MOLECULAR GEOMETRY OPTIMIZATION BY ARTIFICIAL NEURAL NETWORKS

BY HE CHEN

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Ahmed Elgammal

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2019

ABSTRACT OF THE THESIS

Molecular Geometry Optimization by Artificial Neural Networks

by He Chen

Thesis Director: Ahmed Elgammal

Artificial neural network is revolutionizing many areas in science and technology. We applied artificial neural network to solve a non-linear optimization problem in computational chemistry, i.e. molecular geometry optimization, which aims to find an atomic arrangement that corresponds to a stationary point on the potential energy surface.

The implemented ANN can use both function values and derivatives as the reference data for training. The relative importance of function values and derivatives is studied extensively. With the same amount data points, ANN trained with derivatives tend to generalize better. With only derivatives as the reference data, the trained ANN can predict function values accurately if a common offset is allowed.

We trained ANNs that can predict molecule energies and gradients fairly well. Molecular geometry optimization is performed on testing data that are never seen during the training process. About 5% of the testing structures converge with average 0.047 Å RMSD compared with equilibrium state, while others diverge. The divergence is ascribed to poor gradient prediction near the equilibrium.

Acknowledgements

Ahmed Elgammal accepted me as his student to work on a Chemistry problem using Computer Science technology. He is a great teacher who taught me Computer Vision. He also provided useful and straightforward suggestions to help move the project forward. I am grateful to have had him as a mentor.

I audited computational geometry by William Steiger, who struck me as a knowledgeable and nice scholar. We had some talks about research problems from time to time. Pranjal Awasthi taught me Machine Learning Theory, which was a theoretical dense course and I loved it! I thank both William and Pranjal for all the awesome lectures and investing time in me despite their busy schedules.

I owe many thanks to David A. Case, my advisor in Chemistry. He provided valuable advice during this thesis. Besides his great academic capability, his passion in science, extreme self-discipline and patience also impressed me a lot and boosted my self-esteem significantly. Deep in heart, one can not ask for a better mentor.

At last, love to my parents, my brother, and my girl friend. They are the ones I could always share my feelings with, either good or bad.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Conventional Machine Learning

Machine learning (ML) refers to giving computers the ability to learn without being explicitly programmed[3]. Nowadays, ML technology is increasingly present in many aspects of modern society: from online web search, intelligent recommendation to data security, financial trading, and healthcare. In principle, ML learns a function that maps inputs to outputs. It's typically classified into two broad categories, supervised learning and unsupervised learning, depending on whether there is a "target" value available to the learning system[4].

One simple example of a traditional machine learning algorithm is the linear least squares regression, in which the input are linearly independent to each other and the underlying nature of the model is assumed to be known. The input, also known as features, are generally dervied from the original measured data by the domain experts. This feature extraction step intends to facilitate the subsequent learning and generalization steps by precisely balancing a set of input features to produce an output. The limitation of conventional ML systems lies in that it requires careful engineering and considerable domain expertise to design a feature extractor that transforms the raw data into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input[5].

1.2 Artificial Neural Networks

Artificial neural networks (ANNs) are a class of machine learning algorithm inspired by biological neural networks, used to approximate functions by translating a large

number of inputs into output. ANNs are constructed from a series of layers, which are based on a collection of connected nodes called artificial "neurons". Each neuron accepts the signals from the previous layer, processes them by maps the signal onto a non-linear function. The power of ANNs lies in their ability to make multiple non-linear transformations through many hidden layers of neurons. It allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction[5]. In this process, increasingly complex and abstract features can be constructed, through the addition of more layers and/or increasing the width of the layers. The "training" process of ANNs corresponds to tuning the internal parameters, that are used to compute the representation in each layer from the representation in the previous layer, to minimize a cost function by applying the back–propagation algorithm[6, 7]. In principle, ANNs are able to approximate any real–valued function with arbitrary accuracy[8, 9].

ANNs have turned out to be very good at discovering intricate structures in high dimentional data and are revolutionizing many areas of science and technology, particularly in image recognition[10, 11, 12, 13], speech recognition[14, 15, 16], and natural language understanding[17]. Areas like sentiment analysis, question answering[18] and language translation[19, 20] are dominated by ANNs, which produce incredibly promising results. Adding a further point, ANNs outperformed conventional ML methods at predicting the effects of mutations on gene expression and disease[21, 22], discorvering potential drug molecules[23], and reconstructing brain circuits[24].

1.3 Computational Chemistry

Computational chemistry is a branch of chemistry that uses computer simulation to assist in solving chemical problems. It applies methods of theoretical chemistry, incorporated into efficient computer programs, to calculate the structures and properties of molecules and solids.

The classical methods, *ab initio* electronic structure methods, attempt to solve the

electric Schrödinger equation given the positions of the nuclei and the number of electrons in order to yield useful information such as electron densities, energies and other properties of the system[25]. These kind of methods have the advantage that they can be made to converge to the exact solution, when all approximations are sufficiently small in magnitude and when the finite set of basis functions tends toward the limit of a complete set. However, due to the extremely expensive computational cost, studies of complex chemical systems are often limited. One of the most focused areas in computational chemistry nowadays is designing some approximations that are much faster than the *ab initio* methods while maintaining required accuracy. For example, semi–empircal methods[26, 27, 28, 29] make many approximations and obtain some parameters from empirical data, which results in great speed up with substantially degraded accuracy compared to *ab initio* methods.

Machine Learning (ML) methods have been used in various applications in computational chemstriy, such as computer–aided drug design[30], computational structural biology[31], quantum chemistry[32], and computational material design[33]. It has been used to infer properties, such as energies, of new molecules through interpolation in chemical compound space[34, 32, 35, 36]. In this thesis, we aim to apply ANNs in molecular geometry optimization, which is an area in computational chemistry lacking investigations to the author's knowledge.

1.4 Motivation and Organization

Molecular geometry optimization refers to the process seeking to find the geometry of a particular arrangement of the atoms that represents a local or global energy minimum. Essentially, it is non-linear optimization problem. It is of importance since the geometry optimized structures usually have physical significance, which may correspond to a substance as it is found in nature and the geometry of such a structure can be used in a variety of experimental and theoretical investigations in the field of chemical structure, thermodynamics, chemical kinetics, spectroscopy and others.

The traditional and accurate way, i.e. *ab initio* electronic structure methods, has

extreme computational cost, and hence chemical studies are often limited to the use of approximate methods which leads to a trade-off between accuracy and speed. This thesis devotes to build a artificial neural network (ANN) model to predict molecular energy accurately and further more to perform geometry optimization by computing the analytic gradients of the ANN.

Chapter 2 describes theoretical background and related work of applying neural network on computational chemistry problems.

Chapter 3 presents the mathematics of back propagation in neural network by either using function values or gradients. It contains the implementation details of artificial neural network.

Chapter 4 shows the results of the ANN applied on known numerical functions. One and two dimensional functions are considered. The effect of relative importance of function values and gradients during training is extensively studied.

Chapter 5 describes the data set generation for molecules, additional ANN training details, and the final performance of ANN on energy and gradient predictions. Molecular geometry optimization is then conducted for molecules that are beyond the training data set.

Chapter 6 concludes the thesis and suggest further directions.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Molecular Geometry Optimization

Within the Born–Oppenheimer approximation, which assumes the motion of atomic nuclei and electrons in a molecule can be separated, the geometry of a molecule at zero absolute temperature corresponds to the minimum of the total energy:

$$\{x_i, y_i, z_i\}_{T=0} = argmin_{\{x_i,y_i,z_i\}}(\langle\Psi|\hat{H}|\Psi\rangle) \tag{2.1}$$

where Ψ is the wave function describing the quantum state of the molecule under consideration, $\hat{H}$ is the Hamiltonian operator which characterizes the total energy, $T = 0$ is the zero absolute temperature and $\{x_i, y_i, z_i\}$ stands for the Cartesian coordinates of the molecule.

The energy of a molecule, which can be seen as a function of the Cartesian coordinates, labeled as $\mathbf{r}$, forms a potential energy surface (PES). Molecular geometry optimization is the process finding an arrangement in space of a collection of atoms where the net inter-atomic force on each atom is acceptably close to zero and the position on the PES is a stationary point ($\frac{\partial E}{\partial \mathbf{r}} = 0$) from an arbitrary starting state. An example is shown in Figure 2.1, where the points with red cross correspond to optimized structures.

Essentially, molecular geometry optimization is a non-linear optimization problem and variations of Newton–Raphson method, which approximates the objective function by a quadratic surface at each step, are currently the primary geometry optimization algorithms in quantum chemical (QC) software packages. The forces, i.e. gradients,

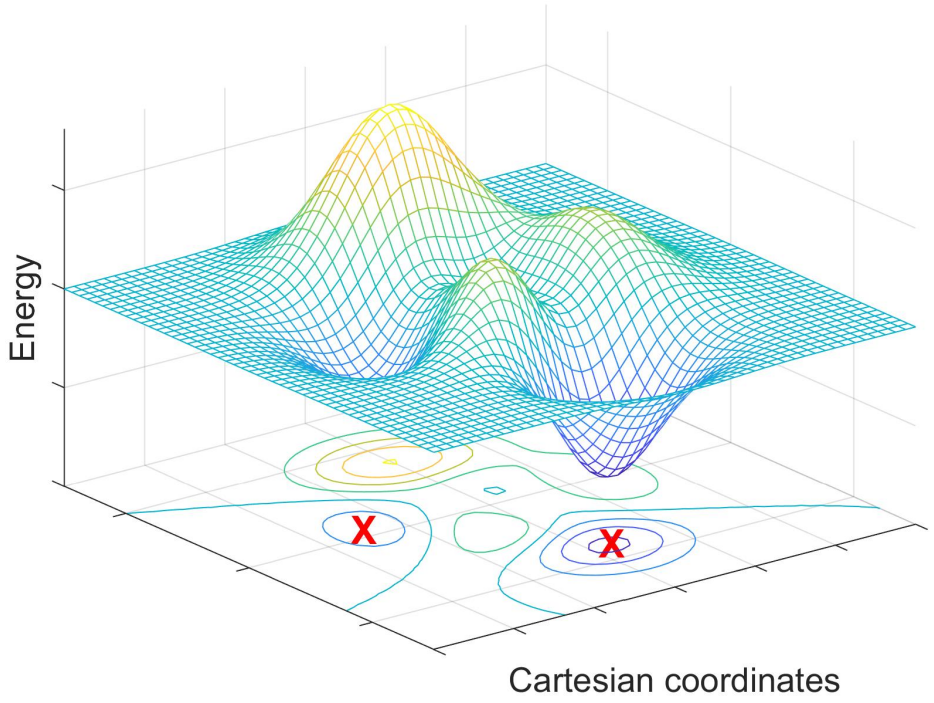

Figure 2.1: An example of a non-linear surface of energy. The points with red cross are possible geometry optimization structures.

on the nuclei can be calculated from the wave function using the Hellmann-Feynman theorem, which basically states

$$\frac{dE_\lambda}{d\lambda} = \langle\psi_\lambda|\frac{d\hat{H}_\lambda}{d\lambda}|\psi_\lambda\rangle \tag{2.2}$$

where $\hat{H}_\lambda$ is a Hamiltonian operator depending upon a continuous parameter λ, $|\psi_\lambda\rangle$ is an eigen-state of the Hamiltonian, depending implicitly upon λ, and E_λ is the energy of the state $|\psi_\lambda\rangle$. As the forces point toward a (at least local) minimum in the energy, the equations of motion for the nuclei will move the nuclei toward an equilibrium structure.

In principle, one can perform molecular geometry optimization if energy and gradients can be computed given the Cartesian coordinates of a molecular.

2.1.2 Molecular Dynamics

In addition to molecular geometry optimization, energy and gradients calculations have also been used extensively in another important area in computational chemistry, i.e. molecular dynamics (MD), which is a computer simulation method for studying the physical movements of atoms and molecules. The atoms and molecules are allowed to interact for a fixed period of time, giving a view of the dynamic evolution of the system. In each time step, the energy and forces are calculated on–the–fly. *Ab initio* molecular dynamics (MD) simulations have become a standard approach to study a variety of chemical process. However, again, due to the demanding computation resources, the *ab initio* MD is not an option for complex molecules.

2.1.3 Molecular Representation

In order to apply machine learning, the information encoded in the molecular three-dimensional structure needs to be converted into an appropriate vector of numbers. This vectorial representation of a molecule, also called "descriptors", is a data representation that reflects prior knowledge of the underlying physics and it is crucial for the success of the learning approach. Since the ANNs here are used to predict the energy of a molecule, only if all information relevant for the energy is appropriately encoded in

the vector will the machine learning algorithm be able to infer the relation between molecules and energies correctly.

2.2 Related Work

Molecular Representation. Two-dimensional molecular "descriptors" have been proposed for a support vector machine (SVM) algorithm for similiarity searching[37]. However, two-dimentional molecular descriptors could fail to predict properties that depend on the three-dimentional geometry. To address this, a variety of descriptors to describe the 3D structure of a molecule have been applied in statistical models for chemical and biology applications[38, 39]. Different system representations that include internal coordinates, system-specific variables, and complex projections of local atom densities have also been proposed in the context of potential-energy prediction[40, 34]. A simple Coulomb matrix representation was introduced by Rupp et. al.[32], where the off-diagonal elements correspond to the Coulomb repulsion between two atoms, while diagonal elements encode a polynomial fit of atomic energies to nuclear charge. Some graphical representations have been developed[41]. Using those represeations, relatively high accuracy has been achieved for inorganic or hybrid inorganic-organic crystals. An alternative means of representation of crystal structures has also be proposed based on Moron space-filling curves[42].

In this work, we adopted the data representation method introduced by Behler and Parrinello[1] and extended by Smith et al.[2]. This approximate molecular representation, called symmetry functions, not only takes advantage of chemical locality but also has some important properties that ensure energy conservation and be useful for machine learning models. The details of symmetry functions are described in Section 5.2.2.

Potential Energy Surface. There has been increasing interests in neural network potentails, i.e. using neural netwoks to represent a high-dimentional ab initio potential–energy surfaces (PES), over the last decades[43, 32, 1, 34, 2]. Those studies differ in the neural network models and molecular representations. Manzhos et al.[43] developed

a nested neural network technique in which one approximate neural network potential is fit first and then another neural network to fit the difference of the true potential and the approximate potential. Behler et al.[1] used a neural network topology by representing the total energy E of the system as a sum of atomic contributions E_i. As a result, there is one different neural network for each different atomic type[1]. Behler's approach has been proved to be successful in many cases[34, 44].

The analytic gradient of the energy hypersurface can be computed simply by differentiating the neural network, assuming the activation functions are defferentiable. This approach is superior to having the gradients appear in the output layer of the NN because it greatly simplifies the required architecture of the network. Both the energies and forces/gradients from electronic structure methods have been used in the training of the ANNs[45, 46, 47]. One practical framework for fitting a function and its derivatives is developed by Pukrittayakammee et al.[48] and it will be summarized in the Section 3.5.

Those ANN potentials have been applied in molecular dynamics[49]. However, the corresponding studies on molecular geometry optimization are very scarce.

Chapter 3

Methodology

3.1 Overview

In this chapter, we will first describe the notations of our NNs, the standard back propagation, analytic gradients of an ANN, fitting a function's first derivatives, and finally some implementation details.

3.2 Notations

We adopt the notations of neural network in [50] and let n_l denote the number of layers in our network and layer l is labeled as L_l. The parameters of our neural network are represented as (w_{ij}^l, b_i^l), where w_{ij}^l denotes the weight associated with the connected between unit j in layer l and unit i in layer $l+1$, and b_i^l is the bias associated with unit i in layer $l+1$. We use s_l to denote the number of nodes in layer l, without counting the bias unit. Also we let z_i^l denote the total weighted sum of inputs to unit i in layer l and a_i^l denotes the activation (i.e. output value) of unit i in layer l. Finally, we define f^l as the activation function in layer l.

It's straightforward to extend the above notation to vector/matrix form. $\boldsymbol{W^l}$ and $\boldsymbol{b^l}$ will represent the weights matrix and bias vector for layer l. They should have dimensions $[s_{l+1} \times s_l]$ and $[s_{l+1} \times 1]$, respectively. Suppose we have a fixed training set $\{(\boldsymbol{x^{(1)}}, \boldsymbol{t^{(1)}}), \cdots, (\boldsymbol{x^{(M)}}, \boldsymbol{t^{(M)}})\}$ (size is M), where $\boldsymbol{x^{(m)}} = (x_1^{(m)}, \cdots, x_{s_1}^{(m)})$ and $\boldsymbol{t^{(m)}} = (t_1^{(m)}, \cdots, t_{s_{n_l}}^{(m)})$ are the vectors of input features and the target output for the m-th training data point. We write $\boldsymbol{z^{l,(m)}}$ and $\boldsymbol{a^{l,(m)}}$ to denote the corresponding inputs and outputs in layer l for the m-th training data. And for the input layer, we have $\boldsymbol{a^{1,(m)}} = \boldsymbol{x^{(m)}}$. Furthermore, we let $\boldsymbol{X}$ denote the input features of all training

data and $\boldsymbol{Z^l}$ and $\boldsymbol{A^l}$ denote the corresponding inputs and outputs in layer l. $\boldsymbol{X}$, $\boldsymbol{Z^l}$, and $\boldsymbol{A^l}$ should have dimensions $[s_1 \times M]$, $[s_l \times M]$, and $[s_l \times M]$, respectively.

We write the derivative of the underlying function, that maps $\boldsymbol{x}$ to $\boldsymbol{t}$, with respect to $\boldsymbol{x^{(m)}}$ as $\frac{\partial \boldsymbol{t^{(m)}}}{\partial \boldsymbol{x^{(m)}}}$. And the derivative of the network activation $\boldsymbol{a_i^{l,(m)}}$ with respect to the input features of the m-th training data is written as $\frac{\partial \boldsymbol{a^{l,(m)}}}{\partial \boldsymbol{x^{(m)}}}$.

3.3 Forward and Back Propagation

Using the notations above, the forward propagation can be written as

$$\begin{aligned} \boldsymbol{A^1} &= \boldsymbol{X} \\ \boldsymbol{A^{l+1}} &= \boldsymbol{f^{l+1}}(\boldsymbol{W^l A^l} + \boldsymbol{b^l}) \end{aligned} \tag{3.1}$$

Assuming the cost function is represented as $\boldsymbol{J(W, b)}$, where $\boldsymbol{W}$ and $\boldsymbol{b}$ denote all the weights and bias of the network, then the back propagation[6, 7] of the error term can be written as

$$\begin{aligned} \boldsymbol{\delta^{n_l}} &= \frac{\partial \boldsymbol{J(W, b)}}{\partial \boldsymbol{A^{n_l}}} \bullet \dot{\boldsymbol{f}}^{\boldsymbol{n_l}}(\boldsymbol{Z^{n_l}}) \\ \boldsymbol{\delta^l} &= ((\boldsymbol{W^l})^{\boldsymbol{T}} \boldsymbol{\delta^{l+1}}) \bullet \dot{\boldsymbol{f}}^{\boldsymbol{l}}(\boldsymbol{Z^l}) \end{aligned} \tag{3.2}$$

where "$\bullet$" represents element-wise product operator and $\boldsymbol{\delta^{l+1}}$ is the error term matrix of all units in layer $l+1$ for all training data. $\boldsymbol{\delta^{l+1}}$ should have dimension $[s_{l+1} \times M]$. Simply speaking, the standard back propagation algorithm[6, 7] propagates the error term and meanwhile computes the gradients of the cost function with respect to the parameters on the fly. These gradients can be computed by

$$\begin{aligned} \frac{\partial J(\boldsymbol{W, b})}{\partial \boldsymbol{W^l}} &= \boldsymbol{\delta^{l+1}}(\boldsymbol{A^l})^{\boldsymbol{T}} \\ \frac{\partial J(\boldsymbol{W, b})}{\partial \boldsymbol{b^l}} &= \boldsymbol{\delta^{l+1}} \vec{\mathbf{1}} \end{aligned} \tag{3.3}$$

where $\vec{\mathbf{1}}$ is a column vector of 1's with length M. These computed gradients can then be used in any gradient-based optimization algorithm to minimize the cost function.

3.4 Analytic Derivatives of Neural Networks

The analytic derivatives of a neural network with respect to arbitrary variables $\boldsymbol{q}$ can be easily deducted through the chain rule. The goal is to obtain $\frac{\partial \boldsymbol{a^{n_l,(m)}}}{\partial \boldsymbol{q}}$, which can be

computed through

$$\begin{aligned}
\frac{\partial \boldsymbol{a}^{\boldsymbol{n_l},(\boldsymbol{m})}}{\partial \boldsymbol{q}} &= \dot{\boldsymbol{f}}^{\boldsymbol{n_l}}(\boldsymbol{z}^{\boldsymbol{n_l},(\boldsymbol{m})})\frac{\partial \boldsymbol{z}^{\boldsymbol{n_l},(\boldsymbol{m})}}{\partial \boldsymbol{q}} \\
\frac{\partial \boldsymbol{z}^{\boldsymbol{n_l},(\boldsymbol{m})}}{\partial \boldsymbol{q}} &= \boldsymbol{W}^{\boldsymbol{n_l-1}}\frac{\partial \boldsymbol{a}^{\boldsymbol{n_l-1},(\boldsymbol{m})}}{\partial \boldsymbol{q}} \\
&\cdots \\
\frac{\partial \boldsymbol{z}^{\boldsymbol{2},(\boldsymbol{m})}}{\partial \boldsymbol{q}} &= \boldsymbol{W}^{\boldsymbol{1}}\frac{\partial \boldsymbol{a}^{\boldsymbol{1},(\boldsymbol{m})}}{\partial \boldsymbol{q}}
\end{aligned} \tag{3.4}$$

If we look at equation 3.4 from bottom to top, it becomes clear that it's a forward propagation with input as $\frac{\partial \boldsymbol{a}^{\boldsymbol{1},(\boldsymbol{m})}}{\partial \boldsymbol{q}}$ and linear activation functions that are determined by a forward propagation with $\boldsymbol{x}^{(\boldsymbol{m})}$ as input. Therefore, to obtain the analytic derivatives of a network with respect to arbitrary variables $\boldsymbol{q}$, one can first propagate the input features $\boldsymbol{x}$ and store $\dot{\boldsymbol{f}}^{\boldsymbol{l}}$ for each layer, then compute the derivatives of all input units with respect to $\boldsymbol{q}$ and propagate these derivatives. Note that, since we treat the output value of the bias units as constant (i.e. 1), the derivatives of the bias terms are 0, hence they do not participate in this gradient propagation process. A simpler matrix form of this process can be expressed as

$$\frac{\partial \boldsymbol{A}^{\boldsymbol{l}}}{\partial \boldsymbol{q}} = (\boldsymbol{W}^{\boldsymbol{l-1}}\frac{\boldsymbol{\partial A}^{\boldsymbol{l-1}}}{\boldsymbol{\partial q}}) \bullet \dot{\boldsymbol{f}}^{\boldsymbol{l}}(\boldsymbol{Z}^{\boldsymbol{l}}) \tag{3.5}$$

If we want to compute the derivatives of the network with respect to the input features, i.e. $\boldsymbol{q} = \boldsymbol{x}^{(\boldsymbol{m})}$, we will have $\frac{\partial \boldsymbol{a}^{\boldsymbol{1},(\boldsymbol{m})}}{\partial \boldsymbol{x}^{(\boldsymbol{m})}} = \boldsymbol{I}$ (identity matrix), since $\boldsymbol{a}^{\boldsymbol{1},(\boldsymbol{m})} = \boldsymbol{x}^{(\boldsymbol{m})}$.

3.5 Fit the Function Derivative

In order to fit the derivatives of a function, we adopted the framework proposed by Pukrittayakamee[48]. A summary of the calculations (slightly modified version) will be provided as follows.

We use $J_d(\boldsymbol{W}, \boldsymbol{b})$ to denote the overall cost function with respect to the function derivatives. The goal is to calculate $\frac{\partial \boldsymbol{J_d(W,b)}}{\partial w_{ij}^l}$ and $\frac{\partial \boldsymbol{J_d(W,b)}}{\partial b_i^l}$, which can then used in gradient-based algorithm. We let $J_d(\boldsymbol{W}, \boldsymbol{b}; \boldsymbol{x}^{(m)}, \boldsymbol{t}^{(m)}, q_r^{(m)})$ denote the cost arose from the r-th derivative of the m-th data entry. Without confusion, this term is abbreviated as $J_{dr}^{(m)}$. Note that different data entry may have different number of derivatives, which

is the case for study in Chapter 5. We have

$$J_d(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{2N} \sum_{m=1}^{M} \sum_{r=1}^{R^{(m)}} J_{dr}^{(m)} \tag{3.6}$$

where M is the total number of training data, $R^{(m)}$ is the number of derivatives of the m-th training data, and N is a normalization factor that equals $\sum_{m=1}^{M} \sum_{r=1}^{R^{(m)}} 1$.

To make the following deduction simple, we first focus on a single training data and one derivative variable, followed by a generalization to matrix form. We have

$$\begin{aligned} \frac{\partial J_{dr}^{(m)}}{\partial w_{ij}^{l}} &= \sum_{s=1}^{s_{n_l}} \left(\frac{\partial J_{dr}^{(m)}}{\partial \frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}}}\right) \times \frac{\partial}{\partial w_{ij}^{l}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}}\right) \\ &= \sum_{s=1}^{s_{n_l}} D_s(J_{dr}^{(m)}) \times \frac{\partial}{\partial w_{ij}^{l}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}}\right) \end{aligned} \tag{3.7}$$

where we have used $D_s(J_{dr}^{(m)})$ to denote $\left(\frac{\partial J_{dr}^{(m)}}{\partial \frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}}}\right)$. Note that, the second term in equation 3.7 is a mixed second-order partial derivative of $a_s^{n_l,(m)}$. Assume these second-order partial derivatives exist and are continuous, then we have

$$\begin{aligned} \frac{\partial}{\partial w_{ij}^{l}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}}\right) &= \frac{\partial}{\partial q_r^{(m)}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial w_{ij}^{l}}\right) \\ &= \frac{\partial}{\partial q_r^{(m)}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l+1,(m)}} \times a_j^{l,(m)}\right) \\ &= a_j^{l,(m)} \times \frac{\partial}{\partial q_r^{(m)}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l+1,(m)}}\right) + \frac{\partial a_j^{l,(m)}}{\partial q_r^{(m)}} \frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l+1,(m)}} \end{aligned} \tag{3.8}$$

By further define the following two terms

$$\begin{aligned} v_{i,r}^{l,(m)} &\equiv \sum_{s=1}^{s_{n_l}} D_s(J_{dr}^{(m)}) \times \frac{\partial}{\partial q_r^{(m)}} \left(\frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l,(m)}}\right) \\ u_{i,r}^{l,(m)} &\equiv \sum_{s=1}^{s_{n_l}} D_s(J_{dr}^{(m)}) \times \frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l,(m)}} \end{aligned} \tag{3.9}$$

Equation 3.7 can be rewritten as

$$\frac{\partial J_{dr}^{(m)}}{\partial w_{ij}^{l}} = a_j^{l,(m)} v_{i,r}^{l+1,(m)} + \frac{\partial a_j^{l,(m)}}{\partial q_r^{(m)}} u_{i,r}^{l+1,(m)} \tag{3.10}$$

The term $\frac{\partial a_j^{l,(m)}}{\partial q_r^{(m)}}$ is updated through forward propagation described in Section 3.4. What's left is to show $u_{i,r}^{l,(m)}$ and $v_{i,r}^{l,(m)}$ can be computed through back propagation.

Note that,

$$\begin{aligned}\frac{\partial a_s^{n_l,(m)}}{\partial z_i^{l,(m)}} &= \sum_{j=1}^{s_{l+1}} \frac{\partial a_s^{n_l,(m)}}{\partial z_j^{l+1,(m)}} \frac{\partial z_j^{l+1,(m)}}{\partial z_i^{l,(m)}} \\ &= \sum_{j=1}^{s_{l+1}} \frac{\partial a_s^{n_l,(m)}}{\partial z_j^{l+1,(m)}} \times w_{ji}^l \times \dot{f}^l(z_i^{l,(m)})\end{aligned} \tag{3.11}$$

Thus, for $u_{i,r}^{l,(m)}$, we have

$$\begin{aligned}u_{i,r}^{l,(m)} &= \dot{f}^l(z_i^{l,(m)}) \sum_{j=1}^{s_{l+1}} w_{ji}^l \sum_{s=1}^{s_{n_l}} D_s(J_{dr}^{(m)}) \times \frac{\partial a_s^{n_l,(m)}}{\partial z_j^{l+1,(m)}} \\ &= \dot{f}^l(z_i^{l,(m)}) \sum_{j=1}^{s_{l+1}} w_{ji}^l u_{j,r}^{l+1,(m)}\end{aligned} \tag{3.12}$$

Equation 3.12 means that $u_{i,r}^{l,(m)}$ can be computed from layer to layer through back propagation with values initialized at the output layer as

$$u_{i,r}^{n_l,(m)} = D_i(J_{dr}^{(m)}) \dot{f}^{n_l}(z_i^{n_l,(m)}), i \in [1, s_{n_l}] \tag{3.13}$$

By applying equation 3.11 into $v_{i,r}^{(m)}$ and some rearrangements, we get

$$v_{i,r}^{l,(m)} = \frac{\partial \dot{f}^l(z_i^{l,(m)})}{\partial q_r^{(m)}} \sum_{j=1}^{s_{l+1}} (w_{ji}^l u_{j,r}^{l+1,(m)}) + \dot{f}^l(z_i^{l,(m)}) \sum_{j=1}^{s_{l+1}} (w_{ji}^l v_{j,r}^{l+1,(m)}) \tag{3.14}$$

where

$$\frac{\partial \dot{f}^l(z_i^{l,(m)})}{\partial q_r^{(m)}} = \frac{\partial \dot{f}^l(z_i^{l,(m)})}{\partial a_i^{l,(m)}} \frac{\partial a_i^{l,(m)}}{\partial q_r^{(m)}} \tag{3.15}$$

Again, we have known how to compute the second term in Section 3.4. Assume we can compute $\frac{\partial \dot{f}^l(z_i^{l,(m)})}{\partial a_i^{l,(m)}}$, e.g. $\frac{\partial \dot{f}^l(z_i^{l,(m)})}{\partial a_i^{l,(m)}} = 1 - 2a_i^{l,(m)}$ for sigmoid function, together with equation 3.14, it is clear that $v_{i,r}^{l,(m)}$ can be computed from layer to layer through back propagation with values initialized at the output layer as

$$v_{i,r}^{n_l,(m)} = D_i(J_{dr}^{(m)}) \times \frac{\partial \dot{f}^{n_l}(z_i^{n_l,(m)})}{\partial q_r^{(m)}} \tag{3.16}$$

Similar to the gradients with respect to $w_i j^l$, we can obtain

$$\frac{\partial J_{dr}^{(m)}}{\partial b_i^l} = v_{i,r}^{l+1,(m)} \tag{3.17}$$

It is straightforward to extend equations 3.10 and 3.17 into matrix forms as shown below:

$$\begin{aligned}\frac{\partial J_d(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W^l}} &= \boldsymbol{V^{l+1}}(\boldsymbol{A^l})^{\boldsymbol{T}} + \boldsymbol{U^{l+1}}(\frac{\partial \boldsymbol{A^l}}{\partial \boldsymbol{Q}})^{\boldsymbol{T}} \\ \frac{\partial J_d(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b^l}} &= \boldsymbol{V^{l+1}}\vec{\mathbf{1}}\end{aligned} \tag{3.18}$$

where $\boldsymbol{Q}$, whose dimension is $[1 \times N]$, is vector that contains all interested variables. The dimensions of $\boldsymbol{V^{l+1}}$, $\boldsymbol{U^{l+1}}$, and $\frac{\partial \boldsymbol{A^l}}{\partial \boldsymbol{Q}}$ are $[s_{l+1} \times N]$, $[s_{l+1} \times N]$, and $[s_l \times N]$.

3.6 Fit the Function Value and the Function Derivative

In addition to fit the function derivative, it is possible to fit the function value and function derivative simultaneously[48]. The cost function can be written as

$$J(\boldsymbol{W},\boldsymbol{b}) = \rho_1 J_f(\boldsymbol{W},\boldsymbol{b}) + \rho_2 J_d(\boldsymbol{W},\boldsymbol{b}) \tag{3.19}$$

where $J_f(\boldsymbol{W},\boldsymbol{b})$ is the cost arose from function value and ρ_1 and ρ_2 are the scale factors that determine the relative importance of $J_f(\boldsymbol{W},\boldsymbol{b})$ and $J_d(\boldsymbol{W},\boldsymbol{b})$. One may imagine to modify ρ_2 so as to tune the behavior of the cost function, while setting $\rho_1 = 1$. Moreover, one can set $\rho_1 = 0$, hence the NN will be only trained using the function derivatives.

As a concrete example, the root mean squared error cost function of both function values and its derivatives can be written as

$$J(\boldsymbol{W},\boldsymbol{b}) = \frac{\rho_1}{2M \times s_{n_l}} \sum_{m=1}^{M} \sum_{s=1}^{s_{n_l}} (a_s^{n_l,(m)} - t_s^{(m)})^2 + \frac{\rho_2}{2N \times s_{n_l}} \sum_{m=1}^{M} \sum_{r=1}^{R^m} \sum_{s=1}^{s_{n_l}} (\frac{\partial a_s^{n_l,(m)}}{\partial q_r^{(m)}} - \frac{\partial t_s^{(m)}}{\partial q_r^{(m)}})^2 \tag{3.20}$$

The overall back propagation algorithm is summarized in Figure 3.1.

3.7 Implementation Details

We implemented fully connected ANNs in CUDA (Computed Unified Device Architecture) C++. Since the computation contains extensive matrix and vector operations, we utilized the CUBLAS library, which is an implementation of BLAS (Basic Linear

1: **Input:** M supervised input features $(\boldsymbol{x}^{(m)})_{1\leq m\leq M} \rightarrow \boldsymbol{X}$ and the corresponding derivatives $(\frac{\partial \boldsymbol{x}^{(m)}}{\partial \boldsymbol{q}^{(m)}})_{1\leq m\leq M} \rightarrow \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{Q}}$.
2: **Forward propagation:**
3: Initialization: $\boldsymbol{A^1} = \boldsymbol{X}$ and $\frac{\partial \boldsymbol{A^1}}{\partial \boldsymbol{Q}} = \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{Q}}$.
4: **for** $l = 1$ **to** $n_l - 1$ **do**
5: **begin**
6: $\boldsymbol{Z^{l+1}} = \boldsymbol{W^l A^l} + \boldsymbol{b^l}$
7: $\boldsymbol{A^{l+1}} = \boldsymbol{f^{l+1}}(\boldsymbol{Z^{l+1}})$
8: Compute $\boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})$ and $\frac{\partial \boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})}{\partial \boldsymbol{A^{l+1}}}$
9: $\frac{\partial \boldsymbol{A^{l+1}}}{\partial \boldsymbol{Q}} = (\boldsymbol{W^l} \frac{\partial \boldsymbol{A^l}}{\partial \boldsymbol{Q}}) \bullet \boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})$
10: $\frac{\partial \boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})}{\partial \boldsymbol{Q}} = \frac{\partial \boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})}{\partial \boldsymbol{A^{l+1}}} \bullet \frac{\partial \boldsymbol{A^{l+1}}}{\partial \boldsymbol{Q}}$
11: **end**
12: **Compute error terms:**
13: Compute $\boldsymbol{\delta}^{n_l}$, $\boldsymbol{U^{n_l}}$, $\boldsymbol{V^{n_l}}$ based on equations 3.2, 3.13, and 3.16.
14: **Back propagation:**
15: **for** $l = n_l - 1$ **to** 1 **do**
16: **begin**
17: $\boldsymbol{\delta^l} = ((\boldsymbol{W^l})^{\boldsymbol{T}} \boldsymbol{\delta^{l+1}}) \bullet \boldsymbol{\dot{f}^l}(\boldsymbol{Z^l})$
18: $\boldsymbol{U^l} = ((\boldsymbol{W^l})^{\boldsymbol{T}} \boldsymbol{U^{l+1}}) \bullet \boldsymbol{\dot{f}^l}(\boldsymbol{Z^l})$
19: $\boldsymbol{V^l} = ((\boldsymbol{W^l})^{\boldsymbol{T}} \boldsymbol{U^{l+1}}) \bullet \frac{\partial \boldsymbol{\dot{f}^{l+1}}(\boldsymbol{Z^{l+1}})}{\partial \boldsymbol{Q}} + ((\boldsymbol{W^l})^{\boldsymbol{T}} \boldsymbol{V^{l+1}}) \bullet \boldsymbol{\dot{f}^l}(\boldsymbol{Z^l})$
20: **end**
21: **Compute gradients w.r.t. weights and bias:**
22: **for** $l = n_l - 1$ **to** 1 **do**
23: **begin**
24: $\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W^l}} = \rho_1 \boldsymbol{\delta^{l+1}} (\boldsymbol{A^l})^{\boldsymbol{T}} + \rho_2 (\boldsymbol{U^{l+1}} (\frac{\partial \boldsymbol{A^l}}{\partial \boldsymbol{Q}})^T + \boldsymbol{V^{l+1}} (\boldsymbol{A^l})^{\boldsymbol{T}})$
25: $\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b^l}} = \rho_1 \boldsymbol{\delta^{l+1}} \vec{\boldsymbol{1}} + \rho_2 \boldsymbol{V^{l+1}} \vec{\boldsymbol{1}}$
26: **end**

Figure 3.1: Back propagation algorithm for fitting both function values and function derivatives.

Algebra Subprograms) on the top of CUDA. In addition to the matrix-vector operations, there is another type of operation: element-wise arithmetic operations (e.g. apply activation function on all the units), which can be parallelized by launching "kernel" functions.

We applied stochastic gradient descent and Adam algorithm to update the weights and bias of the ANNs. The initial parameters of Adam method is chosen as recommended by the original authors[51], i.e. $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The max-norm regularization[52] was used with a maximum length of 3.0. We adopted the strategy by Smith et al.[2], i.e. the training epochs are iterated until the validation set stops improving in accuracy for 100 epochs and the optimization process is carried out 6 times using an order of magnitude smaller α (learning rate) each time. More technical details are covered in individual chapters.

We used dynamic feature in CUDA to generate kernel functions within kernel functions. This feature is not supported by low level architectures, e.g. sm_20.

Chapter 4

Numerical Functions Approximation Using Artificial Neural Networks

4.1 Overview

In this chapter, we focus on numerical functions to study the relative importance of function values and gradients as the training data. Two kinds of numerical functions are considered: one-dimensional and two-dimensional functions. The considered one-dimensional functions are:

$$y = e^x \tag{4.1}$$

$$y = \sin x + \cos x \tag{4.2}$$

$$y = 2x^3 + 5x^2 - 4x - 3 \tag{4.3}$$

$$y = \cos(4x) + x^2/4 + 0.5x - 0.5 \tag{4.4}$$

Eq. 4.4 has multiple minimas (Figure 4.1) and is used to investigate the implications of inclusion of function derivatives in the training process. The two-dimensional function considered is a sinc function defined as:

$$\begin{aligned} R &= \sqrt{x^2 + y^2} + \epsilon, \epsilon = 10^{-10} \\ z &= \frac{\sin(R)}{R} \end{aligned} \tag{4.5}$$

One of the benefits to work on numerical functions is that we know the exact forms of the functions. Thus, we can test our ANNs with confidence. In addition, we wanted to first see how the algorithms work on simple cases before moving on to the much more complex problem, i.e. molecular geometry optimization.

4.2 Dataset

The training set was generated by randomly sampling within the given domain, followed by computing the corresponding function values. For example, we restrict the domain of Eq. 4.4 to be $x \in [-2, 2]$. We randomly sample x and compute the corresponding function values and derivatives. Different sizes of training set, $2^n, n \in [4, \cdots, 16]$, have been generated.

The validation and testing set are fixed for different training and testing processes, so that we could directly compare the performances. For example, the same 1000 validation data and 1000 testing data are used for all experiments of Eq. 4.4. They are also generated randomly within the given domain.

4.3 Method and Notations

The basic back propagation algorithm is shown in Figure 3.1. In this chapter, we have $\boldsymbol{Q} = \boldsymbol{X}$. Since training ANNs has some stochastic features because of the introduced randomness, e.g. randomly generated initial weights and data are randomly shuffle each epoch, we train five ANNs for the same data/settings and report the one with lowest testing error. The cost function is the root mean squared error (RMSE) of both function values and its derivatives (Eq. 3.20). The architecture of the ANNs is represented as numbers separated by hyphen, e.g. 1-8-2-1 stands for an architecture having one dimensional input layer, one dimensional output layer, and two hidden layers with dimensions 8 and 2.

4.3.1 $\mathbf{ANN_f}$, $\mathbf{ANN_d}$, and $\mathbf{ANN_c}$

During the investigation of effect of relative magnitude of ρ_1 and ρ_2, we measure the performance of the ANNs by tuning the values of ρ_1 and ρ_2. If $\rho_1 = 1, \rho_2 = 0$, the ANNs are trained with only function values and defined as ANN_f. With $\rho_1 = 0, \rho_2 = 1$, the ANNs are trained with only function derivatives and defined as ANN_d. With $\rho_1 \neq 0, \rho_2 \neq 0$, the ANNs trained by combining both function values and derivatives and defined as ANN_c.

4.3.2 Root Variance of Error

One may expect that if the ANNs are trained using only the function derivatives as reference data, the predicted function values would be way off the true values. This is true at first glance. However, we find that the predicted function values are close to the true values after subtracting a common offset. The is because that although the original function is not unique given the derivative function, they differ only by a constant. For that reason, we compute the square root mean of the variance of testing errors (RVE) to measure the error for function values. Another reason to do so is that our final goal is to perform optimization, hence a common offset being subtracted would not matter. To measure the error for function derivatives, we compute the root mean squared error (RMSE).

Adding a further point, notice that fitting the RMSE of function values is identical to fitting the RVE of function values if no derivative data are used, hence the use of RVE measurement is only crucial for ANN_d architectures.

4.4 Results and Discussion

4.4.1 One Dimensional Functions

We show our ANNs are able to predict the function values and derivatives sufficiently well by fitting Eq. 4.4 with 1,024 training data points using an ANN architecture as 1-32-8-1 (Figure 4.1). The ANN is trained with only the function values as the reference data (i.e. ANN_f). The RVE for function values and RMSE for function derivatives are 0.0016 and 0.042, respectively. The results for all 1D functions are summarized in Table 4.1.

Performance of ANN_f and ANN_d

Besides showing results of ANNs trained with only function values (ANN_f), Table 4.1 also presents the results of ANNs trained with only function derivatives (ANN_d). It is clear that for all 1D functions considered, ANN_d produces almost one order of magnitude improvement for both function values and derivatives. What's really interesting is

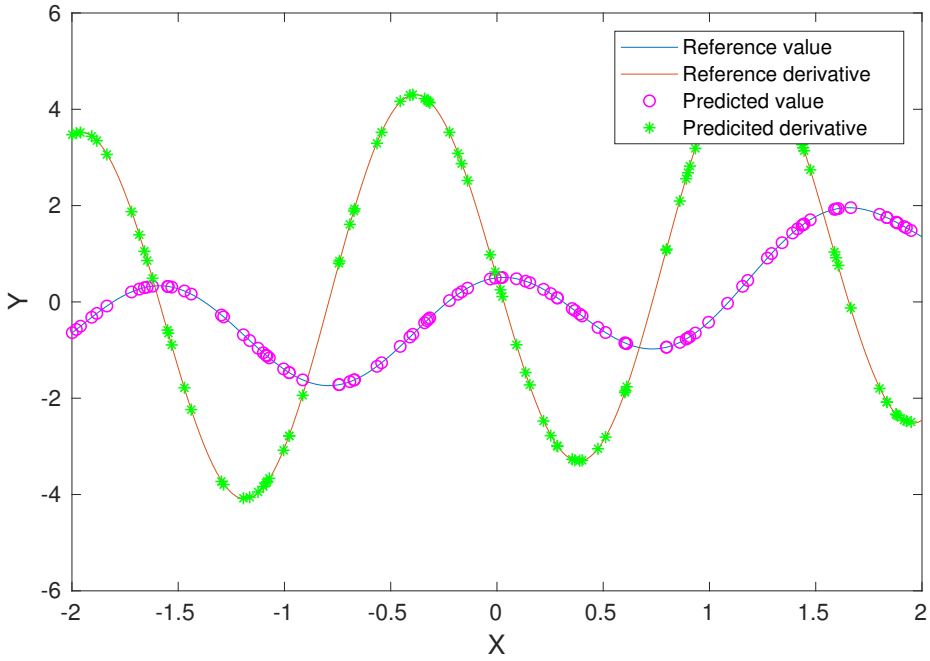

Figure 4.1: ANN_f (1-32-8-1) on 1D function $y = \cos(4x) + x^2/4 + 0.5x - 0.5$. Lines are reference data. Spheres and stars are the predicted function values and derivatives, respectively. Note that only 100 testing data points are selected to show.

Table 4.1: Performance of ANNs trained with 1D function values or derivatives for different functions. For each function, 1024 training data, 1000 validation data, and 1000 testing data are randomly chosen within their domain, respectively. The ANN architecture is 1-32-8-1. RVE is the root mean variance error of function values. RMSE is the root mean square error of function derivatives.

Functions	Domain	ANN_f		ANN_d	
		RVE	RMSE	RVE	RMSE
e^x	[-2,2]	0.0016	0.042	0.00015	0.0021
$\sin x + \cos x$	[-6,6]	0.0024	0.021	0.00063	0.0034
$2x^3 + 5x^2 - 4x - 3$	[-3,1]	0.0046	0.015	0.00042	0.0077
$\cos 4x + x^2/4 + 0.5x - 0.5$	[-2,2]	0.0011	0.020	0.00035	0.0064

that the statement holds true even for exponential function (Eq. 4.2), whose derivative is the same with itself. In another words, we use the identical training data to fit the function value or its derivative, the latter strategy has much better performance.

We can not provide a rigorous proof why fitting the function derivative is better in these cases, but some intuition can be gained from Taylor's Theorem, which states: if f is a function continuous and n times differentiable in an interval $[x, x+h]$, then there exits some point in this interval, denoted by $x + \lambda h$ for some $\lambda \in [0, 1]$, such that

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{h^{(n-1)}}{(n-1)!}f^{(n-1)}(x) + \frac{h^n}{n!}f^n(x+\lambda h) \quad (4.6)$$

Suppose $h \to 0$, then we may ignore the higher order terms and only consider the first two. Hence, we have

$$\begin{aligned} f(x+h) &\approx f(x) + hf'(x) \\ f(x+2h) &\approx f(x+h) + hf'(x+h) \approx f(x) + hf'(x) + hf'(x+h) \\ &\cdots \end{aligned} \quad (4.7)$$

Note that, the accuracy of predicted function derivatives is usually lower than the accuracy of predicted function values, which is verified by Table 4.1. It means that if $\mathrm{ANN_d}$ can fit the function derivatives to a similar accuracy level compared with $\mathrm{ANN_f}$ fitting the function values, $\mathrm{ANN_d}$ will have better accuracy in terms of function derivative prediction. Eq. 4.7 indicates that more accurate function derivatives, the more accurate function values will be predicted. Moving the argument along, if the derivatives predicted by $\mathrm{ANN_d}$ is more accurate than the derivatives predicted by $\mathrm{ANN_f}$, we will also have better predictions for function values by $\mathrm{ANN_d}$ if a common constant offset is allowed. Since the functions considered here have relative similar derivative forms with the function themselves, we may assume the ANN can fit them independently to a similar accuracy level. Hence, $\mathrm{ANN_d}$ is better than $\mathrm{ANN_f}$ in this context.

Table 4.2: Performance of ANNs of function $y = \cos 4x + x^2/4 + 0.5x - 0.5$ with different architectures. ANN_f is trained with function values only, and ANN_d is trained with function derivatives only. RVE is the root mean variance error of function values. RMSE is the root mean square error of function derivatives.

Architectures	ANN_f		ANN_d	
	RVE	RMSE	RVE	RMSE
1-4-2-1	0.0083	0.10	0.0029	0.037
1-8-2-1	0.0023	0.036	0.00059	0.011
1-16-4-1	0.0011	0.030	0.00049	0.0080
1-8-8-2-1	0.0048	0.095	0.00047	0.0079
1-32-8-1	0.0011	0.020	0.00035	0.0064
1-16-16-4-1	0.0013	0.028	0.00035	0.0058
1-64-8-1	0.0013	0.031	0.00027	0.0050

Different ANN Architectures

To investigate whether the argument above holds for different ANN architectures, we choose the relatively more complex Eq. 4.4 and perform ANN training and testing with different architectures (Table 4.2). Over all architectures considered, ANN_d consistently demonstrates much better performance for both function values and derivatives predictions.

Relative Importance of Function Values and Derivatives

The ANNs can also be trained by combining the function values and derivatives (ANN_c). Here we investigate the effect of ρ_1 and ρ_2 (recall the cost function $J = \rho_1 J_f + \rho_2 J_d$) for Eq. 4.4. Basically, we are tuning the relative importance of the function values and derivatives by setting $\rho_1 = 1$ and tuning ρ_2. Specifically, we want to answer two questions:

1. Is there an optimal ρ_2 for a specific problem?

2. If the answer to the first question is yes, does the size of the data set change the optimal ρ_2?

The second question can be useful, since we may find the optimal ρ_2 using much smaller data set and apply it for the training process with much larger data set. Pukrittayakamee et al. proposed that ρ_2 should depend on the relative magnitude of the

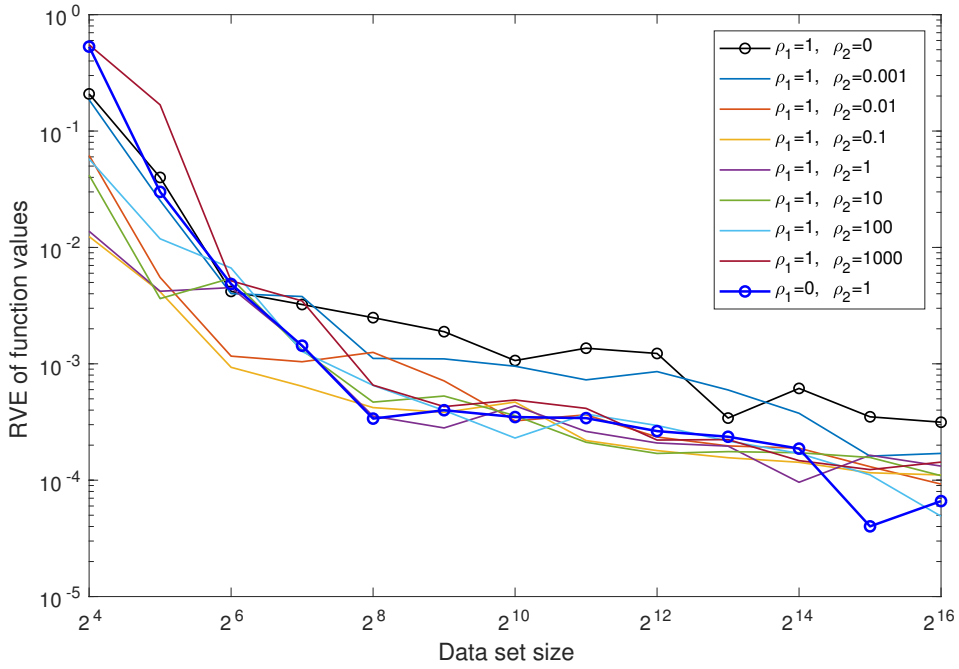

Figure 4.2: Root mean of variance of testing errors of function values with different coefficient ρ. The black line/circle is the result from ANN_f; the blue line/circle is obtained from ANN_d. The ANN has architecture as 1-32-8-1.

absolute values of function values and derivatives[48]. In the present work, we show that given enough data, the inclusion of function derivatives during the ANN training can significantly improve the performance with relatively large ρ_2. Alternatively, using only the function derivatives to train ANN can free us from searching ρ_1 and ρ_2 while providing the results with accuracy to the same level.

To proceed, 13 different training sets have been generated with sizes as $2^n, n \in [4, \cdots, 16]$. The same validation and testing data sets, with 1,000 data points each, are used for all different cases. The ANN architecture is fixed as 1-32-8-1. The RVE of function values and RMSE of function derivatives are shown in Figure 4.2 and 4.3, respectively.

We first noticed that the shapes of the RVE and RMSE errors are very similar to each other, which confirms the earlier argument that the more accurate function derivatives the ANNs predict, the more accurate function values. In general, the inclusion of derivative data in the training process generally significantly improve the performance

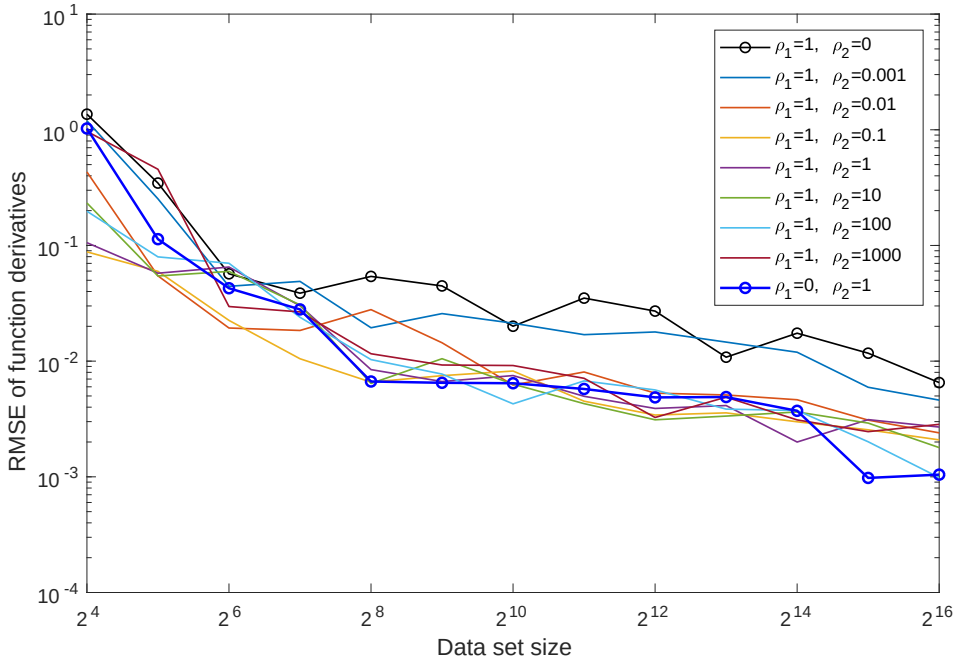

Figure 4.3: Root mean square errors of function derivatives with different ρ. The black line/circle is the result from a ANN_f; the blue line/circle is obtained from ANN_d. The ANN has architecture as 1-32-8-1.

the ANNs. For example, with $\rho_1 = 1$ and $\rho_2 = 1$, the ANN_c with 2^8 training data have similar performance compared with ANN_f with 2^{13}, i.e. similar performance is obtained with five orders of magnitude less training data. The better performance by inclusion of derivative data means better generalization which indicates the derivative data may help overcome over-fitting. Too small value of ρ_2 may perform poorer than larger value, e.g. $\rho_2 = 0.001$ produces larger errors than $\rho_2 = 1$ over all the data set sizes.

The blue lines in Figure 4.2 and 4.3 are obtained from ANN_d, which shows consistently good performance for both function values and derivatives predictions. One can think of $\rho_1 = 0, \rho_2 = 1$ to be identical to $\rho_1 = 1, \rho_2 = \infty$. Hence, larger ρ_2 should produce similar errors to ANN_d, which is exactly what we observe here.

In summary, for one-dimensional functions, we have showed that ANN_d performs much better than ANN_f with the same or much less amount of data. And, training with derivative data may help to prevent over-fitting. Moreover, if we allow the predicted

function values to have a common offset to the true function values, there is no need to search a ρ_2, since ANN_d would not lose any accuracy. One more thing need to be noticed is that the performance does not improve significantly even though the data size increases dramatically after data size larger than 2^8.

4.4.2 Two Dimensional Sinc Function

We now test the 2D sinc function (Eq. 4.5). The derivatives of Eq. 4.5 are

$$\begin{aligned} \frac{\partial z}{\partial x} &= (\frac{\cos R}{R} - \frac{\sin R}{R^2}) \times \frac{\partial R}{\partial x} \\ \frac{\partial z}{\partial y} &= (\frac{\cos R}{R} - \frac{\sin R}{R^2}) \times \frac{\partial R}{\partial y} \end{aligned} \tag{4.8}$$

It seems clear that the derivatives are in a more complex form than the original function. We expect less improvement compared with the 1D functions. To see our neural net works correctly, we first fit a significantly large data set. In specific, the training set contains 2,000,000 data points and the validation set contains 200,000 data points. These data points are randomly sampled from the domain $x \in [-10, 10]$ and $y \in [-10, 10]$. The testing set (described in Section 4.2) has 10,000 data points and are generated by sampling evenly from both x and y axis. The ANN architecture is 2-32-8-1. The result is shown in Figure 4.4. It's clear that our neural net is able to fit this 2D sinc function accurately. Note that the derivatives are of the same magnitude with the function value.

Train with both Function Value and Derivative

Similar to the 1D function, we investigate the influence of coefficients ρ_1 and ρ_2 by training the neural net with both function value and derivatives. Ten different training sets have been generated with sizes as 2^x, where $x \in [8, \cdots, 17]$. The results for the predictions of function value and derivatives are shown in Figure 4.5 and 4.6, respectively.

With small data set size, e.g. 2^8, it turns out ANN_f produces better results in terms of function values but with worse function derivatives compared with ANN_d. We ascribe this observation to over-fitting caused by insufficient data points. With much

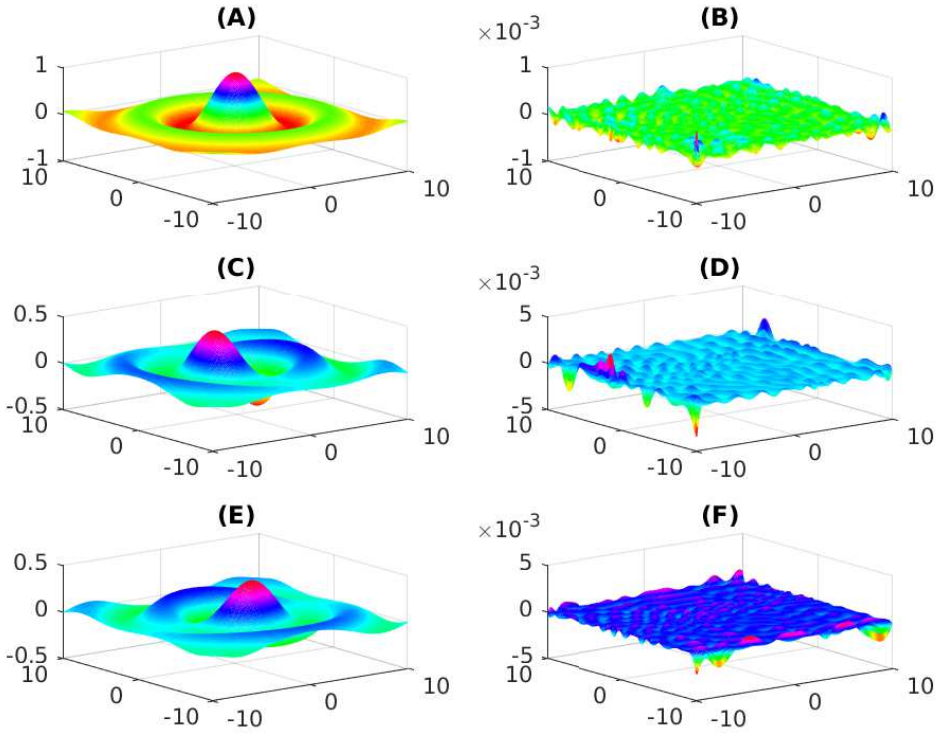

Figure 4.4: Predicted function values and derivatives and the corresponding errors of ANNs on two-dimensional sinc function. (A) The predicted function values; (B) Testing errors of function values; (C) The predicted derivatives with respect to x; (D) The testing errors of derivatives with respect to x; (E) The predicted derivatives with respect to y; (F) The testing errors of derivatives with respect to y.

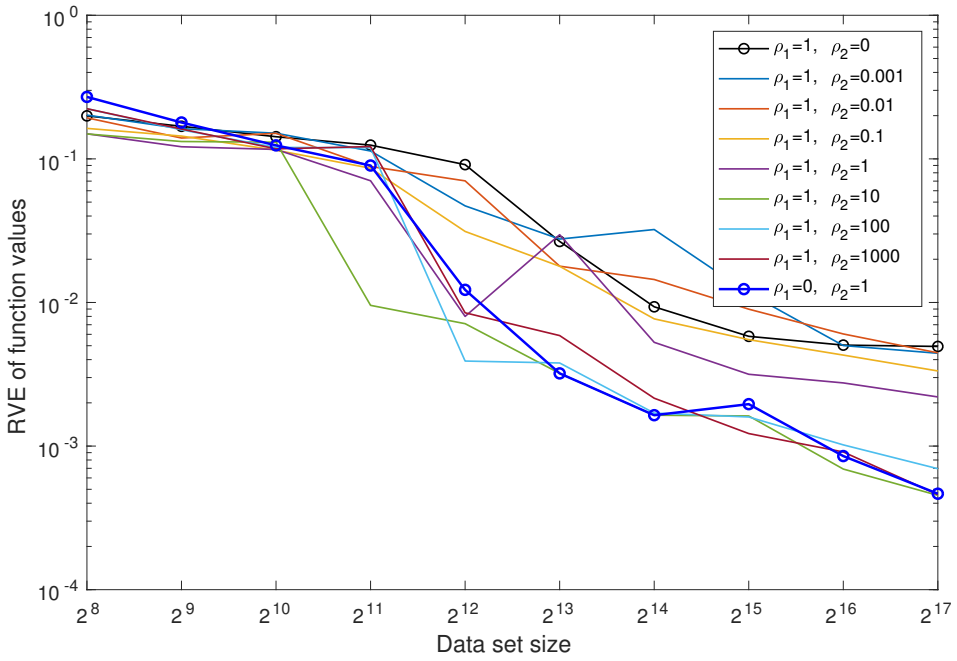

Figure 4.5: Square root of variance of testing errors of function values with different coefficients ρ_1 and ρ_2. The back line/cirle is the result from ANN_f; the blue line/circle is obtained from ANN_d. The ANN has architecture 1-32-8-1.

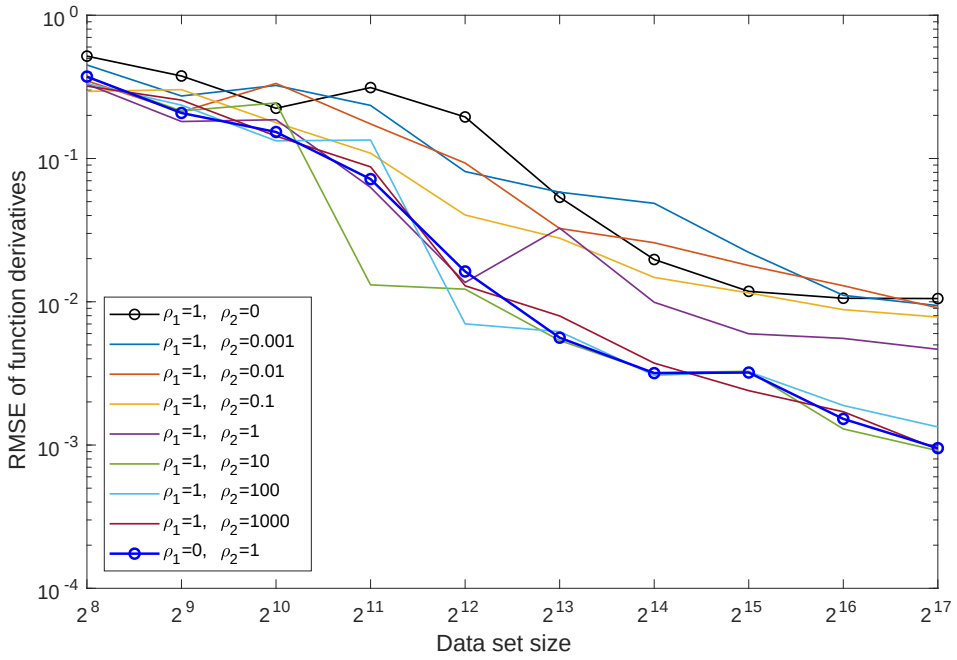

Figure 4.6: Root mean square errors of function derivatives with different coefficients ρ_1 and ρ_2. The back line/cirle is the result from ANN_f; the blue line/circle is obtained from ANN_d. The ANN has architecture 1-32-8-1.

larger data set, the ANN_c and ANN_d outperform ANN_f in most cases. For example, with $\rho_1 = 1, \rho_2 = 100$, both RVE of function values and RMSE of function derivatives are lower than 10^{-2}, whereas the corresponding results for ANN_f are more than 10^{-1}, which is an order of magnitude worse.

Besides, from Figure 4.5 and 4.6, the performance of ANN_f improves only marginally after the data set size larger than 2^{15}, whereas the ANN_d and some of ANN_c continually achieve better and better results. This indicates a fundamental advantage of training with function derivatives.

In terms of relative importance of ρ_1 and ρ_2, we observe that ANN_c with $\rho_1 = 1, \rho_2 = 100$ has very similar performance compared with ANN_d. This is also true for one-dimensional functions (Figure 4.2 and 4.3).

4.5 Summary

In this chapter, we applied our ANNs on 1D and 2D numerical functions. The activation function is chosen to be tanh, which is differentiable so that the analytic gradients of the ANN can be readily computed according to Section 3.4. Different ANN architectures are explored for the 1D functions. In addition, we also investigated the relative importance of function values and derivatives during training process.

The results show that our ANNs can fit the considered functions to sufficiently high accuracy. The usage of derivatives can significantly improve the performance of the ANN in terms of predictions of both function values and derivatives. This conclusion holds true for different ANN architectures.

In terms of the relative importance of function values and derivatives, i.e. magnitudes of ρ_1 and ρ_2, we found that ANN_c with $\rho_1 = 1, \rho_2 = 100$ generally produces good results. Moreover, if a common offset is allowed, artificial neural network with solely function derivatives as the reference data (ANN_d) predict accuracy function values and derivatives. As a result, the selection of ρ_1 and ρ_2 may not be necessary in some cases.

In conclusion, the inclusion of function derivatives helps the ANN to achieve better generalization. Experiments with inclusion of even higher order of derivatives may be

pursued to further smooth the ANN.

Chapter 5

Geometry Optimization of Hydrocarbon Molecules

5.1 Overview

To use machine learning techniques, e.g. artificial neural network, to predict molecular energies, the key is to find a good molecular representation. ANNs for predicting molecular energies are also referred to as neural network potentials (NNPs). As summarized by Behler[44], there are several properties that such presentation should maintain to make neural network potentials transferable. Those are:

1. Since the structure of a system and its energy remains unchanged upon these operations, the molecular representation must be invariant with respect to translations and rotations of the system.

2. Any exchange of atoms of the same element, which is not at all restricted to positions being equivalent by symmetry, must yield the same result.

3. It should provide a unique description of the atomic positions.

The molecular presentation approximated using symmetry functions, developed by Behler and Parrinello, takes advantage of chemical locality and satisfies the above requirements and has been applied to chemical studies[1]. A detailed description of a modified version of symmetry functions by Smith et al.[2] is described in Section 5.2.2. The computed molecular presentations are then fed into the neural network for training, validation and testing.

In this chapter, we first introduce the concepts of symmetry functions and the neural network architecture for potential energy. We then describes the generation and representation of our data set. With a much smaller data set, we determined the

optimal activation function. ANNs are then trained using energies or gradients. Finally, the ANNs are combined with a optimization algorithm, e.g. Fletcher-Reeves conjugate gradient, to perform molecular geometry optimization.

5.2 Theory

5.2.1 Neural Network Potential for Molecule

We applied Behler and Parrinello high-dimensional neural network potential model[1] to represent the total energy E_T of a molecule as a sum of atomic contributions E_i, an approach that is typically used in empirical potentials.

$$E_T = \sum_i E_i \tag{5.1}$$

Figure 5.1 shows the overall neural network potentials (NNP) for atomic energies and molecular total energy. In short, an atomic environment vector (AEV), $\vec{G}_i^X = \{G_1, G_2, \cdots, G_M\}$, where $\vec{G}_i^X$ is for the i-th atom of a molecule with atomic number X, is computed for each atom in a molecule based on symmetry functions (see below) to describe the atomic environment of the interested atom. Those AEVs are then fed into the corresponding networks according to their atomic number to predict the atomic energies, whose summation is the prediction of the total energy.

5.2.2 Symmetry Functions

The symmetry function representation of each atom reflects the local environment that determines its energy. It works as a molecule descriptor that describes the radial and angular chemical environment of each individual atom. The formulas for the derivatives of these symmetry functions are also dervied here.

To capture the energetically local atomic environment, a piece-wise cutoff function of the inter-atomic distance R_{ij} will be employed. Basically, cutoff function diminishes as R_{ij} increases and becomes 0 after a threshold value R_c. This implies atoms further

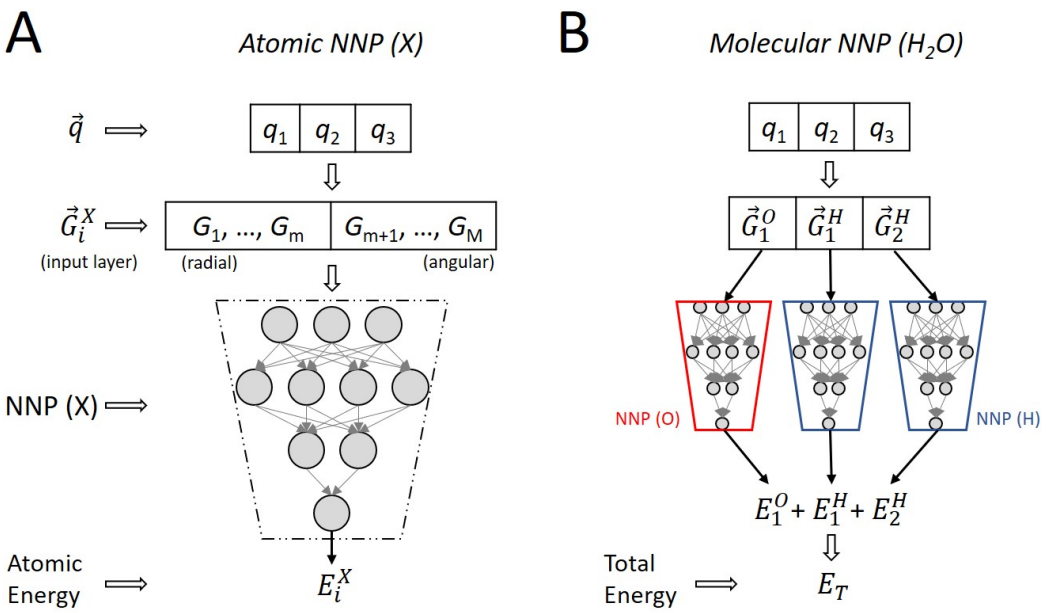

Figure 5.1: Neural network potentials for atomic energy and molecular total energy by Behler and Parrinello[1]. $\vec{q}$, the Cartesian coordinates of the molecule, is used to generate $\vec{G}_i^X$, i.e. the atomic environment vector for atom i with atomic number X. Each type of neural network popential (NNP) exists for each atomic number, e.g. NNP(O) for oxygen and NNP(H) for hydrogen. The atomic environment vectors are fed into the corresponding NNP to predict the atomic energies, which are summed to generate the predicted total energy. (Figure is reconstructed from ref.[2])

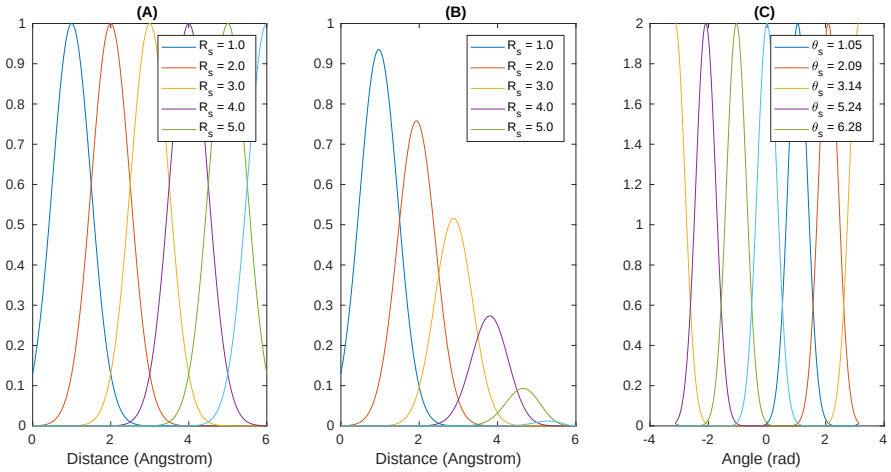

Figure 5.2: Different symmetry functions. (A) Radial symmetry functions; (B) Radial symmetry functions with cutoff $R_c = 6$ Å; (C) Angular symmetry functions.

away from the interested atom would have less impact.

$$f_c(R_{ij}) = \begin{cases} 0.5 \times [\cos(\frac{\pi R_{ij}}{R_c}) + 1], & \text{if } R_{ij} \leq R_c \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Radial Symmetry Functions. Radial symmetry functions, introduced by Behler and Parrinello, are constructed as a sum of Gaussians, multiplied by the cutoff function (Eq. 5.3). Sample plots are shown in Figure 5.2A and 5.2B for without and with cutoff function, respectively.

$$G_m^R = \sum_{j \neq i}^{\text{all}} e^{-\eta(R_{ij} - R_s)^2} f_c(R_{ij}) \tag{5.3}$$

The radial symmetry function is over a set of η and R_s hyper parameters, where the former changes the width of the Gaussian distribution and the later shifts the center of the peak. The summation over all neighbors j ensures the independence of the coordination number, i.e. the dimension of the AEV depends only on the set of η and R_s.

The partial derivative of G_i^R with respect to variable $\vec{q}$ is 0 if $R_{ij} > R_c$. With $R_{ij} \leq R_c$, and assuming q_{ix} is the x axis of the i^{th} atom, then $\frac{\partial R_{ij}}{\partial q_{ix}} = \frac{q_{ix} - q_{jx}}{R_{ij}}$ and

$$\frac{\partial G_i^R}{\partial q_{ix}} = \sum_{j \neq i}^{\text{all}} e^{-\eta(R_{ij} - R_s)^2} (f_c'(R_{ij}) - 2\eta(R_{ij} - R_s) f_c(R_{ij})) \times \frac{1}{R_{ij}} \times (q_{ix} - q_{jx}) \tag{5.4}$$

Details of the derivation are shown in Appendix A.1. Similar results can be obtained for q_{iy} and q_{iz}.

Angular Symmetry Functions. A modified version of angular symmetry functions, introduced by Smith et al.[2], are constructed for all triplets of atoms by summing the product of a radial and an angular factor (Eq. 5.5).

$$\begin{aligned} G_m^A =& 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} (1+\cos(\theta_{ijk}-\theta_s))^\zeta \\ & \times e^{-\eta(\frac{R_{ij}+R_{ik}}{2}-R_s)^2} f_c(R_{ij}) f_c(R_{ik}) \end{aligned} \tag{5.5}$$

The angular symmetry function is over a set of ζ, θ_s, η, and R_s, where η and R_s have similar effects as in Eq. 5.3. The choices of ζ alter the width of the peak values, while a set of θ_s let it probe specific region in terms of angular environment. Figure 5.2C shows plots of angular symmetry functions with different θ_s. Apparently, by increasing the number of θ_s, one can increase the resolution of the angular environment with the penalty of a larger vector size.

The partial derivatives of G_i^A with respect to variable $\vec{q}$ are in much more complex forms. Details are provided in Appendix A.2. Together with the derivatives of G_i^R w.r.t. $\vec{q}$, they are used for neural network training against gradients.

As pointed out by Behler and Parrinello[1], the choice of symmetry functions and their parameters is not unique, and many types of functions can be used, as long as the set of function values is suitable for describing the environment of an atom. We adopted this modified version by Smith et al.[2] simply because they have showed it worked well for chemical molecule energy prediction.

5.3 Dataset

5.3.1 Dataset Selection

In this thesis, we limit the scope of the chemical space to hydro-carbon molecules, which contains only two atom types: hydrogen (H) and carbon (C). Since our goal is to perform geometry optimization, it's crutial to have accurate energy and gradient predictions near the equilibrium conformations while lower accuracy is acceptable for conformations

away from the equilibriums. Hence, we designed our data generation method (see below) to bias structures close to the equilibrium conformations. To obtain the reference data, we performed QM calculations with ORCA[53]. Geometry optimizations were carried out with BP86 functional[54, 55], which is known to yield more realistic structural parameters than hybrid functionals[56]. The basis set is 6-31G(d). Tight convergence, increased integration grids (Grid4 in ORCA convention) and an unrestricted Kohn–Sham method were used. All calculations are performed with neural charge in the singlet spin state.

The original structures of our molecules come from the GDB–11 database[57, 58], which collects all molecules of up to 11 atoms of C, N, O, and F possible under consideration of simple valency, chemical stability, and synthetic feasibility rules. We use the RDKit software package to convert the molecules in GDB–11, which are stored in SMILES format, to 3D structures. In this work, we generate data from a subset of the GDB–11 database containing only H and C. In addition, considering the giant database, only molecules up to 9 atoms of C are included, which currently gives us 8,299 distinct molecules in total. For final testing purpose, 100 molecules with 11 C atoms are randomly chosen from the GDB-11 database.

5.3.2 Dataset Generation

From the 8,299 distinct molecules, we need to generate a data set in which the structures near the equilibrium structure are sufficiently sampled. Different sampling methods have been proposed, e.g. quantum mechanical molecular dynamics (QMMD) simulation[59] and Normal Mode Sampling[2]. In this work, we applied a much simpler stochastic approach to fulfill the needs.

Specifically, for each molecule obtained and converted from the GDB–11 database, the following steps are applied:

1. Optimize the molecule structure using quantum chemical (QM) calculations.

2. Starting from the optimized structure, generate K structures for the molecule. Each set of coordinates is generated by randomly sampling for each atom within

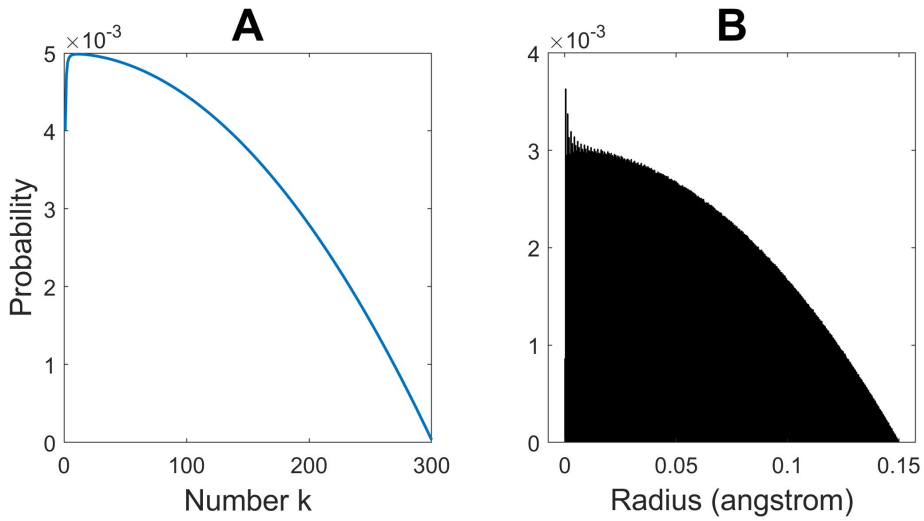

Figure 5.3: Distributions of randomly generated data points. (A) Theoretical probability density of randomly generated points fall between i^th and $(i-1)^th$ spheres. (B) Corresponding experimental probability (histogram) of the generated coordinates. The i^{th} sphere in **A** corresponds to a sphere with radius $0.0005 \times i$ Å in **B** with the atom's original position being the center.

a radius r. Currently, we set $R = 0.15$ Å and split it into $K = 300$ intervals evenly, i.e. $r \in [0.0005, 0.001, \cdots, 0.15]$ Å. A new set of coordinates is generated with each r. For example, with $r = 0.001$ Å, the new coordinate of O in H_2O would be a random point within 0.001 Å from the original position of O (the same goes for the two H's).

3. Calculate the energy and gradients for each of the generated structure.

The philosophy behind the described data set generation is that it allows more data points near the optimal structures. To see this, we can compute the corresponding theoretical probability density. Suppose the interval length is d, it's easy to show that

$$p(E_i) = [i^3 - (i-1)^3] \times (\frac{1}{i^3} + \frac{1}{(i+1)^3} + \cdots + \frac{1}{K^3}) \tag{5.6}$$

where i indicates the i^{th} interval, K is the total number of considered intervals ($K = 300$), and $p(E_i)$ is the probability of E_i, which indicates an event that a randomly generated point falls between two spheres with radius $i \times d$ and $(i-1) \times d$ (with the original position of the atom being the center). A plot of $p(E_i)$ is shown in Figure 5.3A.

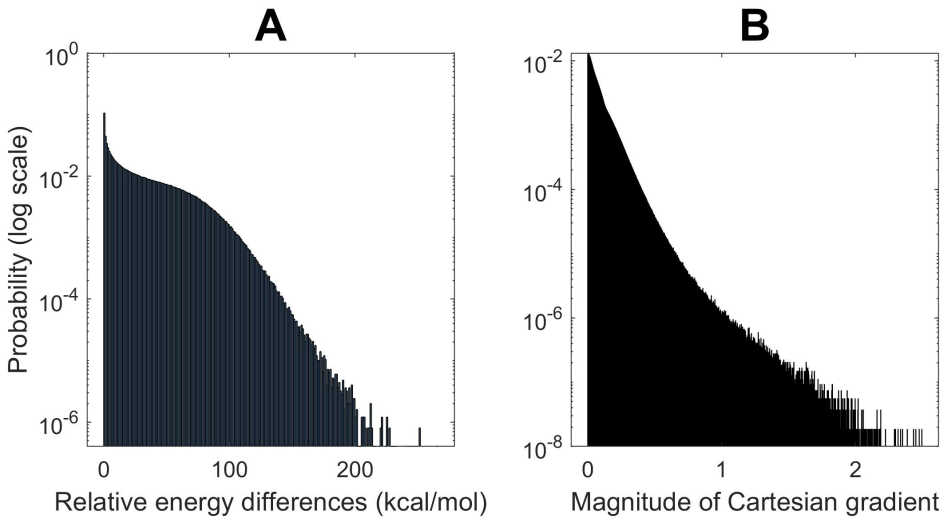

Figure 5.4: Experimental probability (log scale) of the energy gaps of the generated structures.

A corresponding histogram of our generated coordinates of molecules is shown in Figure 5.3B. Clearly, the experimental probability closely reproduces the theoretical prediction. By comparing the energies of the randomly generated structures with the corresponding optimal structure, we obtain a plot describing the probability with respect to energy differences (Figure 5.4A). In addition, the distribution of the Cartesian gradient is shown in Figure 5.4B. As a result, more data points are generated closer to the equilibrium. In this way, we expect the trained neural net to have better precision in terms of prediction when it's close to the optimal structures. When the structure is away from the optimal structure, we are not so concerned about the precision of the prediction as long as the predicted gradients point the structure to the right direction.

This approach generates 2,489,700 conformations in total. To train the ANNs using the energy or gradient data, 80% of individual set of random conformations for each molecule is used for training, while 10% for validation and the remaining 10% for testing.

5.4 Method

Detailed descriptions of the ANN implementation are shown in Chapter 3. Each model for each training set is trained 5 times, the best of which is chosen. Here we summarize the hyper parameters used in current work.

Table 5.1: Hyper parameters for the symmetry functions.

Parameters		Values	Units
Radial	R_c	4.600	Å
	η	98.88	–
	R_s	$[r, 2r, \cdots, N^R r]$, $r = 0.1438, N^R = 32$	Å
Angular	R_c	3.100	Å
	ζ	31.03	–
	θ_s	$[0, \theta, 2\theta, \cdots, (N^A - 1)\theta]$, $\theta = 0.7854, N^A = 8$	rad
	η	13.61	–
	R_s	$[r, 2r, \cdots, N^A r]$, $r = 0.3875, N^A = 8$	Å

Atomic environment vector. For symmetry functions (Table 5.1), the cutoff radii for radial and angular symmetry functions are chosen to be 4.6 and 3.1 Å based on the distribution of atomic distances[2]. Radial and angular R_s and θ_s are chosen such that they are equally spaced with interval $r = \frac{R_c}{N}$ ($N = 32$ for radial and 8 for angular), $\theta = \frac{\pi}{N}$ ($N = 8$). We use Eq. 5.7 to determine the values of η and ζ. In our implementation, we set $h = 0.6$, which indicates two adjacent functions intersect at height 0.6, hence they overlap slightly.

$$\begin{aligned} \eta &= -\frac{4\log(h)}{r^2} \\ \zeta &= \frac{\log(h/2)}{\log(\frac{1+\cos(\theta/2)}{2})} \end{aligned} \tag{5.7}$$

We also adopted the approach proposed by Smith et al.[2] to differentiate between atomic numbers in the AEV through supplying a radial part for each atomic number and an angular part for each atomic number pair in the local chemical environment. For example, for molecules with C and H, the AEV for C will be in the form of $\{G^{R,\mathrm{CC}}, G^{R,\mathrm{CH}}, G^{A,\mathrm{CCC}}, G^{A,\mathrm{CCH}}, G^{A,\mathrm{CHH}}\}$, where $G^{R,\mathrm{CC}}$ indicates the radial part concerning only carbon and $G^{A,\mathrm{CCH}}$ indicates the angular part concerning a carbon and hydrogen pair. This approach leads to AEV with dimension $O(N^2)$, where N is the number of atom types. The generation of AEVs is also powered by GPU.

ANN architecture. With $N^R = 32$ and $N^A = 8$, the input for the ANNs has dimension 256. For both C and H, the ANN architectures are chosen to be 256-128-128-64-1.

The cost function is the root mean squared error (RMSE) of both molecular energies and gradients ($J = \rho_1 J_f + \rho_2 J_d$). The relative importance of the energies and gradients

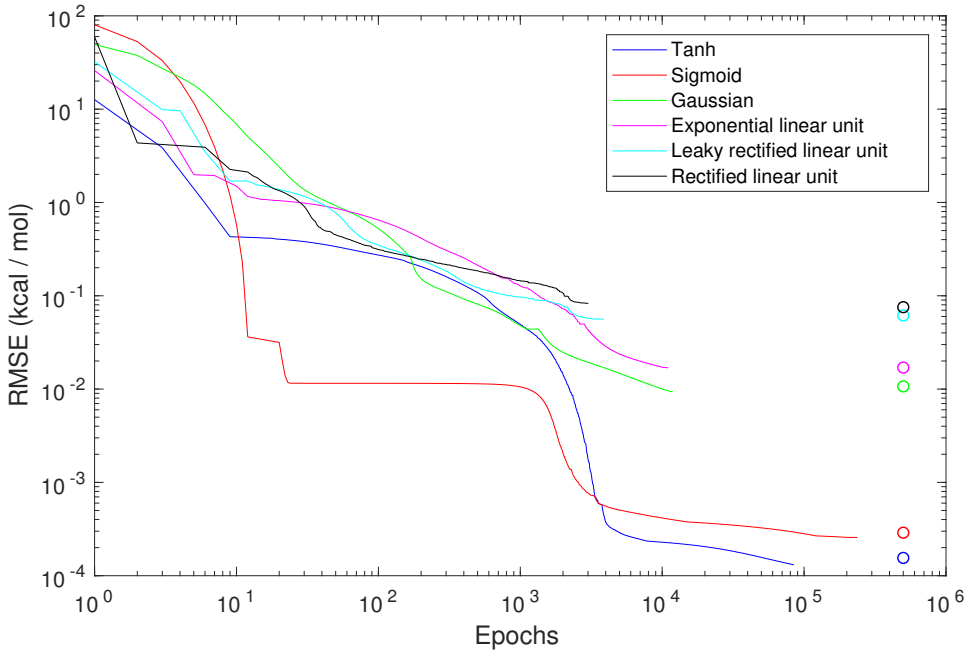

Figure 5.5: Performance of different activation functions on samples of butane. The lines are validation errors along the training epochs and the circles on the right side are the final testing errors.

are tuned by setting ρ_1 and ρ_2. There is one benefit of this cost function for gradients. To see this, suppose the predicted Cartesian gradients for an atom is $\hat{g}_p = (g_{xp}, g_{yp}, g_{zp})$ with the true values being $\hat{g}_t = (g_{xt}, g_{yt}, g_{zt})$, then the cost function for gradient part is $\sqrt{(g_{xp} - g_{xt})^2 + (g_{yp} - g_{yt})^2 + (g_{zp} - g_{zt})^2}$. Note that $\hat{g}_p - \hat{g}_t = (g_{xp} - g_{xt}, g_{yp} - g_{yt}, g_{zp} - g_{zt})$, so the effect of the cost function is equivalent to minimize the length of a vector $\hat{g}_p - \hat{g}_t$ instead of the 3 dimensional Cartesian gradients. This is considered to be computationally efficient.

5.5 Results and Discussion

5.5.1 Selection of Activation Function

There are quite a few activation functions at our disposal, e.g. tanh, sigmoid, gaussian, etc. We trained ANNs with various activation functions for 10,000 randomly generated butane conformations (C_4H_{10}). The cost function is based on molecular energies only. The result is shown in Figure 5.5. Sigmoid activation function tends to reduce the cost

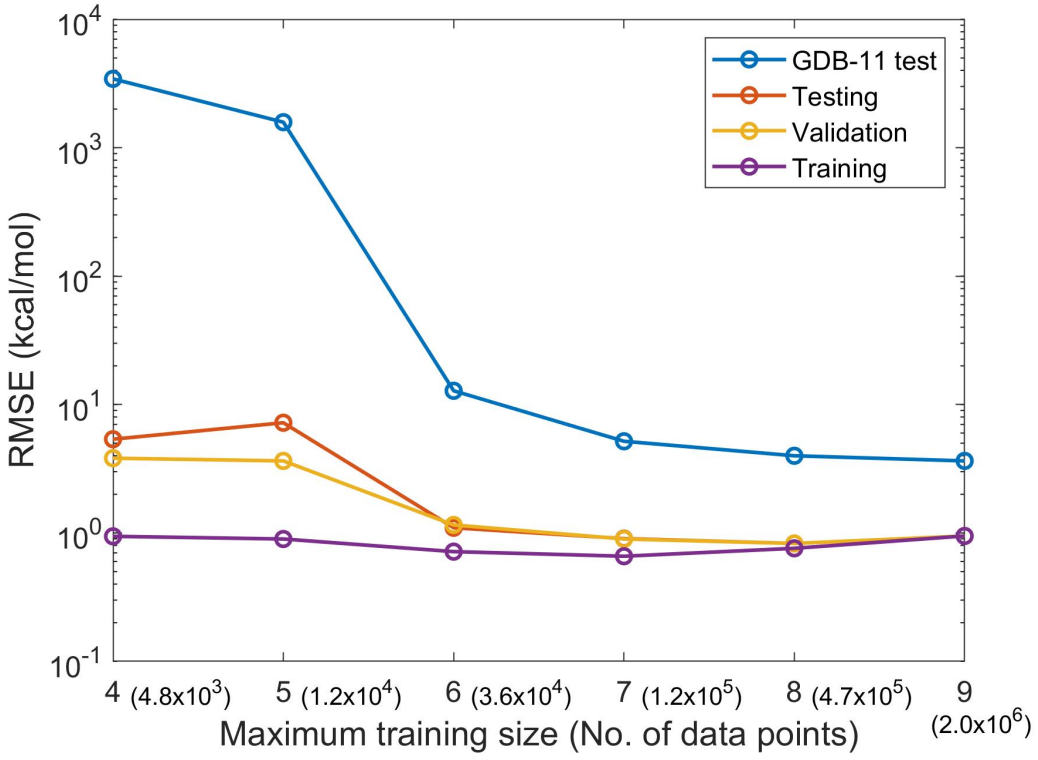

Figure 5.6: Plots of errors of the training, validation, testing, and 100 randomly GDB-11 (molecules with 11 heavy atoms) versus increasing data set size.

function very fast at the early stage but is beaten by tanh in the long run. Hence, tanh is chosen for later experiments. Some activation functions, e.g. rectified linear unit (RELU), are not differentiable, hence may introduce some complexity if used in training with gradients as the reference data.

5.5.2 Performance of ANN_f

We now train ANN using tanh as the activation function with only molecular energies as the reference data, i.e. ANN_f. Figure 5.6 shows how the RMSE of our ANN potential to reference DFT energies decreases as the number of distinct molecules grows in the training set. The performance improvement is significant when number of data points is small. With about 2 million data points (training set contains molecules up to 9 carbon atoms), ANN_f achieves RMSE 3.6 kcal/mol for GDB-11 test set.

In order to compare the predicted gradients (g^{ANN}) with the gradients from DFT

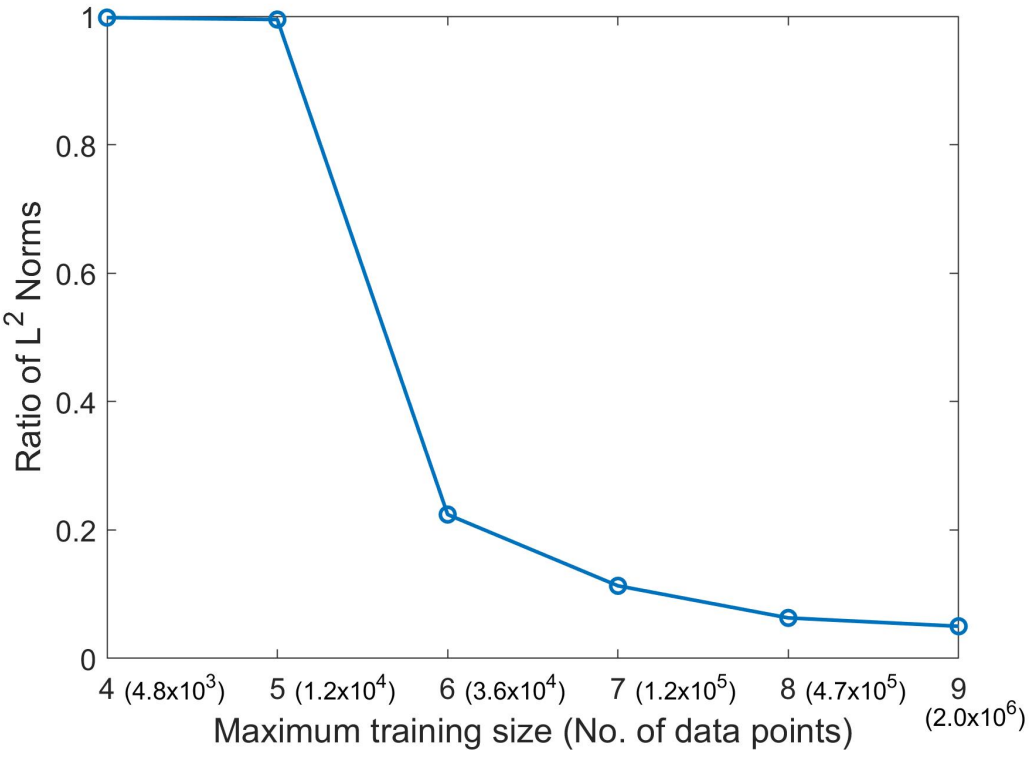

Figure 5.7: Performance of gradient prediction versus training size. The y axis stands for ratio of L^2 norm of the differences between predicted and true gradients to L^2 norm of the sum of both gradients.

calculations (g^{DFT}), we consider the relative error defined by the ratio of L^2 norm of the differences to that of the sum of both gradients:

$$r_{L^2} = \frac{\|g^{\mathrm{ANN}} - g^{\mathrm{DFT}}\|}{\|g^{\mathrm{ANN}} + g^{\mathrm{DFT}}\|} \tag{5.8}$$

Apparently, r_{L^2} is close to 0 when g^{ANN} is close to g^{DFT}, and r_{L^2} becomes larger when g^{ANN} is very different from g^{DFT}. Figure 5.7 shows the r_{L^2} versus training size. Again, the largest training data set size gives the best result with $r_{L^2} \approx 0.05$.

5.5.3 Selection of ρ_1 and ρ_2

The data set size increases rapidly as the number of structures increases and $\mathrm{ANN_c}/\mathrm{ANN_d}$ training is much slower, hence we first investigate the relative importance of energy and gradients, i.e. ρ_1 and ρ_2, with molecules up to 4 carbon atoms, which gives us about 4.8×10^3 energy data points and 1.4×10^5 gradient data points. The measure of RVE

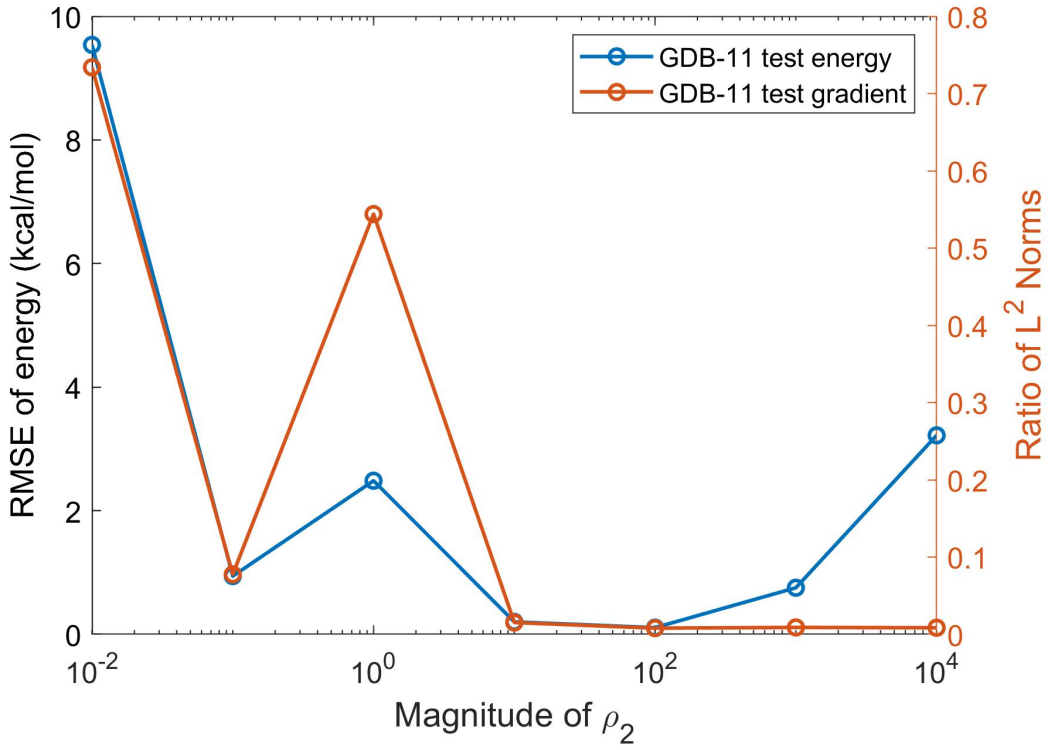

Figure 5.8: Performance in terms of energy and gradient predictions of ANN_c with different ρ_2. ρ_1 is set to be 1. The training data set contains 80% molecules up to 4 carbon atoms. The testing data set is another separated 10% data with up to 4 carbon atoms.

(Section 4.3.2) may not be available for molecular energy predictions because multiple NNPs are involved for different number of times. Hence, we simply use RMSE to describe the error in terms of energy predictions. For gradient, we use r_{L^2} (Eq. 5.8).

Figure 5.8 shows the results with $\rho_1 = 1$ and various ρ_2. With very small ρ_2, both energy and gradient predictions are not good. With very large ρ_2, which means the effect of gradients is significant, we obtain accurate gradient prediction while the energy prediction starts getting worse. We choose $\rho_1 = 1, \rho_2 = 100$ as the setting for further ANN training with larger data set, since it gives the lowest errors for both energy and gradient predictions, 0.1 kcal/mol and 7.9×10^{-3}, respectively. Note that, the testing is not against the GDB-11 testing data.

For ANN_d, i.e. $\rho_1 = 0, \rho_2 = 1$, the RMSE of energy is huge, whereas the $r_{L^2} \approx 8.7 \times 10^{-3}$, which is slightly higher than the corresponding ratio 7.9×10^{-3} with $\rho_1 =$

Table 5.2: Performance of ANN_c with $\rho_1 = 1, \rho_2 = 100$ on GDB-11 testing data. The first three training set contains up to 4, 5, and 6 carbon atoms. The last training set contains all energy data points of molecules up to 9 carbon atoms.

Training Set		GDB-11 Testing	
Energy data	Gradient data	RMSE of energy (kcal/mol)	r_{L^2}
4.8×10^3	1.4×10^5	> 600	0.983
1.2×10^4	4.2×10^5	49.3	0.494
3.6×10^4	1.5×10^6	16.8	0.0954
2.0×10^6	0	3.6	0.05

$1, \rho_2 = 100$. Hence, ANN_d models are no longer considered below.

Moving forward, train our ANN_c with larger data set applying $\rho_1 = 0, \rho_2 = 100$ and test again the GDB-11 data to verify the generalization property of our ANN. The results are shown in Table 5.2.

Clearly, as the size of the data set increase, better performance is achieved. What's surprising is that with data of molecules containing only up to 6 carbon atoms, the ANN_c reaches similar accuracy in terms of gradient prediction with ANN_f trained with all energy data of molecules containing up to 9 carbon atoms. This is remarkable, since molecules with up to 6 carbon atoms cover much less conformation space. It indicates that the inclusion of gradient as part of the reference data greatly improves the generalization of the ANN. Most likely, the reason that ANN_f performs better is it covers significantly more conformation space.

Figure 5.9 shows a more comprehensive comparison in terms of gradient prediction. One issue with current ANN models is that the gradient predictions have large variations when the true gradient is small, i.e. near the equilibrium state.

5.5.4 Molecular Geometry Optimization by ANN

In this section, we perform molecular geometry optimization on 100 randomly generated structures from GDB-11 data set. Those structures are generated according to the method described in Section 5.3.2 with $r = 0.15$ and $K = 1$. The ANN is chosen to be ANN_f with all energy data obtained, since it outperforms other models. Fletcher-Reeves conjugate gradient algorithm (implemented in GNU scientific library) is used for the optimization process. The geometry optimization is performed with max iteration

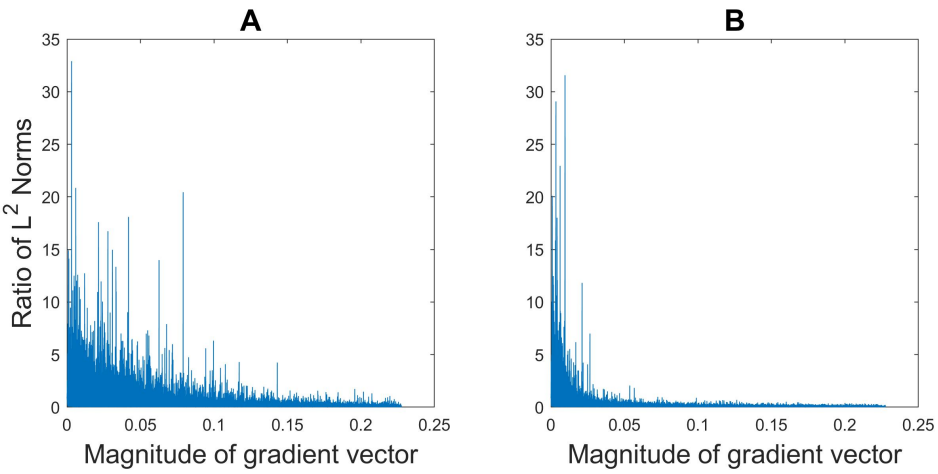

Figure 5.9: Performance in terms of gradient predictions of (A) ANN_c with $\rho_1 = 1, \rho_2 = 100$ using energy and gradients data points of molecules up to 6 carbon atoms. (B) ANN_f with energy data points of molecules up to 9 carbon atoms. The x axis is the magnitude of the true gradients. The y axis is r_{L^2}.

200. The measurement of the performance is the root mean square deviation (RMSD) between the optimized structure using DFT and the final structure using ANN. The RMSD is computed based on Kabsch algorithm.

5 of the 100 structures converge within 200 iterations. The average RMSD is 0.047 Å. This result is expected, since the gradient predictions do not work very well (Figure 5.9). We did another testing by perform molecular geometry optimization on randomly generated structures having 9 carbon atoms with $r = 0.15$ and $K = 1$. 20 of 100 structures converge within 200 iterations with mean RMSD around 0.038 Å. The performance is better because the conformation space is more extensively explored for molecules with 9 carbon atoms.

We closely studied the geometry optimization process of the GDB-11 testing data set by investigating the changes of RMSD during the optimization processes (Figure 5.10).

One structure (row 2, column 1) diverges from the optimal structure from the very beginning. This may be caused by insufficient sampling of the conformation space. The RMSD of most other structures actually first reduce to a fairly low value (< 0.05 Å). However, instead of converging to the equilibrium structures, their RMSDs keeps increasing after passing the lowest point. This is most likely caused by poor predictions

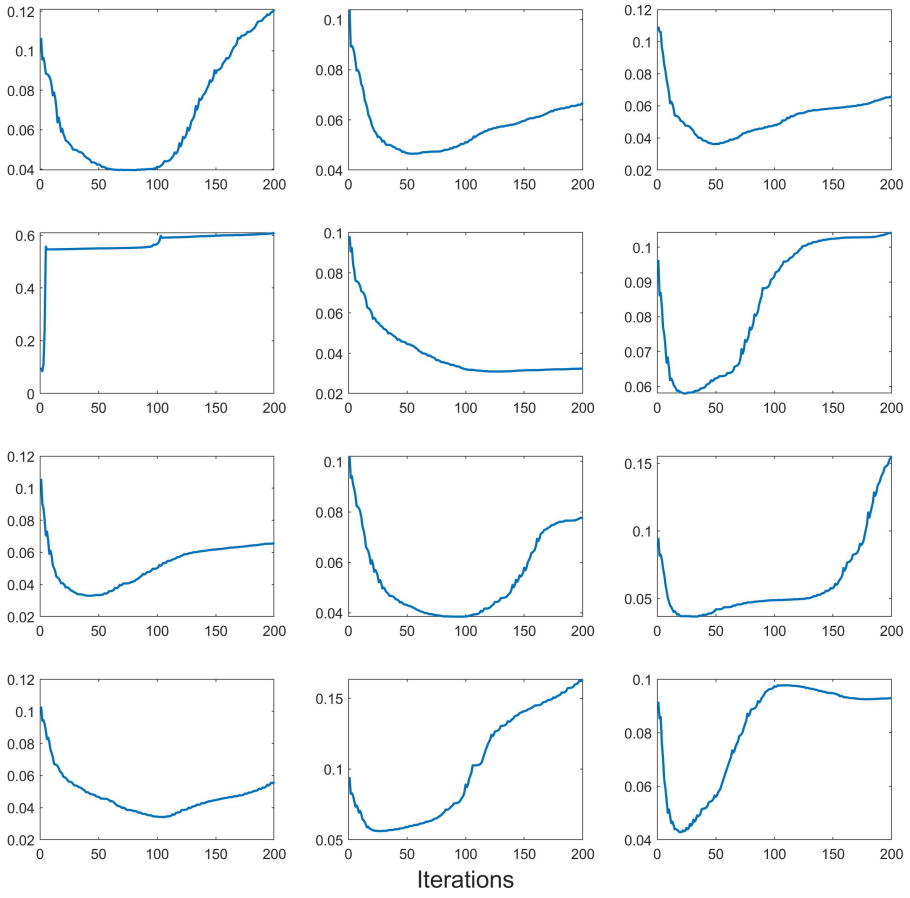

Figure 5.10: 12 randomly chosen geometry optimization processes from the GDB-11 testing data. The y axis is RMSD Å compared with the optimal structures.

near the equilibrium (Figure 5.9).

5.6 Summary

In this chapter, we described the theoretical background of NNP and symmetry functions, data set generations, and details of ANN for chemical molecules. We applied three kinds of ANNs, i.e. $\mathrm{ANN_f}$, $\mathrm{ANN_d}$, and $\mathrm{ANN_c}$, on molecules to predict energies and gradients.

The inclusion of gradients greatly improves the performance of the ANNs, similar to result in Chapter 4. With data points of molecules with up to 6 carbon atoms, we have trained an $\mathrm{ANN_c}$ model ($\rho_1 = 1, \rho_2 = 100$) that achieves closely accuracy compared with an $\mathrm{ANN_f}$ model trained with all energy data points of molecules with up to 9 carbon atoms. The inclusion of gradients as the reference data boosts the generalization of the ANN, which makes it more transferable.

We have performed molecular geometry optimization using the trained ANN. However, only 5% of the GDB-11 testing structures converge. This is mainly caused by the poor prediction of gradients around the equilibrium, since the square error cost function leads to a situation where data with larger magnitude produces larger cost. Further investigation may consider a different cost function in terms of gradients such that the cost increases as the gradients become smaller. Moreover, the accuracy of the ANN depends heavily on the data used during training. Thus, new molecules will improve the transfer-ability of the ANN.

Chapter 6

Concluding Remarks and Future Directions

In this thesis, molecular energy and gradient predictions are studied extensively. Molecular geometry optimization by artificial neural network is realized.

A detailed description of back propagation with either function values and derivatives is summarized in Chapter 3. ANN has been implemented in CUDA C++ from scratch, accelerated by CUBLAS library. Both function values and derivatives can be treated as reference data points. We tested our ANNs on known 1D and 2D numerical functions. The considered functions are predicted with sufficiently high accuracy. As for the relative importance of function values and derivatives, we found that function values are not required for those simple function forms if one is only interested in obtaining accurate accuracy. The predicted function values by ANN trained with only function derivatives differ from the true values by a constant offset.

To train an ANN for molecular energy prediction, we have adopted a modified version of symmetry functions proposed by Smith et al.[2] as the molecular descriptor. On one hand, the inclusion of molecular gradients greatly improve the performance of the ANN. On the other hand, ANN trained with significantly more energy data points outperforms other models and hence is used for molecular geometry optimization. Experiments with inclusion of even higher order of derivatives may be pursued to further smooth the ANN.

Our ANN is able to converge a small portion of the GDB-11 testing structures, which are beyond the training data set. However, the gradient prediction near the equilibrium is not sufficiently accurate. As a result, a large percentage of the GDB-11 testing structures passed the equilibrium but eventually diverge from the optimal points during geometry optimization.

Further experiments would focus on providing new molecules for the training process. Another focus may be put on the cost function in terms of gradients. A cost function whose values increase as the molecular gradients get smaller may help train ANN to achieve better accuracy near equilibrium and hence make it easier to converge. Current optimization algorithm (Fletcher-Reeves conjugate gradient) does not utilize the molecular energies to assist the optimization process. More sophisticated optimization should be tested and compared. Alternatively, one can combine the ANN with an existed QM package, such as ORCA. One could even train an ANN to directly mimic the geometry optimization performed by QM packages. There are many things to be done and can be done. The present work should be able to provide a good starting point.

Appendix A

Derivatives of Symmetry Functions

Here we derive the formulas for the derivatives of symmetry functions (equation 5.3 and 5.5) with respect to q_{ix}, which is the Cartesian coordinate of the i^{th} atom along the x axis. Similar results also apply to q_{iy} and q_{iz}, where i denotes the i^{th} atom of a molecule and x, y, z denote the Cartesian coordinates. For the readers reference, we restate the radial and angular symmetry functions here

$$\begin{aligned} G_m^R &= \sum_{j\neq i}^{\text{all}} e^{(-\eta(R_{ij}-R_s)^2} f_c(R_{ij}) \\ G_m^A &= 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} (1+\cos(\theta_{ijk}-\theta_s))^\zeta \\ &\quad \times e^{-\eta(\frac{R_{ij}+R_{ik}}{2}-R_s)^2} f_c(R_{ij}) f_c(R_{ik}) \end{aligned} \tag{A.1}$$

A.1 Derivatives of Radial Symmetry Function

We have $R_{ij} = \sqrt{(q_{ix}-q_{jx})^2+(q_{iy}-q_{jy})^2+(q_{iz}-q_{jz})^2}$, therefore $\frac{\partial R_{ij}}{\partial q_{ix}} = \frac{q_{ix}-q_{jx}}{R_{ij}}$. As for the derivative of G_i^R with respect to q_{ix}, we have

$$\begin{aligned} \frac{\partial G_i^R}{\partial q_{ix}} &= \sum_{j\neq i}^{\text{all}} e^{-\eta(R_{ij}-R_s)^2}(f_c'(R_{ij}) - 2\eta(R_{ij}-R_s)f_c(R_{ij})) \times \frac{\partial R_{ij}}{\partial q_{ix}} \\ &= \sum_{j\neq i}^{\text{all}} e^{-\eta(R_{ij}-R_s)^2}(f_c'(R_{ij}) - 2\eta(R_{ij}-R_s)f_c(R_{ij})) \times \frac{q_{ix}-q_{jx}}{R_{ij}} \\ &= \sum_{j\neq i}^{\text{all}} g(R_{ij}) \times (q_{ix}-q_{jx}) \end{aligned} \tag{A.2}$$

where we define

$$g(R_{ij}) = e^{-\eta(R_{ij}-R_s)^2}(f_c'(R_{ij}) - 2\eta(R_{ij}-R_s)f_c(R_{ij})) \times \frac{1}{R_{ij}} \tag{A.3}$$

The partial derivative of G_j^R, where $j \neq i$, with respect to q_{ix} is

$$\frac{\partial G_j^R}{\partial q_{ix}} = g(R_{ij}) \times (q_{ix} - q_{jx}) \tag{A.4}$$

We can readily see that

$$\frac{\partial G_i^R}{\partial q_{ix}} = \sum_{j \neq i}^{\text{all}} \frac{\partial G_j^R}{\partial q_{ix}} \tag{A.5}$$

A.2 Derivatives of Angular Symmetry Function

First, we have $\theta_{ijk} = \arccos(\frac{R_{ij}^2 + R_{ik}^2 - R_{jk}^2}{2R_{ij}R_{ik}})$, which gives us

$$\begin{aligned}
\frac{\theta_{ijk}}{\partial q_{ix}} &= \frac{-1}{\sqrt{1-(\cos\theta_{ijk})^2}}\left(\frac{R_{ij}\frac{\partial R_{ij}}{\partial q_{ix}} + R_{ik}\frac{\partial R_{ik}}{\partial q_{ix}}}{R_{ij}R_{ik}} - \frac{R_{ij}^2 + R_{ik}^2 - R_{jk}^2}{2(R_{ij}R_{ik})^2}\left(R_{ik}\frac{\partial R_{ij}}{\partial q_{ix}} + R_{ij}\frac{\partial R_{ik}}{\partial q_{ix}}\right)\right) \\
&= \frac{-1}{\sqrt{1-(\cos\theta_{ijk})^2}}\left(\left(\frac{1}{R_{ij}R_{ik}} - \frac{\cos\theta_{ijk}}{R_{ij}^2}\right) \times (q_{ix} - q_{jx}) + \left(\frac{1}{R_{ij}R_{ik}} - \frac{\cos\theta_{ijk}}{R_{ik}^2}\right) \times (q_{ix} - q_{kx})\right)
\end{aligned} \tag{A.6}$$

We define $A(\theta_{ijk}) = 1 + \cos(\theta_{ijk} - \theta_s)$ and $A(R_{ijk}) = e^{-\eta(\frac{R_{ij}+R_{ik}}{2} - R_s)^2} f_c(R_{ij}) f_c(R_{ik})$, and the partial derivatives of $A(\theta_{ijk})$ and $A(R_{ijk})$ with respect to q_{ix} are

$$\begin{aligned}
\frac{\partial A(\theta_{ijk})}{\partial q_{ix}} &= -\sin(\theta_{ijk} - \theta_s) \times \frac{\partial \theta_{ijk}}{\partial q_{ix}} \\
\frac{\partial A(R_{ijk})}{\partial R_{ij}} &= f_c(R_{ik})[f_c'(R_{ij}) - \eta(\frac{R_{ij} + R_{ik}}{2} - R_s) f_c(R_{ij}))e^{-\eta(\frac{R_{ij}+R_{ik}}{2} - R_s)^2}] \\
\frac{\partial A(R_{ijk})}{\partial R_{ik}} &= f_c(R_{ij})[f_c'(R_{ik}) - \eta(\frac{R_{ij} + R_{ik}}{2} - R_s) f_c(R_{ik}))e^{-\eta(\frac{R_{ij}+R_{ik}}{2} - R_s)^2}] \\
\frac{\partial A(R_{ijk})}{\partial q_{ix}} &= \frac{\partial A(R_{ijk})}{\partial R_{ij}} \times \frac{q_{ix} - q_{jx}}{R_{ij}} + \frac{\partial A(R_{ijk})}{\partial R_{ik}} \times \frac{q_{ix} - q_{kx}}{R_{ik}}
\end{aligned} \tag{A.7}$$

And finally we have

$$\begin{aligned}
\frac{\partial G_i^A}{\partial q_{ix}} &= 2^{1-\zeta} \sum_{j,k \neq i}^{\text{all}} \zeta A^{\zeta-1}(\theta_{ijk}) A(R_{ijk}) \times \frac{\partial A(\theta_{ijk})}{\partial q_{ix}} + A^{\zeta}(\theta_{ijk}) \times \frac{A(R_{ijk})}{\partial q_{ix}} \\
&= \sum_{j,k \neq i}^{\text{all}} A_1(R_{ijk}) \times (q_{ix} - q_{jx}) + A_2(R_{ijk}) \times (q_{ix} - q_{kx})
\end{aligned} \tag{A.8}$$

where

$$
\begin{aligned}
A_1(R_{ijk}) &= 2^{1-\zeta}[\zeta A^{\zeta-1}(\theta_{ijk})A(R_{ijk})\frac{\sin(\theta_{ijk}-\theta_s)}{\sqrt{1-(\cos\theta_{ijk})^2}}(\frac{1}{R_{ij}R_{ik}}-\frac{\cos\theta_{ijk}}{R_{ij}^2}) \\
&\quad + A^{\zeta}(\theta_{ijk})\frac{\partial A(R_{ijk})}{\partial R_{ij}}\times\frac{1}{R_{ij}}] \\
A_2(R_{ijk}) &= 2^{1-\zeta}[\zeta A^{\zeta-1}(\theta_{ijk})A(R_{ijk})\frac{\sin(\theta_{ijk}-\theta_s)}{\sqrt{1-(\cos\theta_{ijk})^2}}(\frac{1}{R_{ij}R_{ik}}-\frac{\cos\theta_{ijk}}{R_{ik}^2}) \\
&\quad + A^{\zeta}(\theta_{ijk})\frac{\partial A(R_{ijk})}{\partial R_{ik}}\times\frac{1}{R_{ik}}]
\end{aligned}
\tag{A.9}
$$

Now consider the partial derivative of G_j^A, where $j \neq i$. It's straightforward to obtain

$$
\begin{aligned}
\frac{\partial G_j^A}{\partial q_{ix}} &= 2^{1-\zeta}\sum_{k\neq i,j}^{\text{all}}\zeta A^{\zeta-1}(\theta_{jik})A(R_{jik})\times\frac{\partial A(\theta_{jik})}{\partial q_{ix}} + A^{\zeta}(\theta_{jik})\times\frac{\partial A(R_{jik})}{\partial q_{ix}} \\
&= \sum_{k\neq i,j}^{\text{all}} A_1(R_{jik})\times(q_{ix}-q_{jx}) + A_2(R_{jik})\times(q_{ix}-q_{kx})
\end{aligned}
\tag{A.10}
$$

where

$$
\begin{aligned}
\theta_{jik} &= \arccos(\frac{R_{ji}^2+R_{jk}^2-R_{ik}^2}{2R_{ji}R_{jk}}) \\
A(\theta_{jik}) &= 1+\cos(\theta_{jik}-\theta_s) \\
A(R_{jik}) &= e^{-\eta(\frac{R_{ji}+R_{jk}}{2}-R_s)^2}f_c(R_{ji})f_c(R_{jk})
\end{aligned}
\tag{A.11}
$$

The corresponding derivatives are

$$
\begin{aligned}
\frac{\partial A(\theta_{jik})}{\partial q_{ix}} &= \frac{\sin(\theta_{jik}-\theta_s)}{\sqrt{1-(\cos\theta_{jik})^2}}((\frac{1}{R_{ji}R_{jk}}-\frac{\cos\theta_{jik}}{R_{ji}^2})\times(q_{ix}-q_{jx}) - \frac{1}{R_{ji}R_{jk}}\times(q_{ix}-q_{kx})) \\
\frac{\partial A(R_{jik})}{\partial q_{ix}} &= \frac{\partial A(R_{jik})}{\partial R_{ji}}\times\frac{\partial R_{ji}}{\partial q_{ix}} \\
&= f_c(R_{jk})[f_c'(R_{ji}) - \eta(\frac{R_{ji}+R_{jk}}{2}-R_s)f_c(R_{ji})e^{-\eta(\frac{R_{ji}+R_{jk}}{2}-R_s)^2}]\times\frac{q_{ix}-q_{jx}}{R_{ji}}
\end{aligned}
\tag{A.12}
$$

And $A_1(R_{jik})$ and $A_2(R_{jik})$ can be obtained as

$$
\begin{aligned}
A_1(R_{jik}) &= 2^{1-\zeta}[\zeta A^{\zeta-1}(\theta_{jik})A(R_{jik})\frac{\sin(\theta_{jik}-\theta_s)}{\sqrt{1-(\cos\theta_{jik})^2}}(\frac{1}{R_{ji}R_{jk}}-\frac{\cos\theta_{jik}}{R_{ji}^2}) \\
&\quad + A^{\zeta}(\theta_{jik})\frac{\partial A(R_{jik})}{\partial R_{ji}}\times\frac{1}{R_{ji}}] \\
A_2(R_{jik}) &= -2^{1-\zeta}\zeta A^{\zeta-1}(\theta_{jik})A(R_{jik})\frac{\sin(\theta_{jik}-\theta_s)}{\sqrt{1-(\cos\theta_{jik})^2}}\frac{1}{R_{ji}R_{jk}}
\end{aligned}
\tag{A.13}
$$

References

[1] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," *Phys. Rev. Lett.*, vol. 98, p. 146401, Apr 2007.

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

[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, July 1959.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, May 2015.

[6] A. E. BRYSON, W. F. DENHAM, and S. E. DREYFUS, "Optimal programming problems with inequality constraints," *AIAA Journal*, vol. 1, pp. 2544–2550, Nov 1963.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[9] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1915–1929, Aug 2013.

[12] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 1799–1807, Curran Associates, Inc., 2014.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[14] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition Understanding*, pp. 196–201, Dec 2011.

[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, Nov 2012.

[16] T. N. Sainath, A. r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8614–8618, May 2013.

[17] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *CoRR*, vol. abs/1103.0398, 2011.

[18] A. Bordes, S. Chopra, and J. Weston, "Question answering with subgraph embeddings," *CoRR*, vol. abs/1406.3676, 2014.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[20] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *CoRR*, vol. abs/1412.2007, 2014.

[21] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, pp. i121–i129, Jun 2014. btu277[PII].

[22] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. C. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, Q. Morris, Y. Barash, A. R. Krainer, N. Jojic, S. W. Scherer, B. J. Blencowe, and B. J. Frey, "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, 2015.

[23] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure–activity relationships," *Journal of Chemical Information and Modeling*, vol. 55, no. 2, pp. 263–274, 2015. PMID: 25635324.

[24] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, pp. 168–174, Aug 2013. Article.

[25] E. Finkel, "Quantum chemistry, fourth edition (levine, ira n.)," *Journal of Chemical Education*, vol. 70, no. 5, p. A145, 1993.

[26] M. J. S. Dewar, E. G. Zoebisch, E. F. Healy, and J. J. P. Stewart, "Development and use of quantum mechanical molecular models. 76. am1: a new general purpose quantum mechanical molecular model," *Journal of the American Chemical Society*, vol. 107, no. 13, pp. 3902–3909, 1985.

[27] M. Elstner, "The scc-dftb method and its application to biological systems," *Theoretical Chemistry Accounts*, vol. 116, pp. 316–325, Aug 2006.

[28] W. Thiel, *Perspectives on Semiempirical Molecular Orbital Theory*, pp. 703–757. John Wiley & Sons, Inc., 2007.

[29] J. J. P. Stewart, "Application of the pm6 method to modeling proteins," *Journal of Molecular Modeling*, vol. 15, pp. 765–805, Jul 2009.

[30] F. R. Burden, M. G. Ford, D. C. Whitley, and D. A. Winkler, "Use of automatic relevance determination in qsar studies using bayesian neural networks," *Journal of Chemical Information and Computer Sciences*, vol. 40, no. 6, pp. 1423–1430, 2000. PMID: 11128101.

[31] J. Eickholt and J. Cheng, "Predicting protein residue–residue contacts using deep networks and boosting," *Bioinformatics*, vol. 28, no. 23, pp. 3066–3072, 2012.

[32] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," *Phys. Rev. Lett.*, vol. 108, p. 058301, Jan 2012.

[33] P. Raccuglia, K. C. Elbert, P. D. F. Adler, C. Falk, M. B. Wenny, A. Mollo, M. Zeller, S. A. Friedler, J. Schrier, and A. J. Norquist, "Machine-learning-assisted materials discovery using failed experiments," *Nature*, vol. 533, pp. 73 EP –, May 2016.

[34] J. Behler, "Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations," *Phys. Chem. Chem. Phys.*, vol. 13, pp. 17930–17955, 2011.

[35] O. A. von Lilienfeld, "First principles view on chemical compound space: Gaining rigorous atomistic control of molecular properties," *International Journal of Quantum Chemistry*, vol. 113, no. 12, pp. 1676–1689, 2013.

[36] K. Hansen, G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. von Lilienfeld, A. Tkatchenko, and K.-R. Müller, "Assessment and validation of machine learning methods for predicting molecular atomization energies," *Journal of Chemical Theory and Computation*, vol. 9, no. 8, pp. 3404–3419, 2013. PMID: 26584096.

[37] H. Geppert, T. Horváth, T. Gärtner, S. Wrobel, and J. Bajorath, "Support-vector-machine-based ranking significantly improves the effectiveness of similarity searching using 2d fingerprints and multiple reference compounds," *Journal of Chemical Information and Modeling*, vol. 48, no. 4, pp. 742–746, 2008. PMID: 18318473.

[38] R. Todeschini and V. Consonni, *Handbook of molecular descriptors. WileyVCH, Weinheim*, vol. 11. Wiley-VCH, 2000, 01 2000.

[39] G. Schneider, "Virtual screening: an endless staircase?," *Nature Reviews Drug Discovery*, vol. 9, pp. 273 EP –, Apr 2010. Perspective L3 -.

[40] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, "Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons," *Phys. Rev. Lett.*, vol. 104, p. 136403, Apr 2010.

[41] T. D. Huan, A. Mannodi-Kanakkithodi, and R. Ramprasad, "Accelerated materials property predictions and design using motif-based fingerprints," *Phys. Rev. B*, vol. 92, p. 014106, Jul 2015.

[42] D. Jasrasaria, E. O. Pyzer-Knapp, D. Rappoport, and A. Aspuru-Guzik, "Space-filling curves as a novel crystal structure representation for machine learning models," 08 2016.

[43] S. Manzhos, X. Wang, R. Dawes, and T. Carrington, "A nested molecule-independent neural network approach for high-quality potential fits," *The Journal of Physical Chemistry A*, vol. 110, no. 16, pp. 5295–5304, 2006. PMID: 16623455.

[44] J. Behler, "Constructing high-dimensional neural network potentials: A tutorial review," *International Journal of Quantum Chemistry*, vol. 115, no. 16, pp. 1032–1050, 2015.

[45] J. B. Witkoskie and D. J. Doren, "Neural network models of potential energy surfaces: prototypical examples," *Journal of Chemical Theory and Computation*, vol. 1, no. 1, pp. 14–23, 2005. PMID: 26641111.

[46] A. Pukrittayakamee, M. Malshe, M. Hagan, L. M. Raff, R. Narulkar, S. Bukkapatnum, and R. Komanduri, "Simultaneous fitting of a potential-energy surface and its corresponding force fields using feedforward neural networks," *The Journal of Chemical Physics*, vol. 130, no. 13, p. 134101, 2009.

[47] H. M. Le and L. M. Raff, "Molecular dynamics investigation of the bimolecular reaction beh + h2 → beh2 + h on an ab initio potential-energy surface obtained using neural network methods with both potential and gradient accuracy determination," *The Journal of Physical Chemistry A*, vol. 114, no. 1, pp. 45–53, 2010. PMID: 19852450.

[48] A. Pukrittayakamee, M. Hagan, L. Raff, S. T. S. Bukkapatnam, and R. Komanduri, "Practical training framework for fitting a function and its derivatives," *IEEE Transactions on Neural Networks*, vol. 22, pp. 936–947, June 2011.

[49] M. Gastegger, J. Behler, and P. Marquetand, "Machine learning molecular dynamics for the simulation of infrared spectra," *Chem. Sci.*, vol. 8, pp. 6924–6935, 2017.

[50] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "Ufldl tutorial," 2010.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[52] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[53] F. Neese, "The orca program system," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 2, no. 1, pp. 73–78, 2012.

[54] J. P. Perdew, "Erratum: Density-functional approximation for the correlation energy of the inhomogeneous electron gas," *Phys. Rev. B*, vol. 34, pp. 7406–7406, Nov 1986.

[55] A. D. Becke, "Density-functional exchange-energy approximation with correct asymptotic behavior," *Phys. Rev. A*, vol. 38, pp. 3098–3100, Sep 1988.

[56] M. Orio, D. A. Pantazis, and F. Neese, "Density functional theory," *Photosynthesis Research*, vol. 102, no. 2, pp. 443–453, 2009.

[57] T. Fink, H. Bruggesser, and J.-L. Reymond, "Virtual exploration of the small-molecule chemical universe below 160 daltons," *Angewandte Chemie International Edition*, vol. 44, no. 10, pp. 1504–1508, 2005.

[58] T. Fink and J.-L. Reymond, "Virtual exploration of the chemical universe up to 11 atoms of c, n, o, f: assembly of 26.4 million structures (110.9 million stereoisomers) and analysis for new ring systems, stereochemistry, physicochemical properties, compound classes, and drug discovery," *Journal of Chemical Information and Modeling*, vol. 47, no. 2, pp. 342–353, 2007. PMID: 17260980.

[59] L. M. Raff, M. Malshe, M. Hagan, D. I. Doughan, M. G. Rockley, and R. Komanduri, "Ab initio potential-energy surfaces for complex, multichannel systems using modified novelty sampling and feedforward neural networks," *The Journal of Chemical Physics*, vol. 122, no. 8, p. 084104, 2005.