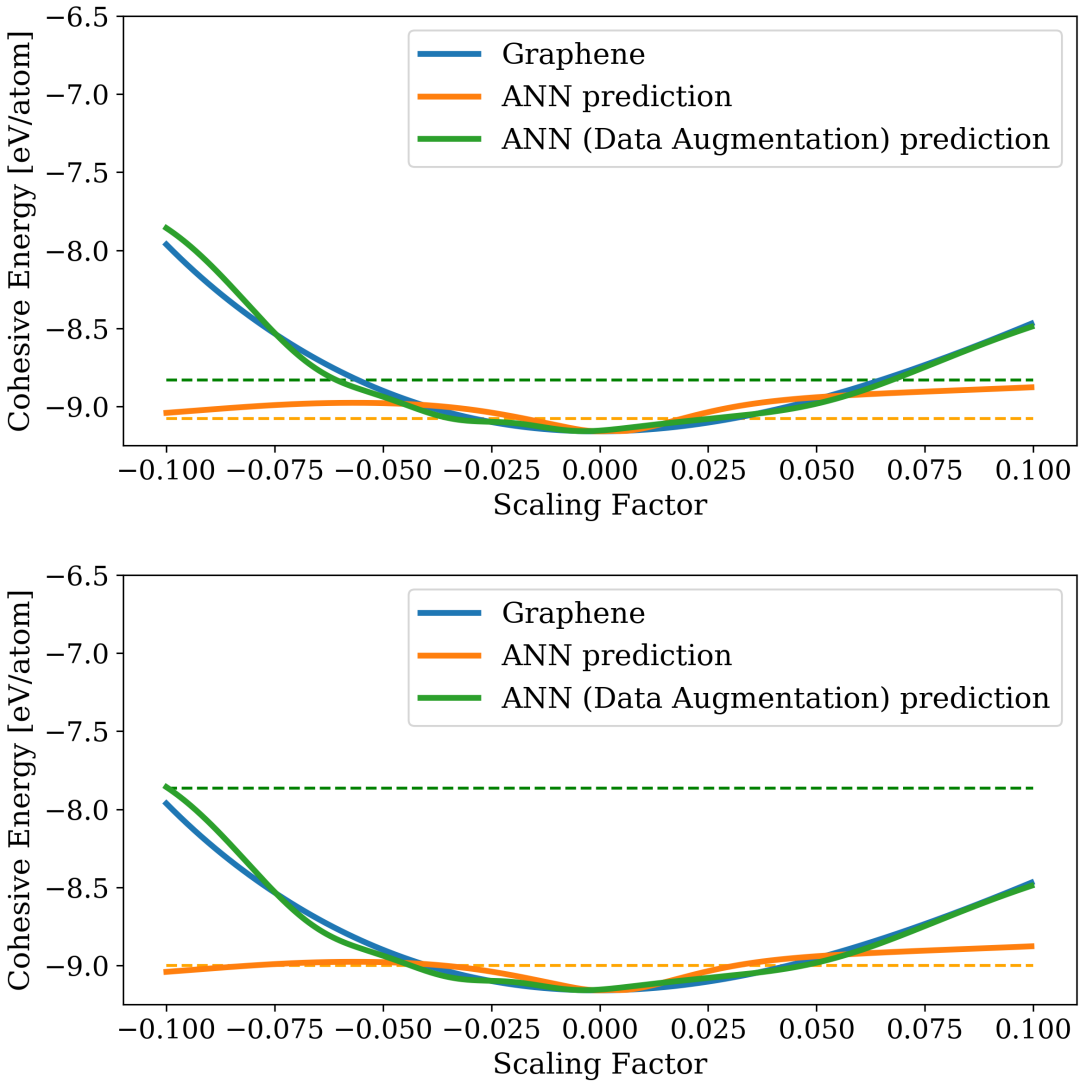

FIGURE 5.10: Cohesive energy per atom as a function of scaling factor. The ANN prediction corresponds to a network trained on MD data only (dataset A). The ANN (Data Augmentation) prediction corresponds to the augmented dataset (dataset B). **Top:** The mean value of cohesive energies per atom in the dataset are plotted for dataset A (orange line) and dataset B (green line). **Bottom:** The maximum value of cohesive energy per atom in the training set is plotted for dataset A (orange line) and dataset B (green line).

Another point worth mentioning is that there is a large gap between the mean and the maximum energies in the augmented reference data. This suggests that even if most of the dataset samples low energy configurations, sampling some data at high energy configurations can help improve the general fit.

Now that we have seen how the potential trained on the dataset B compares to the potential trained on dataset A, let us look at how the dataset B performs overall. In this first test, the potential trained on dataset B is used to make predictions on separate datasets. The first dataset consists entirely of augmented structures (referred to as modified structures). The second dataset consists entirely of MD data (referred to as original structures). These datasets are made up of structures that are independent of training and testing datasets.

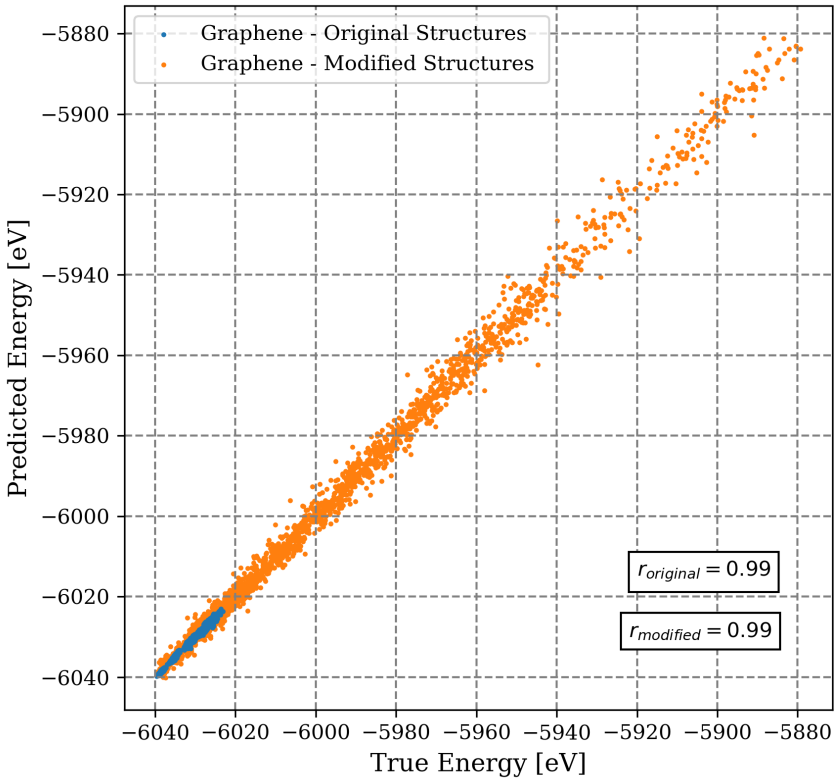

FIGURE 5.11: The network trained on dataset B is used to predict energies of non-augmented data (blue) and augmented data (orange)

In figure 5.11, the predicted and true energies are plotted against one another

for each dataset. The modified structures span a much larger range of energies than the original structures. However, they are further away from a perfect fit than the original structures.

To quantify this, the squared errors are taken between the predicted energies and the true energies. These errors are then represented as a histogram to plot the frequency at which each squared error value occurs. To enhance the differences at small values, the squared error is expressed on a logarithmic scale. Finally, the mean squared error for each dataset is plotted as a vertical line. In figure 5.12, we see that the mean squared error for the original structures is approximately an order of magnitude better than that of the modified structures. This implies that while training the neural network, the testing error computed on the augmented dataset will be much higher than the actual error on the structures most commonly encountered in MD. Specifically, the testing loss on the augmented dataset is not a good predictor of how the network will perform in MD, since the error is averaged over structures that do not generally come up in MD.

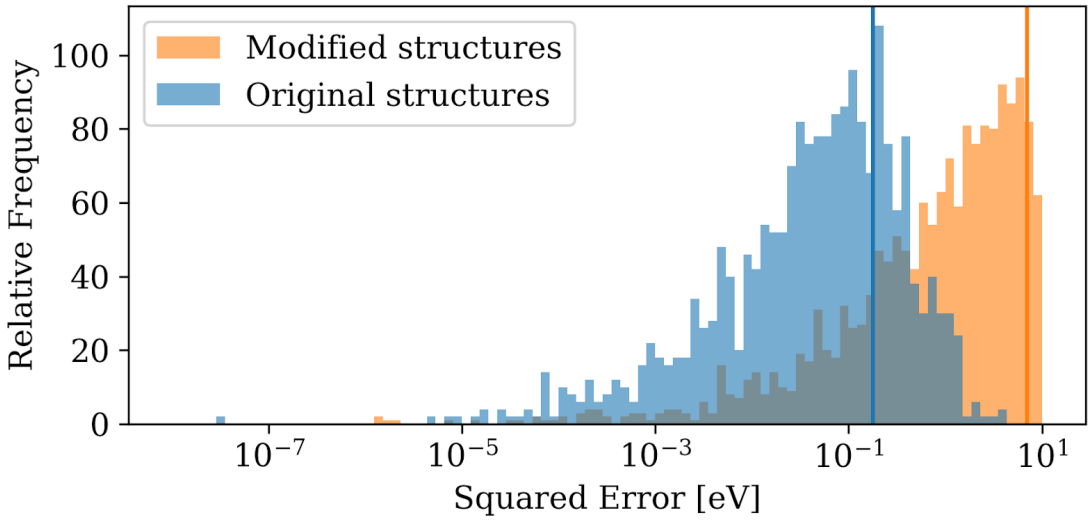

FIGURE 5.12: The squared error of the scaled and unscaled data sets are compared. The frequency of the unscaled data is normalized to the number of examples for the scaled data i.e. the counts for the unscaled data were doubled since there were 2000 modified structures and 1000 original structures.

While the NNP predicts relatively well on the energies, there is low correlation between predicted and true forces for the modified structures. In figure

5.13, the predicted vs. true errors are plotted for the predictions of the forces on the original structures and augmented structures separately. In the top figure, we see that the network generally predicts the forces on the original structures. In the bottom figure, we see that the forces are greatly over-predicted by the network. If we look again at figure 5.10, we can see that the neural network fit winds around the curve of true energies. Therefore, the derivative is constantly changing direction. Since the derivative of the potential gives us the force, it is not surprising that there is a low correlation between the predicted and true energies for the modified graphene structures. Since MD simulations require correct forces to integrate the equations of motion, the derivatives of neural network potential must be accurate enough for MD. It is not sufficient to generally learn the energies i.e. predictions on augmented data will yield unphysical forces.

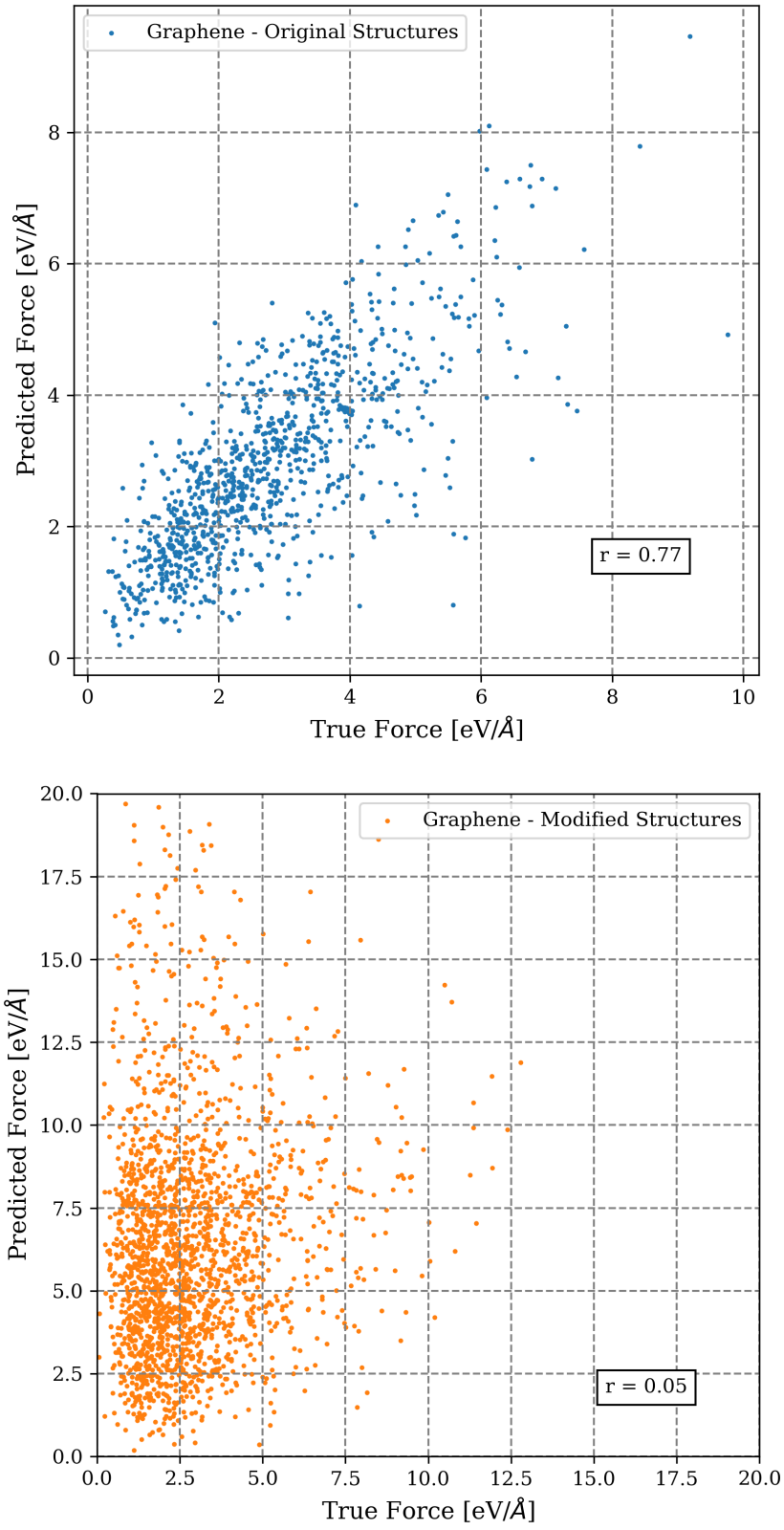

FIGURE 5.13: Force predictions by the neural network trained on dataset B (augmented dataset). **Top:** Forces are predicted on original structures. There is some correlation between predicted and true energies. **Bottom:** Forces are predicted on the modified structures. There is very little correlation between predicted and true values. This is expected as derivatives of the network potential away from equilibrium are not learned by the network.

In figure 5.13, we see that there is a correlation between predicted and true forces for the predictions made on MD data using a network trained on dataset B. With a pearson correlation coefficient of $r = 0.77$, we see that the predicted and target forces are correlated, but not as strongly correlated as the energies (pearson correlation coefficient of $r = 0.99$). Here we will compare the force predictions two different neural network potentials for graphene. The neural networks are either trained on the augmented dataset, or on MD structures only. Both neural networks will make predictions on MD data only.

In figure 5.14, we see that the neural network trained on MD structures only has a very strong correlation ($r = 0.97$) between predicted and target quantities. This result indicates that the neural network potential is capable of making a good prediction on the derivative of the potential energy surface. We note that by training over a larger spread of energies, the force predictions may be less accurate for the network trained on dataset B. In this case, the target function is more complicated as it has to include regions of high nonlinearity. To improve the force predictions by the neural network trained on dataset B, more training examples may be required, and potentially an increase in network capacity.

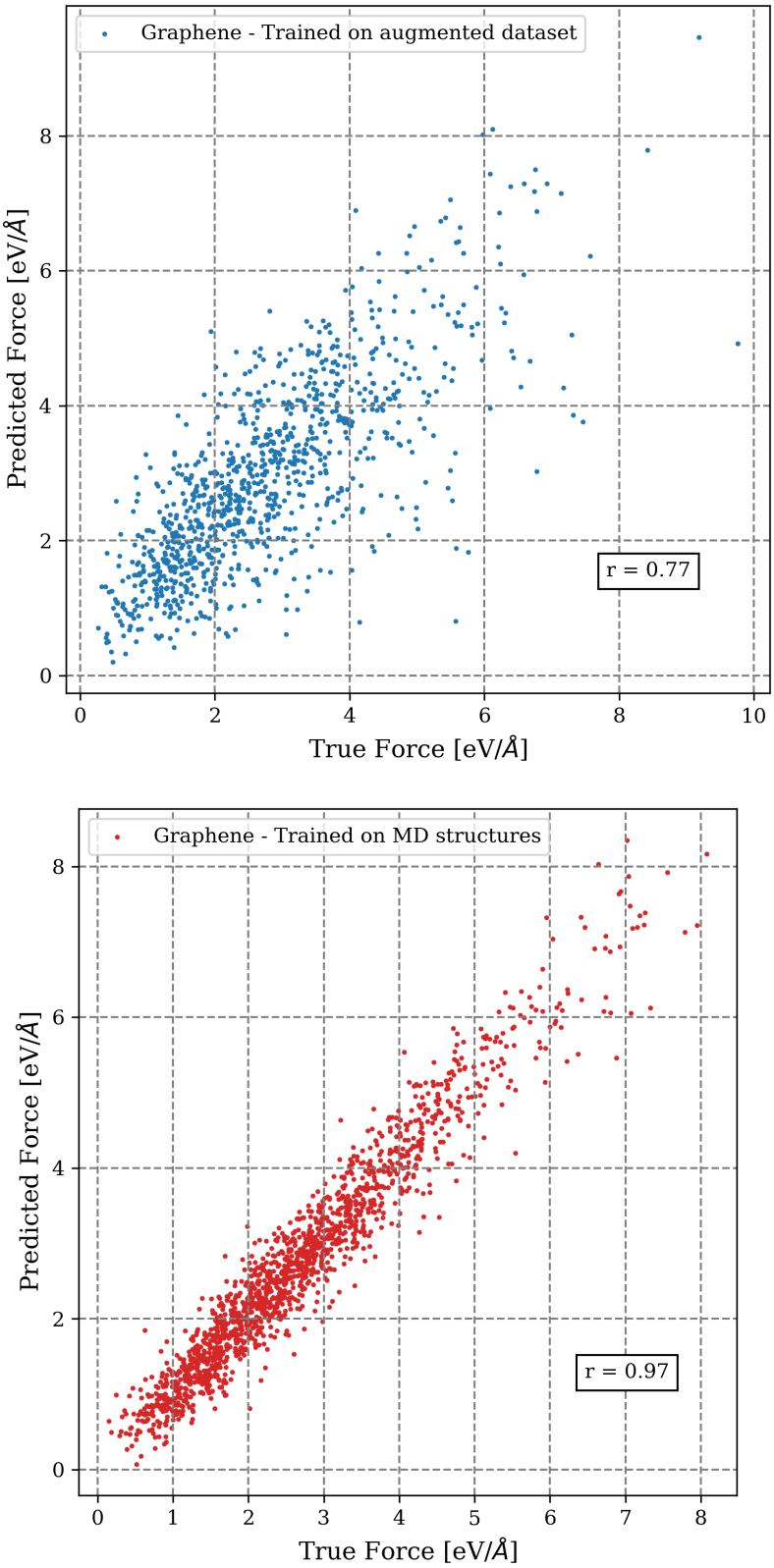

FIGURE 5.14: Force predictions by the neural network trained on dataset B (augmented graphene dataset) compared to predictions made by neural network trained on dataset A (MD graphene data). **Top:** Forces predictions made by the potential trained on dataset B. **Bottom:** Force predictions made by the potential trained on dataset A.

5.3 Comparing 2D materials

Comparing the loss curves of all 2D materials (Figure 5.15), graphene was found to converge to the highest error, and hBN was found to converge the fastest to the lowest error. To explore why graphene converged to a higher error than

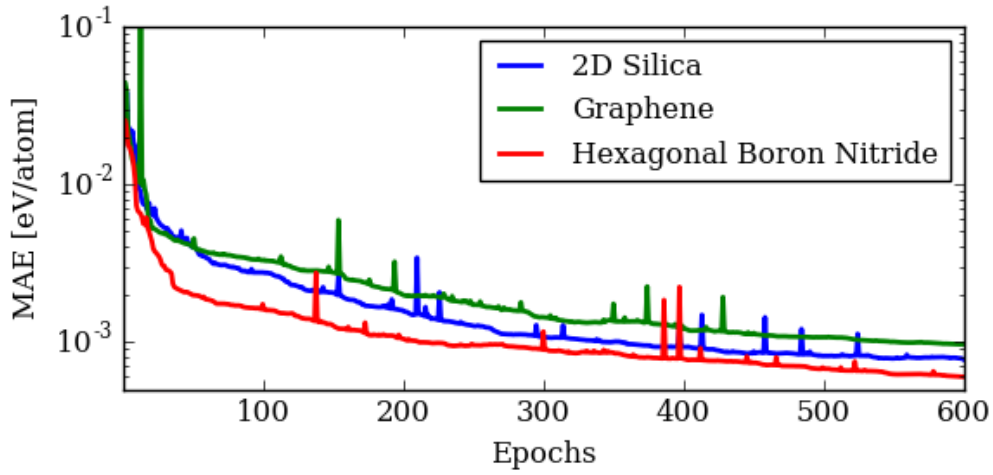

FIGURE 5.15: Loss curves for all 2D materials (datasets are sampled from MD trajectories). A network architecture of 70-10-10-1 is optimized for hexagonal boron nitride and 2D silica using l-BFGS. A network architecture of 26-10-10-1 is optimized for graphene using l-BFGS.

the other materials, we compare the cohesive energy scaling tests of previous materials. To get a clearer idea of how much nonlinearity there is in each curve, the rate of change of cohesive energy is taken with respect to scaling factor. In figure 5.16, we see that the rate of change near the origin is the steepest for graphene. By looking near the origin, we restrict our view to the part of the curve which is most frequently sampled during MD. The steepness of the curve around the origin indicates that there is more nonlinearity between the energies and interatomic distances for graphene than for 2D silica or hexagonal boron nitride. This interpretation, that the nonlinearity increases the difficulty of the potential energy surface, is supported throughout this work.

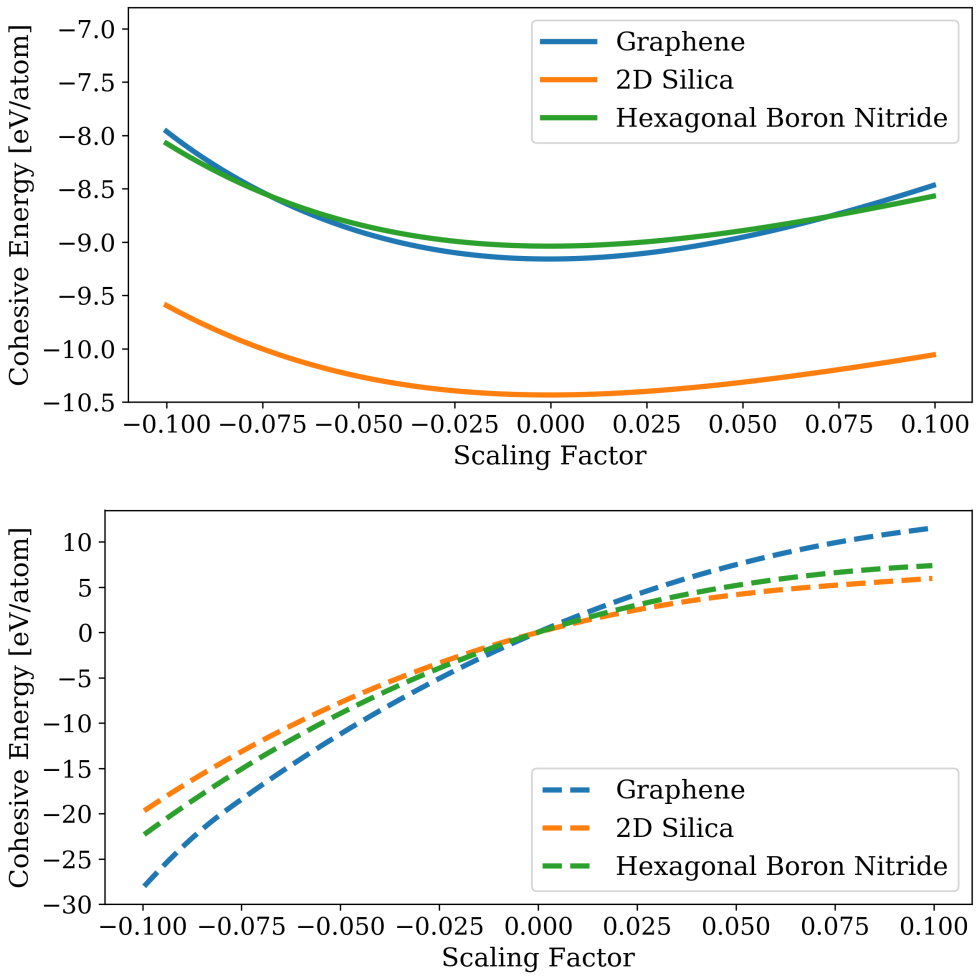

FIGURE 5.16: **Top:** Cohesive energy per atom as a function of scaling for each 2D material. **Bottom:** The rate of change of cohesive energy per atom as a function of scaling factor for each 2D material.

5.3.1 Chapter Summary

- Enforcing close encounters of atoms by maintaining a smaller box size (increasing the pressure) increases nonlinearity between energies and atomic coordinates.
- Training on a dataset that spans a larger range of energies may improve general shape of the neural network predictions, however it may not sufficiently learn the forces of augmented structures.

- Temperature sampling alone may not result in as much structural change for some materials compared to others (eg. hBN, cBN)
- Some materials have a more nonlinear potential energy surface compared to others eg. diamond compared to graphene.

Chapter 6

Recommendations

We have looked at how sampling and data augmentation affects the quality of the neural network fit and extrapolation capabilities. We have also explored the added difficulty of learning more complex regions of the potential energy surface. Here we will provide some general recommendations for generating a neural network potential and discuss some suggestions for improving the quality of the model.

6.1 Sampling

The entirety of a high dimensional potential energy surface cannot be evaluated. For this reason, we must identify the regions of configuration space that are most important. To get the basic features of the potential energy surface, one should implement temperature sampling near equilibrium. By thermally distorting ideal structures, the ideal bond lengths and lattice constants can be obtained. We know that the most frequently visited energy states occur around the minima, so it is important that this be sufficiently represented in the training data.

In this work, we sampled the canonical ensemble at various temperatures using MD simulations. To avoid correlations in the training data, the MD trajectories should be sampled using a sufficiently large window. To cheaply generate structures using a large sampling window, force fields methods can be implemented. A higher level calculation (eg. *ab-initio*, semi-empirical) can be performed on a set of structures obtained using force fields simulations. Structures can be added to the training set until the neural network potential converges below the target error (we used a target error of 5 [meV/atom]). Note that it

may not be sufficient to only look at how well the network makes predictions on energies. The predictions of the network on forces will be a better indicator of how well the neural network potential will perform in an MD simulation.

Once a neural network potential has achieved an acceptable error, it can be used to generate new structures to add to the training data. If the network is failing to predict certain configurations, the target quantities can be computed to correct the predictions made by the neural network. These new configurations can then be provided as new examples in the training set. Since generating a large reference dataset is expensive, this has the added benefit of being able to cheaply evaluate the neural network potential to generate new configurations, instead of implementing an MD simulation that evaluates a computationally expensive potential.

6.1.1 Training the Network

Symmetry function parameters determine how sensitive the network will be over corresponding distances. As suggested by Behler [11], the highest η value for the radial symmetry functions should correspond to the shortest bond length in the system and symmetry functions should span configuration space in an unbiased way. Generally, as the number of basis functions increase, so does the accuracy. Using a large number of basis functions, however, greatly increases the computational cost.

Choosing the best network architecture for the reference data depends on the complexity of the problem and varies between datasets. Choosing a network architecture that is too small can result in a model that does not have enough capacity to describe the problem. Increasing the number of hidden layers in neurons may increase the capacity for a network to learn, but it also increases the computational cost of training and evaluating the network. If the network has too many parameters, there is also the risk of overfitting to the training data. The optimal network architecture will balance having enough model capacity as well as being compact enough to reduce the computational cost associated with training and evaluating the network. Various architectures should be tested to determine which architecture is optimal.

In this work, the l-BFGS optimizer was used as it consistently converged to

a low error (almost independently of random seed) and did not require optimization of the learning rate parameter. In general, the online gradient descent method is the least computationally demanding of the methods. In this work we found it to converge to the highest error compared to the other two optimizers available in ænet. Artrith et al. [3] found that the Levenberg-Marquardt method tends to converge faster with respect to training iterations than the l-BFGS method. It may be efficient for small datasets, but becomes computationally demanding for large datasets and network architectures due to the inversion of the Hessian matrix. For this reason, the l-BFGS method (which approximates the Hessian matrix) is a good choice for large datasets or network architectures. If using the online gradient descent method or the Levenberg-Marquard method, learning rate parameter should be tested to achieve the lowest possible error. The gradient descent and Levenberg-Marquardt methods were also shown to have some sensitivity to random seeds. It would be useful to run the optimization more than once to converge to the lowest possible error when using these optimization schemes.

6.1.2 Extrapolation Capabilities

As discussed in Behler's review on constructing high-dimensional neural network potentials [12], since configuration space cannot be completely sampled, failures of the neural network potential will not necessarily manifest in a high testing error. If regions of configuration space are not represented in the reference data, we cannot adjust the model to be predictive there. This is a consequence of the complexity of a high dimensional potential energy surface. The dimensionality of the system is given by 3N-6 where N is the number of atoms in the system (3 degrees of freedom for each atom minus the total translation and rotation of the system). Thus, as the number of atoms increase, the more complicated the function will be to model.

Behler addresses this using two different methods. In the first approach, the minimum and maximum values of the symmetry functions used during training define an interval upon which the network is likely to perform best. When new structures are presented to the neural network and mapped onto symmetry functions, the minimum and maximum values of these symmetry functions can be compared to the interval upon which the network is likely

to perform best. If the minimum and maximum symmetry function values lie outside of the range of this interval, then a warning should be issued that the neural network may not make an accurate prediction. The identified structures can then be added to the training data to improve the applicability of the neural network potential.

A second approach, suggested by Behler [12], is to identify unphysical structures using multiple neural network potentials. Here, multiple neural networks are trained on the same training data. The only difference between the neural network potentials are that they are trained on different architectures. The networks should be of comparable quality eg. the network should have enough capacity to describe the fit. Once the networks are trained, they can be used in MD to generate new configurations. The same simulation conditions should be maintained to ensure that the results are comparable.

Once the configurations are obtained, each neural network should be used to make predictions on all of the configurations. If all of the neural network potentials make similar predictions of the target quantities on the configurations, there is likely a sufficient amount of training data sampled from this region of configuration space. If predictions are very different from one another, more training data is required. These configurations can be added to the reference dataset to improve the model.

To continue improving the extrapolation capabilities of the neural network potential, data augmentation can be implemented. This can be achieved through scaling coordinates and lattice vectors as discussed in chapter 5 or adding random fluctuations to the coordinates, for example.

Chapter 7

Conclusions and Future Work

In this work we used feed-forward artificial neural networks to interpolate the relationship between local structural environment and atomic energies for six different materials (graphene, diamond, hexagonal boron nitride, cubic boron nitride, 2D silica and bulk silica). We explored how various aspects of the dataset i.e. sampling and baseline complexity of the potential energy surface of the material contributes to the difficulty of the problem. When the training set consists entirely of structures sampled from MD trajectories, we found that the network performs very well (MAE$\sim$0.5-1.0 [meV/atom] compared to a target error of 5 [meV/atom]). We found that this reference dataset was fit to the same degree of accuracy by a linear regression model. Furthermore, it achieved a faster rate of convergence. We then saw that while a network of linear activation functions is almost equivalent to a one layer RBF network, a large number of basis functions are needed to approximate the potential energy surface for regions that are highly nonlinear. Nonlinear activation functions, however, are able to interpolate nonlinear mappings very well on an incomplete basis set. This is especially important for network potentials that are designed for a systems of diverse composition, as the number of basis functions scale with the number of atomic species. Even for a small number of neurons, we see that tanh is able to effectively learn the general shape of the potential energy surface (section 4.1.2).

We then found that augmentation of the dataset significantly increases the complexity of the potential energy surface. Since high energy configurations rarely occur during MD sampling around the minima, the potential energy surface interpolated from a dataset containing only MD data will not have a high degree of nonlinearity. It is therefore sufficient to use a network of linear activations to interpolate the potential energy surface. In order to generate a robust

potential, however, rare events must be explicitly accounted for. We saw that on an augmented dataset (TiO_2) the network using linear activation functions had a much higher error ($\sim$ 17.5 [meV/atom]) compared to the error from the network using tanh activation functions ($\sim$ 8.5 [meV/atom]). This indicates that for more complex potential energy surfaces with a high degree of nonlinearity, there is a significant benefit to using a network of nonlinear activation functions.

Following this analysis on activation functions and potential energy surface complexity, the baseline complexity of the potential energy surface for each material was tested. This was tested by computing the cohesive energy per atom as a function of scaling factor used to compress or expand the system around the equilibrium structure. Here we saw that sampling away from equilibrium (2D silica) led to a more difficult potential energy surface for the network to learn. Moving away from the energy minima results in more nonlinearity of the potential energy surface curve. We then turned our attention to graphene and diamond which both converged to higher errors than all other materials. Our scaling experiments suggest that diamond has a highly nonlinear potential energy surface, and that graphene has the most nonlinear potential energy surface compared to all other 2D materials.

We learned that the neural network trained on the augmented graphene dataset was able to fit the curve of cohesive energy per atom with respect to scaling factor quite well. It failed to make any meaningful predictions on the forces of the augmented structures, although it predicted well on the MD structures. We also noted that while this network potential converged to a high MAE overall, the errors of predictions made on structures obtained from MD were an order of magnitude better than the errors of predictions made on structures that were augmented.

Lastly, from the convergence of the error on the boron nitride networks, we see that cubic boron nitride converges to a lower loss but predicts less well on structures that are not close to the equilibrium configuration. This is likely the result of less deviation from the equilibrium state as thermal energy is added to the system than for hexagonal boron nitride.

Naturally, the next steps for this work would involve refining the neural network potentials trained on an augmented dataset. To refine the potentials, as suggested by Artrith et al [3], a preliminary neural network potential could be

used to generate structures drawn from molecular dynamics trajectories. These trajectories can be used to run a higher level DFTB calculation and add structures into the dataset iteratively. Alternatively, configuration space could be searched using a genetic algorithm as implemented by Artrith et al [4], as this method has been shown to be competitive with the traditional ænet method but uses significantly fewer training examples.

Ultimately, machine learned potentials are a promising approach for accelerating materials simulations. Due to their flexible functional form, they can learn complex high dimensional potential energy surfaces. Neural network potentials can be as accurate as the underlying method which was used to generate the reference data, but they are significantly cheaper to evaluate. This allows us to do materials simulations at time scales and system sizes we previously could not access (unless we made approximations and fit to reproduce specific quantities), but at the same level of accuracy of a high level quantum mechanics method.

Bibliography

[1] Balint Aradi, Ben Hourahine, and Th Frauenheim. "DFTB+, a sparse matrix-based implementation of the DFTB method". In: *The Journal of Physical Chemistry A* 111.26 (2007), pp. 5678–5684.

[2] Nongnuch Artrith and Alexie M Kolpak. "Understanding the composition and activity of electrocatalytic nanoalloys in aqueous solvents: A combination of DFT and accurate neural network potentials". In: *Nano letters* 14.5 (2014), pp. 2670–2676.

[3] Nongnuch Artrith and Alexander Urban. "An implementation of artificial neural-network potentials for atomistic materials simulations: Performance for TiO2". In: *Computational Materials Science* 114.Supplement C (2016), pp. 135 –150. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2015.11.047. URL: http://www.sciencedirect.com/science/article/pii/S0927025615007806.

[4] Nongnuch Artrith, Alexander Urban, and Gerbrand Ceder. "Constructing first-principles phase diagrams of amorphous LixSi using machine-learning-assisted sampling with an evolutionary algorithm". In: *The Journal of Chemical Physics* 148.24 (2018), p. 241711. DOI: 10.1063/1.5017661. eprint: https://doi.org/10.1063/1.5017661. URL: https://doi.org/10.1063/1.5017661.

[5] Nongnuch Artrith, Alexander Urban, and Gerbrand Ceder. "Efficient and accurate machine-learning interpolation of atomic energies in compositions with many species". In: *Phys. Rev. B* 96 (1 2017), p. 014112. DOI: 10.1103/PhysRevB.96.014112. URL: https://link.aps.org/doi/10.1103/PhysRevB.96.014112.

[6] Albert P Bartok, Risi Kondor, and Gabor Csanyi. "Publisher's Note: On representing chemical environments [Phys. Rev. B 87, 184115 (2013)]". In: *Physical Review B* 87.21 (2013), p. 219902.

[7] Albert P Bartók et al. "Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons". In: *Physical review letters* 104.13 (2010), p. 136403.

[8] Axel D. Becke. "Density-functional thermochemistry. III. The role of exact exchange". In: *The Journal of Chemical Physics* 98.7 (1993), pp. 5648–5652. DOI: 10.1063/1.464913. eprint: https://doi.org/10.1063/1.464913. URL: https://doi.org/10.1063/1.464913.

[9] Jörg Behler. "Atom-centered symmetry functions for constructing high-dimensional neural network potentials". In: *The Journal of chemical physics* 134.7 (2011), p. 074106.

[10] Jörg Behler and Michele Parrinello. "Generalized neural-network representation of high-dimensional potential-energy surfaces". In: *Physical review letters* 98.14 (2007), p. 146401.

[11] Jörg Behler. "Constructing high-dimensional neural network potentials: A tutorial review". In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1032–1050. ISSN: 1097-461X. DOI: 10.1002/qua.24890. URL: http://dx.doi.org/10.1002/qua.24890.

[12] Jörg Behler. "Perspective: Machine learning potentials for atomistic simulations". In: *The Journal of Chemical Physics* 145.17 (2016), p. 170901. DOI: 10.1063/1.4966192. eprint: https://doi.org/10.1063/1.4966192. URL: https://doi.org/10.1063/1.4966192.

[13] Torbjörn Björkman et al. "Defects in bilayer silica and graphene: common trends in diverse hexagonal two-dimensional systems". In: *Scientific reports* 3 (2013), p. 3482.

[14] S Francis Boys. "Electronic wave functions-I. A general method of calculation for the stationary states of any molecular system". In: *Proc. R. Soc. Lond. A* 200.1063 (1950), pp. 542–554.

[15] David FR Brown, Mark N Gibbs, and David C Clary. "Combining ab initio computations, neural networks, and diffusion Monte Carlo: An efficient method to treat weakly bound molecules". In: *The Journal of chemical physics* 105.17 (1996), pp. 7597–7604.

[16] Weiwei Cai et al. "Thermal transport in suspended and supported monolayer graphene grown by chemical vapor deposition". In: *Nano letters* 10.5 (2010), pp. 1645–1651.

[17] Data camp. [Online; accessed June 27, 2018]. URL: https://www.datacamp.com/community/tutorials/deep-learning-python.

[18] C. Crespos et al. "Multi-dimensional potential energy surface determination by modified Shepard interpolation for a molecule–surface reaction: H2+Pt(111)". In: *Chemical Physics Letters* 376.5 (2003), pp. 566 –575. ISSN: 0009-2614. DOI: https://doi.org/10.1016/S0009-2614(03)01033-9. URL: http://www.sciencedirect.com/science/article/pii/S0009261403010339.

[19] Balázs Csanád Csáji. "Approximation with artificial neural networks". In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24 (2001), p. 48.

[20] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[21] Richard Dawes et al. "Interpolating moving least-squares methods for fitting potential energy surfaces: A strategy for efficient automatic data point placement in high dimensions". In: *The Journal of chemical physics* 128.8 (2008), p. 084107.

[22] Richard Dawes et al. "Interpolating moving least-squares methods for fitting potential energy surfaces: Computing high-density potential energy surface data from low-density ab initio data points". In: *The Journal of chemical physics* 126.18 (2007), p. 184108.

[23] M Elstner. "The SCC-DFTB method and its application to biological systems". In: *Theoretical Chemistry Accounts* 116.1-3 (2006), pp. 316–325.

[24] Marcus Elstner and Gotthard Seifert. "Density functional tight binding". In: *Phil. Trans. R. Soc. A* 372.2011 (2014), p. 20120483.

[25] Marcus Elstner et al. "Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties". In: *Physical Review B* 58.11 (1998), p. 7260.

[26] Mazran Esro et al. "Structural and electrical characterization of SiO2 gate dielectrics deposited from solutions at moderate temperatures in air". In: *ACS applied materials & interfaces* 9.1 (2016), pp. 529–536.

[27] Clement Faugeras et al. "Thermal conductivity of graphene in corbino membrane geometry". In: *ACS nano* 4.4 (2010), pp. 1889–1892.

[28] David Feller. "Benchmarks of improved complete basis set extrapolation schemes designed for standard CCSD (T) atomization energies". In: *The Journal of chemical physics* 138.7 (2013), p. 074103.

[29] David Feller, Kirk A. Peterson, and J. Grant Hill. "Calibration study of the CCSD(T)-F12a/b methods for C2 and small hydrocarbons". In: *The Journal of Chemical Physics* 133.18 (2010), p. 184102. DOI: 10.1063/1.3491809. eprint: https://doi.org/10.1063/1.3491809. URL: https://doi.org/10.1063/1.3491809.

[30] David Feller, Kirk A. Peterson, and J. Grant Hill. "On the effectiveness of CCSD(T) complete basis set extrapolations for atomization energies". In: *The Journal of Chemical Physics* 135.4 (2011), p. 044102. DOI: 10.1063/1.3613639. eprint: https://doi.org/10.1063/1.3613639. URL: https://doi.org/10.1063/1.3613639.

[31] Ken-Ichi Funahashi. "On the approximate realization of continuous mappings by neural networks". In: *Neural networks* 2.3 (1989), pp. 183–192.

[32] Michael Gaus, Qiang Cui, and Marcus Elstner. "DFTB3: extension of the self-consistent-charge density-functional tight-binding method (SCC-DFTB)". In: *J. Chem. Theory Comput* 7.4 (2011), pp. 931–948.

[33] Peter MW Gill. "Molecular integrals over Gaussian basis functions". In: *Advances in quantum chemistry*. Vol. 25. Elsevier, 1994, pp. 141–205.

[34] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[35] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[36] J. Grant Hill et al. "Extrapolating MP2 and CCSD explicitly correlated correlation energies to the complete basis set limit with first and second row correlation consistent basis sets". In: *The Journal of Chemical Physics* 131.19 (2009), p. 194105. DOI: 10.1063/1.3265857. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.3265857. URL: https://aip.scitation.org/doi/abs/10.1063/1.3265857.

[37] Pierre Hohenberg and Walter Kohn. "Inhomogeneous electron gas". In: *Physical review* 136.3B (1964), B864.

[38] William G Hoover. "Nonequilibrium molecular dynamics". In: *Annual Review of Physical Chemistry* 34.1 (1983), pp. 103–127.

[39] Josef Ischtwan and Michael A Collins. "Molecular potential energy surfaces by interpolation". In: *The Journal of chemical physics* 100.11 (1994), pp. 8080–8088.

[40] Josef Ischtwan and Michael A Collins. "Molecular potential energy surfaces by interpolation". In: *The Journal of chemical physics* 100.11 (1994), pp. 8080–8088.

[41] Masa Ishigami et al. "Atomic structure of graphene on SiO2". In: *Nano letters* 7.6 (2007), pp. 1643–1648.

[42] Anubhav Jain et al. "The Materials Project: A materials genome approach to accelerating materials innovation". In: *APL Materials* 1.1 (2013), p. 011002. ISSN: 2166532X. DOI: 10.1063/1.4812323. URL: http://link.aip.org/link/AMPADS/v1/i1/p011002/s1\&Agg=doi.

[43] Quasar Jarosz. [Online; accessed June 27, 2018]. URL: https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg.

[44] Christof Köhler et al. "Theoretical investigation of carbon defects and diffusion in α-quartz". In: *Physical Review B* 64.8 (2001), p. 085333.

[45] Walter Kohn and Lu Jeu Sham. "Self-consistent equations including exchange and correlation effects". In: *Physical review* 140.4A (1965), A1133.

[46] Pekka Koskinen and Ville Mäkinen. "Density-functional tight-binding for beginners". In: *Computational Materials Science* 47.1 (2009), pp. 237–253.

[47] David C Langreth and MJ Mehl. "Beyond the local-density approximation in calculations of ground-state electronic properties". In: *Physical Review B* 28.4 (1983), p. 1809.

[48] Ask Hjorth Larsen et al. "The atomic simulation environment—a Python library for working with atoms". In: *Journal of Physics: Condensed Matter* 29.27 (2017), p. 273002. URL: http://stacks.iop.org/0953-8984/29/i=27/a=273002.

[49] Yann LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[50] Yann LeCun et al. "Generalization and network design strategies". In: *Connectionism in perspective* (1989), pp. 143–155.

[51] Changgu Lee et al. "Measurement of the elastic properties and intrinsic strength of monolayer graphene". In: *science* 321.5887 (2008), pp. 385–388.

[52] Jae-Ung Lee et al. "Thermal conductivity of suspended pristine graphene measured by Raman spectroscopy". In: *Physical Review B* 83.8 (2011), p. 081419.

[53] Sönke Lorenz, Axel Groß, and Matthias Scheffler. "Representing high-dimensional potential-energy surfaces for reactions at surfaces by neural networks". In: *Chemical Physics Letters* 395.4-6 (2004), pp. 210–215.

[54] Binit Lukose et al. "On the reticular construction concept of covalent organic frameworks". In: *Beilstein journal of nanotechnology* 1 (2010), p. 60.

[55] Gia G Maisuradze et al. "Interpolating moving least-squares methods for fitting potential energy surfaces: Detailed analysis of one-dimensional applications". In: *The Journal of chemical physics* 119.19 (2003), pp. 10002–10014.

[56] Andrei Malashevich, Sohrab Ismail-Beigi, and Eric I Altman. "Directing the Structure of Two-Dimensional Silica and Silicates". In: *The Journal of Physical Chemistry C* 120.47 (2016), pp. 26770–26781.

[57] Hegoi Manzano et al. "Benchmark of ReaxFF force field for subcritical and supercritical water". In: *The Journal of Chemical Physics* 148.23 (2018), p. 234503.

[58] Sergei Manzhos et al. "A nested molecule-independent neural network approach for high-quality potential fits". In: *The Journal of Physical Chemistry A* 110.16 (2006), pp. 5295–5304.

[59] Richard M Martin. *Electronic structure: basic theory and practical methods*. Cambridge university press, 2004.

[60] Hrushikesh Mhaskar, Qianli Liao, and Tomaso A Poggio. "When and why are deep networks better than shallow ones?" In: *AAAI*. 2017, pp. 2343–2349.

[61] Gaus Michael, Cui Qiang, and Elstner Marcus. "Density functional tight binding: application to organic and biological molecules". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 4.1 (), pp. 49–61. DOI: 10.1002/wcms.1156. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1156. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1156.

[62] PB Mirkarimi, KF McCarty, and DL Medlin. "Review of advances in cubic boron nitride film synthesis". In: *Materials Science and Engineering: R: Reports* 21.2 (1997), pp. 47–100.

[63] Hendrik J. Monkhorst and James D. Pack. "Special points for Brillouin-zone integrations". In: *Phys. Rev. B* 13 (12 1976), pp. 5188–5192. DOI: 10.1103/PhysRevB.13.5188. URL: https://link.aps.org/doi/10.1103/PhysRevB.13.5188.

[64] Grégoire Montavon and Klaus-Robert Müller. "Deep Boltzmann machines and the centering trick". In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 621–637.

[65] Robert S Mulliken. "Electronic population analysis on LCAO–MO molecular wave functions. I". In: *The Journal of Chemical Physics* 23.10 (1955), pp. 1833–1840.

[66] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[67] Kyoung Tai No et al. "Description of the potential energy surface of the water dimer with an artificial neural network". In: *Chemical physics letters* 271.1-3 (1997), pp. 152–156.

[68] Shyue Ping Ong et al. "Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis". In: *Computational Materials Science* 68 (Feb. 2013), pp. 314–319. ISSN: 09270256. DOI: 10.1016/j.commatsci.2012.10.028. URL: http://linkinghub.elsevier.com/retrieve/pii/S0927025612006295.

[69] Leo Paradis et al. *Diamond heat sink*. US Patent App. 10/114,601. 2003.

[70] Jooyoung Park and Irwin W Sandberg. "Universal approximation using radial-basis-function networks". In: *Neural computation* 3.2 (1991), pp. 246–257.

[71] John P. Perdew, Kieron Burke, and Matthias Ernzerhof. "Generalized Gradient Approximation Made Simple". In: *Phys. Rev. Lett.* 77 (18 1996), pp. 3865–3868. DOI: 10.1103/PhysRevLett.77.3865. URL: https://link.aps.org/doi/10.1103/PhysRevLett.77.3865.

[72] Dirk Porezag et al. "Construction of tight-binding-like potentials on the basis of density-functional theory: Application to carbon". In: *Physical Review B* 51.19 (1995), p. 12947.

[73] Frederico V Prudente and JJ Soares Neto. "The fitting of potential energy surfaces using neural networks. Application to the study of the photodissociation processes". In: *Chemical physics letters* 287.5-6 (1998), pp. 585–589.

[74] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), p. 533.

[75] Kevin Ryczko et al. "Convolutional neural networks for atomistic systems". In: *Computational Materials Science* 149 (2018), pp. 134–142.

[76] Franco Scarselli and Ah Chung Tsoi. "Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results". In: *Neural Networks* 11.1 (1998), pp. 15 –37. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00097-X. URL: http://www.sciencedirect.com/science/article/pii/S089360809700097X.

[77] Thomas P Senftle et al. "The ReaxFF reactive force-field: development, applications and future directions". In: *npj Computational Materials* 2 (2016), p. 15011.

[78] J. C. Slater. "Atomic Shielding Constants". In: *Phys. Rev.* 36 (1 1930), pp. 57–64. DOI: 10.1103/PhysRev.36.57. URL: https://link.aps.org/doi/10.1103/PhysRev.36.57.

[79] Li Song et al. "Large scale growth and characterization of atomic hexagonal boron nitride layers". In: *Nano letters* 10.8 (2010), pp. 3209–3215.

[80] Sho Sonoda and Noboru Murata. "Neural network with unbounded activation functions is universal approximator". In: *Applied and Computational Harmonic Analysis* 43.2 (2017), pp. 233 –268. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2015.12.005. URL: http://www.sciencedirect.com/science/article/pii/S1063520315001748.

[81] Bobby G Sumpter and Donald W Noid. "Potential energy surfaces for macromolecules. a neural network technique". In: *Chemical physics letters* 192.5-6 (1992), pp. 455–462.

[82] Jerry Tersoff. "New empirical approach for the structure and energy of covalent systems". In: *Physical Review B* 37.12 (1988), p. 6991.

[83] Laurence Vel, Gerard Demazeau, and Jean Etourneau. "Cubic boron nitride: synthesis, physicochemical properties and applications". In: *Materials Science and Engineering: B* 10.2 (1991), pp. 149–164.

[84] Jingang Wang et al. "Electrical properties and applications of graphene, hexagonal boron nitride (h-BN), and graphene/h-BN heterostructures". In: *Materials Today Physics* 2 (2017), pp. 6 –34. ISSN: 2542-5293. DOI: https://doi.org/10.1016/j.mtphys.2017.07.001. URL: http://www.sciencedirect.com/science/article/pii/S2542529317300597.

[85] Lanhua Wei et al. "Thermal conductivity of isotopically modified single crystal diamond". In: *Physical Review Letters* 70.24 (1993), p. 3764.

[86] Yongjie Zhan et al. "Large-area vapor-phase growth and characterization of MoS2 atomic layers on a SiO2 substrate". In: *Small* 8.7 (2012), pp. 966–971.

[87] Guishan Zheng et al. "Implementation and benchmark tests of the DFTB method and its application in the ONIOM method". In: *International Journal of Quantum Chemistry* 109.9 (2009), pp. 1841–1854.