



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Master of Science

The development of neural network potentials
for cubic sodium chloride crystals with vacancies

The Graduate School
of the University of Ulsan
Department of Physics
Doan Thi Xuan Loan

The development of neural network potentials
for cubic sodium chloride crystals with vacancies

Supervisor: Prof. Young-Han Shin

This dissertation is submitted to University of Ulsan in partial fulfillment of
the requirements for the degree of
Master of Science in physics

by

Doan Thi Xuan Loan

Department of Physics

University of Ulsan

January, 2020

University of Ulsan

Department of Physics

This dissertation, "The development of neural network potentials for cubic sodium chloride crystals with vacancies" is hereby approved in partial fulfillment of the requirements for the degree of Master of Science in physics.



Committee Chair: Prof. Tae Heon Kim
Department of Physics,
University of Ulsan



Examiner: Assist. Prof. Kyoungmin Min
Department of Mechanical Engineering,
Soongsil University



Supervisor: Prof. Young-Han Shin
Department of Physics,
University of Ulsan

The development of neural network potentials for cubic sodium chloride crystals with vacancies



Doan Thi Xuan Loan

Department of Physics

University of Ulsan

This dissertation is submitted to University of Ulsan in partial fulfillment of the requirements for the degree of

Master of Science

January 20, 2020

Dedicated to my loving parents

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

Doan Thi Xuan Loan

January 20, 2020

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor Prof. Young-Han Shin for your support and understanding over these past two years in the Multiscale Materials Modeling Lab (M3L). Without your patient guidance and useful critiques, this thesis would have never been accomplished. I wish to extend my thanks to all the members of M3L, especially Tan-Lien Pham, a friend as well as my sister, who guide me in both scientific research work and life in general. I also would like to thank all the members of the BK21 and Physics Department Office for helping me prepare the required documents during my studying period time at the University of Ulsan.

Besides academic support, I always appreciate my friendship with Pham Thi Hoa with whom I can share my joys and sorrows. I am immensely grateful to Tran Van Loi, who sees my anxieties but frees my spirit. And all the fantastic memories in Korea with N.N.Hoang Anh, T.H.Vu Anh, etc. will be engraved on my mind.

Last but not least, I am indebted to my family for their huge support and enthusiastic encouragement in spiritually throughout my study.

Abstract

We implement a method for computing the interatomic potentials by training and fitting neural networks with the data obtain from molecular dynamics simulations. We construct a long-range neural network potential and apply it to NaCl system including ideal and defected systems. We keep the short-range part with Behler type and the long-range part is calculated using the Ewald summation. The reason for choosing the Ewald sum is because it provides high accuracy and feasible computational speed when estimating the long-range potential, due to the rapid convergence between long-range contribution in reciprocal space and the short-range contribution in real space. In this work, we not only compare the result to molecular dynamics simulations but also adopt them into the training set of neural network.

Contents

Contents	xi
List of Figures	xiii
List of Tables	xv
1 Introduction	1
2 Ewald summation	3
2.1 Why Ewald sum?	3
2.2 Ewald summation	4
2.2.1 Problem statement	4
2.2.2 Splitting the charge distributions	6
2.2.3 Short-range potential in real space	7
2.2.4 Long-range potential in reciprocal space	7
3 Fundamental of Artificial Neural Networks	11
3.1 Perceptron	12
3.2 Feed-forward neural network	14
3.3 Activation function	16
3.3.1 Binary step function	16
3.3.2 Linear activation function	16
3.3.3 Non-linear activation functions	17
3.4 Backpropagation	18
4 Constructing Long-range Neural Network Potentials	23
4.1 Conventional NNPs	23
4.2 High-dimensional NNPs	24
4.3 Long-range neural network potentials	27

5	Training Neural Network Potentials for NaCl	31
5.1	Preparing for the model	31
5.2	Optimizer and learning rate	33
5.3	Overfitting problem	35
5.3.1	Ridge Regression	36
5.3.2	Lasso Regression	36
5.3.3	Elastic Net	37
5.3.4	Early Stopping	37
5.4	Constraint in the model weights	38
5.5	Results and discussion	39
6	Conclusion	43
	References	45

List of Figures

1.1	Workflow of the NNPs.	2
2.1	Charge distribution in the Ewald sum (<i>Ewald summation techniques in perspective: a survey</i> . Computer Physics Communications, 95(2):73–92, 1996). .	4
2.2	Building up the sphere of simulation boxes. Each copy of the periodic box is represented by a small square. The shaded region represents the external dielectric continuum of relative permittivity ϵ_s (<i>Computer simulation of liquids</i> . Oxford University Press, Oxford, United Kingdom, second edition, 2017). .	5
3.1	ANNs performing simple logical computations (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).	12
3.2	a) Threshold logic unit and b) single layer perceptron (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).	13
3.3	Multi-layer perceptron.	14
3.4	Feed-forward neural network with the only one output value.	15
3.5	Activation function: (a) Binary step function, (b) Linear activation function, (c) ReLU activation function, (d) Sigmoid activation function, (e) Hyperbolic Tangent activation function.	16
4.1	Conventional NNPs.	24
4.2	Structure of high-dimensional NNPs for system containing three atoms. . . .	25
4.3	Example of an initial radial symmetry function set G^2 with cutoff radius $R_c = 6\text{\AA}$ and $R_s = 0$	26
4.4	Structure of long-range NNPs.	28
5.1	Ideal NaCl and defected NaCl.	31
5.2	Learning curves for various learning rates.	35

5.3	Early stopping regularization (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).	38
5.4	Constrain on the long-range part.	39
5.5	Comparison of DFT energy and predicted energy with the high-dimensional NNPs and long-range NNPs applied to ideal and defected sodium chloride systems.	40

List of Tables

5.1	Symmetry function parameters for G^2 and G^5	32
5.2	The hyperparameters of the model NNPs. The learning rate was optimal using exponential scheduling method.	40
5.3	Long-range fitting parameters and RMSE for the ideal NaCl and the defected NaCl.	41
5.4	Comparison of the computational times (unit: second) of the high-dimensional NNP and the long-range NNP for the ideal NaCl and defected NaCl.	41

Chapter 1

Introduction

An accurate description of the potential-energy surfaces (PESs) is an essential ingredient to carry out the reliable MD simulations [1]. The PESs describes the energy of a system in terms of certain parameters, normally the positions of the atoms. The *ab initio* method using density-functional theory can give quite accurate PES of many systems. It proposes to calculate electronic wave functions as the solution of the Schrödinger equation in the Born–Oppenheimer approximation. The electronic wave functions can be split into many elementary functions approximated with only one-electron functions which are the linear combination of a finite set of basis functions. However, since this approach can render the exact solution with the high numerical burden, it confines the computation in simulation time to restrict the number of particles [2]. Another solution is classical force fields which divides the total energy into bonded and non-bonded interactions. Low dimensional bonds, bonding angles, and dihedral angles are taken into account in the bonded terms, whereas the non-bonded term describing the electrostatic and van der Waals interactions. The most popular drawback of classical force fields is the difficulty in simulating the construction and dissociation of bonds. A new promising approach to compute PES is *machine learning* (ML) techniques without a direct physical meaning, which will be studied in this thesis.

The fundamental for ML was laid down in the middle of the last century. However, the rapid development in this technique drives an explosion of ML applications in everything, including physics. ML with the development of *neural network potentials* (NNPs) has opened a new direction to deal with PES since the old methods facing difficulties. In principle, NNPs need to fit an analytic expression to a set of reference data obtained in electronic structure calculations as accurately as possible [3]. In simple words, each set of atomic positions gives a corresponding energy, through the NN. This relationship will be expressed by a function of weight parameters, and the final task is to find the optimal weight parameters for the NN. The workflow of the NNPs is given in Fig. 1.1, this process is repeated until the best weights are

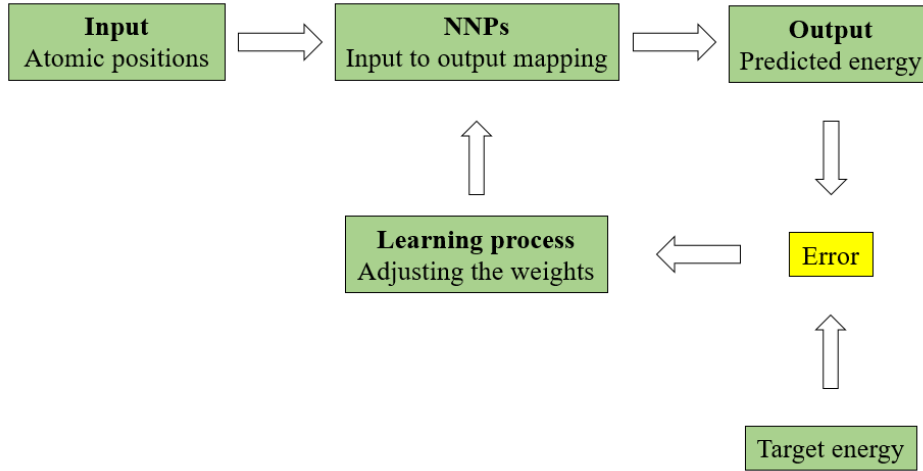


Figure 1.1: Workflow of the NNPs.

gained. So far some small size systems have been constructed with many frozen degrees of freedom to reduce complexity. However, one requirement is that the potential energy values must be invariant under permutations of the same atoms. We need to handle this problem to construct physically correct PESs and improve the efficiency of the fitting process. The high-dimensional NN which can overcome this issue has been proposed by Behler and Parrinello in 2007. This NNPs represent well the short-range energy which is the sum of atomic energies and crucially depends on the local environment of each atom. The local environment is characterized by a set of symmetry function values converting from the Cartesian coordinates of atomic positions. Each element accompanies a specific NN architecture, making the resulting PES fully permutation invariant. The high-dimensional NNPs are also able to scale for large systems of up to thousands of atoms with much less computational time compared to the ab initio calculations. Furthermore, it does not require the exact functional form of PES [4].

We constructed the high-dimensional NNPs for ideal and defected sodium chloride crystals. This model performs well in ideal case. However, the transferability to defected system is not good. This might be due to the truncation in the cutoff radius which can yield only the short range energy. Therefore, we consider the long-range energy based on the model of Behler-Parrinello type.

Chapter 2

Ewald summation

2.1 Why Ewald sum?

In classical MD simulations, the potential energy function is denoted as the force field to consider the bonds and non-bonded interactions. In this chapter, we only discuss the latter. The non-bond contributions can be the van der Waals interaction, which is built in the Lennard-Jones potential (V_{LJ}) or the electrostatic interaction (V_C) from Coulomb's Law [5]. The equations of these potentials are:

$$V_{LJ} = \frac{4\epsilon\sigma^{12}}{r^{12}} - \frac{4\epsilon\sigma^6}{r^6} = \frac{A_{rep}}{r^{12}} - \frac{B_{disp}}{r^6}, \quad (2.1)$$

where the first term is the repulsive term and the second is the attractive long-range term describing dispersion force and

$$V_C = \frac{Q}{4\pi\epsilon_0 r}, \quad (2.2)$$

where Q is the charge. The main drawback of V_C is the slow convergence with distance in the periodic systems. The closed forms of these potential are not known since they are defined with any distance larger than 0, but they must be truncated at some cutoff distance to be evaluated:

$$V_{trunc}(r) = \begin{cases} V(r) - V(r_c) & \text{for } r \leq r_c \\ 0 & \text{for } r > r_c, \end{cases} \quad (2.3)$$

where r_c is the cutoff radius.

In addition, there is another problem in Coulomb potential. The interaction between ions decays in the inverse power of r , but the number of interacting ions at the radius r is pro-

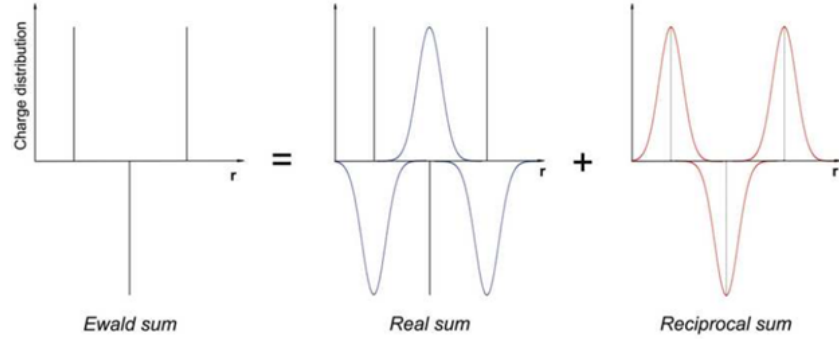


Figure 2.1: Charge distribution in the Ewald sum (*Ewald summation techniques in perspective: a survey*. Computer Physics Communications, 95(2):73–92, 1996).

portional to the surface area of a sphere, which is given by $4\pi r^2$ [6]. Therefore, the energy density interaction increases with distance. An alternative solution that can handle all these problems is an Ewald summation. The idea is that transforming a single conditionally and slowly convergent sum into two rapidly convergent sums.

2.2 Ewald summation

The Ewald summation, or Ewald sum for short, is a technique to sum long-range interactions between particles and all their infinite periodic images efficiently [7]. The long-range interaction in this method is split into two parts: a short-range contribution, which is calculated in real space, and a long-range contribution which is using Fourier transform to compute. As in Fig. 2.1, the charge distributions in the real space and reciprocal space cancel each other and the remainder is Ewald sum. Assuming that the short-range part can be summed easily, the main challenge is to obtain the long-range term. Since the long-range term is computed from the Fourier sum, the system is infinitely periodic. One unit cell is chosen as the "central cell" and the remaining cells are called images. The method of Ewald sum will be explained in detail in the next following sections.

2.2.1 Problem statement

Consider we have N ions at locations $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N$ and they are imposed to be in periodic boundary conditions. Then, the potential energy can be written as

$$E = \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{m \in \mathbb{Z}^3} \sum_{i=1}^N \sum_{j=1}^N \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{m}L|}, \quad (2.4)$$

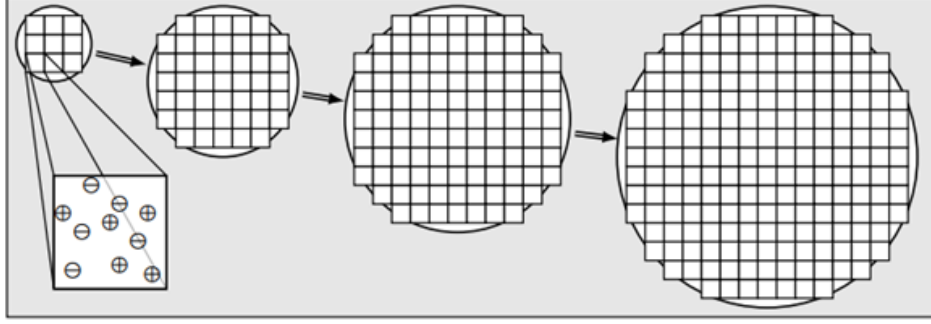


Figure 2.2: Building up the sphere of simulation boxes. Each copy of the periodic box is represented by a small square. The shaded region represents the external dielectric continuum of relative permittivity ϵ_s (*Computer simulation of liquids*. Oxford University Press, Oxford, United Kingdom, second edition, 2017).

where q_i, q_j are charges and $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$. The sum is over all ionic pairs (i, j) and the factor of $\frac{1}{2}$ prevents double counting. The vector $\mathbf{m} = (m_x, m_y, m_z)$ is over all triples of integers and we ignore $\mathbf{m} = 0$, for $i = j$. For cubic box, the center of each box in the periodic array is represented by $\mathbf{m}L$. Increasing the values m_x, m_y , and m_z to infinite, we can construct a infinite sphere system as in Fig. 2.2.

Electric potential due to an ion with charge q_i at \mathbf{r}_i is

$$\phi_i(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|}. \quad (2.5)$$

Electric potential due to N ions with their periodic images is

$$\phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \sum_{\mathbf{m}} \sum_{j=1}^N \frac{q_j}{|\mathbf{r} - \mathbf{r}_j + \mathbf{m}L|}. \quad (2.6)$$

Electric potential due to all ions excluding ion i is

$$\phi_{[i]}(\mathbf{r}) \equiv \phi(\mathbf{r}) - \phi_i(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \sum_{\mathbf{m}}' \sum_{j=1}^N \frac{q_j}{|\mathbf{r} - \mathbf{r}_j + \mathbf{m}L|}, \quad (2.7)$$

where $i = j$ is omitted. The potential energy in the Eq. 2.4 can be written as

$$E = \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{i=1}^N q_i \phi_{[i]}(\mathbf{r}_i). \quad (2.8)$$

The Ewald method will evaluate E by transforming it into summations that converge rapidly and absolutely.

2.2.2 Splitting the charge distributions

The Poisson's equation for electrostatics is

$$\nabla^2 \phi_i(\mathbf{r}) = -\frac{\rho_i(\mathbf{r})}{\epsilon_0}, \quad (2.9)$$

where $\rho_i(\mathbf{r}) = q_i \delta(\mathbf{r} - \mathbf{r}_i)$ is the charge density at the point charge q_i and the potential $\phi_i(\mathbf{r})$ is the solution of Poisson's equation which now can be given as the form

$$\phi_i(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \int \frac{\rho_i(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d^3\mathbf{r}'. \quad (2.10)$$

The total potential energy in the Eq. 2.8 is given by the new expression

$$\begin{aligned} E &= \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{\mathbf{m}}' \sum_{i=1}^N \sum_{j=1}^N \iint \frac{\rho_i(\mathbf{r}) \rho_j(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}' + \mathbf{m}L|} d^3\mathbf{r} d^3\mathbf{r}' \\ &= \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{\mathbf{m}}' \sum_{i=1}^N \sum_{j=1}^N q_i \int \frac{\rho_j(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}' + \mathbf{m}L|} d^3\mathbf{r}'. \end{aligned} \quad (2.11)$$

Splitting the charge distributions by adding and subtracting a Gaussian distribution [8], we have

$$\begin{aligned} \rho_i(\mathbf{r}) &= [q_i \delta(\mathbf{r} - \mathbf{r}_i) - q_i G_\sigma(\mathbf{r} - \mathbf{r}_i)] + q_i G_\sigma(\mathbf{r} - \mathbf{r}_i) \\ &= \rho_i^S(\mathbf{r}) + \rho_i^L(\mathbf{r}), \end{aligned} \quad (2.12)$$

where $G_\sigma(\mathbf{r}) = \frac{1}{(2\pi\sigma^2)^{3/2}} \exp\left(-\frac{|\mathbf{r}|^2}{2\sigma^2}\right)$, and σ is the standard deviation of the Gaussian distribution. For convenience, we use another parameter $\kappa = \frac{1}{\sqrt{2}\sigma}$ indicating the width of the distribution. Hence, $G_\kappa(\mathbf{r}) = \frac{\kappa^3}{\pi^{3/2}} \exp\left(-\kappa^2 |\mathbf{r}|^2\right)$. Likewise, the electric potential can be divided into

$$\begin{aligned} \phi_i(\mathbf{r}) &= \frac{q_i}{4\pi\epsilon_0} \int \frac{\delta(\mathbf{r} - \mathbf{r}') - G_\kappa(\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d^3\mathbf{r}' + \frac{q_i}{4\pi\epsilon_0} \int \frac{G_\kappa(\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d^3\mathbf{r}' \\ &= \phi_i^S(\mathbf{r}) + \phi_i^L(\mathbf{r}), \end{aligned} \quad (2.13)$$

and also the total energy

$$\begin{aligned} E &= \frac{1}{2} \sum_{i=1}^N q_i \phi_i^S(\mathbf{r}_i) + \frac{1}{2} \sum_{i=1}^N q_i \phi_i^L(\mathbf{r}_i) \\ &= \frac{1}{2} \sum_{i=1}^N q_i \phi_i^S(\mathbf{r}_i) + \frac{1}{2} \sum_{i=1}^N q_i \phi_i^L(\mathbf{r}_i) - \frac{1}{2} \sum_{i=1}^N q_i \phi_i^L(\mathbf{r}_i) \\ &= E^S + E^L - E^{self}. \end{aligned} \quad (2.14)$$

2.2.3 Short-range potential in real space

The solution of the Poisson's equation in sphere coordinates is $\phi_\kappa(\mathbf{r}) = \frac{q}{4\pi\epsilon_0 r} \text{erf}(\kappa r)$, where $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \equiv 1 - \text{erfc}(z)$ [9]. Thus,

$$\phi_i^S(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|} \text{erfc}(\kappa |\mathbf{r} - \mathbf{r}_i|) \quad (2.15)$$

$$\phi_i^L(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|} \text{erfc}(\kappa |\mathbf{r} - \mathbf{r}_i|), \quad (2.16)$$

and

$$\phi_{[i]}^S(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \sum_{\mathbf{m}}' \sum_{j=1}^N \frac{q_j}{|\mathbf{r} - \mathbf{r}_j + \mathbf{m}L|} \text{erfc}(\kappa |\mathbf{r} - \mathbf{r}_j + \mathbf{m}L|). \quad (2.17)$$

The short-range term in the Ewald sum is

$$\begin{aligned} E^S &= \frac{1}{2} \sum_{i=1}^N q_i \phi_{[i]}^S(\mathbf{r}_i) \\ &= \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{\mathbf{m}}' \sum_{i=1}^N \sum_{j=1}^N \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{m}L|} \text{erfc}(\kappa |\mathbf{r}_{ij} + \mathbf{m}L|). \end{aligned} \quad (2.18)$$

2.2.4 Long-range potential in reciprocal space

From Eq. 2.16 and the fact that $\lim_{z \rightarrow 0} \text{erf}(z) = \frac{2}{\sqrt{\pi}} z$ [9], the electric field is

$$\phi_i^L(\mathbf{r}_i) = \frac{q_i}{4\pi\epsilon_0} \frac{2}{\sqrt{\pi}} \frac{1}{\sqrt{2}\kappa}, \quad (2.19)$$

and therefore the self-term energy is

$$E^{self} = \frac{1}{4\pi\epsilon_0} \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^N q_i^2. \quad (2.20)$$

Because the long-range energy cannot be directly computed in the real space, the Ewald sum will transform it into a sum in the reciprocal space. Since the potential $\phi^L(\mathbf{r}_i)$ is generated by a periodic array of ions, the total charge density is

$$\rho^L(\mathbf{r}) = \sum_{\mathbf{m}} \sum_{i=1}^N \rho_i^L(\mathbf{r} + \mathbf{m}L). \quad (2.21)$$

Applying Fourier transformation for $\phi^L(\mathbf{r}_i)$ and $\rho^L(\mathbf{r})$, we have

$$\hat{\phi}^L(\mathbf{k}) = \int_V \phi^L(\mathbf{r}) \exp(-i\mathbf{k} \cdot \mathbf{r}) d^3\mathbf{r}, \quad (2.22)$$

$$\hat{\rho}^L(\mathbf{k}) = \int_V \rho^L(\mathbf{r}) \exp(-i\mathbf{k} \cdot \mathbf{r}) d^3\mathbf{r}. \quad (2.23)$$

The inverse Fourier transform give

$$\phi^L(\mathbf{r}) = \frac{1}{V} \sum_{\mathbf{k}} \hat{\phi}^L(\mathbf{k}) \exp(i\mathbf{k} \cdot \mathbf{r}), \quad (2.24)$$

$$\rho^L(\mathbf{r}) = \frac{1}{V} \sum_{\mathbf{k}} \hat{\rho}^L(\mathbf{k}) \exp(i\mathbf{k} \cdot \mathbf{r}). \quad (2.25)$$

Once again, the electric field and the charge distribution are related by the Poisson's equation $\nabla^2 \phi^L(\mathbf{r}) = -\frac{\rho^L(\mathbf{r})}{\epsilon_0}$, which can be transformed into reciprocal space $k^2 \hat{\phi}^L(\mathbf{k}) = \frac{\hat{\rho}^L(\mathbf{k})}{\epsilon_0}$. The Fourier transform of the charge density is

$$\rho^L(\mathbf{r}) = \sum_m \sum_{j=1}^N q_j G_{\kappa}(\mathbf{r} - \mathbf{r}_j + \mathbf{m}L), \quad (2.26)$$

and

$$\begin{aligned} \hat{\rho}^L(\mathbf{k}) &= \int_V \sum_{j=1}^N q_j G_{\kappa}(\mathbf{r} - \mathbf{r}_j + \mathbf{m}L) \exp(-i\mathbf{k} \cdot \mathbf{r}) d^3\mathbf{r} \\ &= \sum_{j=1}^N q_j \int_{\mathbf{R}^3} G_{\kappa}(\mathbf{r} - \mathbf{r}_j) \exp(-i\mathbf{k} \cdot \mathbf{r}) d^3\mathbf{r} \\ &= \sum_{j=1}^N q_j \exp(-i\mathbf{k} \cdot \mathbf{r}_j) \exp(-k^2/4\kappa^2). \end{aligned} \quad (2.27)$$

The potential in the reciprocal space is

$$\hat{\phi}^L(\mathbf{k}) = \frac{1}{k^2 \epsilon_0} \sum_{j=1}^N q_j \exp(-i\mathbf{k} \cdot \mathbf{r}_j) \exp(-k^2/4\kappa^2), \quad (2.28)$$

and the inverse Fourier transform gives

$$\begin{aligned} \phi^L(\mathbf{r}) &= \frac{1}{V} \sum_{\mathbf{k} \neq 0} \hat{\phi}^L(\mathbf{k}) \exp(i\mathbf{k} \cdot \mathbf{r}) \\ &= \frac{1}{V \epsilon_0} \sum_{\mathbf{k} \neq 0} \sum_{j=1}^N \frac{q_j}{k^2} \exp(-i\mathbf{k} \cdot (\mathbf{r} - \mathbf{r}_j)) \exp(-k^2/4\kappa^2). \end{aligned} \quad (2.29)$$

We ignore $k = 0$ because the contribution to this term is zero if the supercell is charge neutral,

i.e. $\sum_{i=1}^N q_i = 0$. Hence, the long-range energy is

$$\begin{aligned} E^L &= \frac{1}{2} \sum_{i=1}^N q_i \phi^L(\mathbf{r}_i) \\ &= \frac{1}{2L^3 \epsilon_0} \sum_{\mathbf{k} \neq 0} \sum_{i=1}^N \sum_{j=1}^N \frac{q_i q_j}{k^2} \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \exp(-\mathbf{k}^2/4\kappa^2). \end{aligned} \quad (2.30)$$

Consider the dielectric constant ϵ_s which is related to the nature of medium surrounding the sphere, the medium is conductive as $\epsilon_s = \infty$ and vacuum as $\epsilon_s = 1$. The energy difference between these two cases is the dipole energy and should be included in the Ewald sum [10]:

$$E(\epsilon_s = 1) - E(\epsilon_s = \infty) = \frac{1}{6L^3 \epsilon_0} \left| \sum_i q_i \mathbf{r}_i \right|^2. \quad (2.31)$$

Combining all Eqs. 2.18, 2.20, 2.31, and 2.30, and for convenience, omitting $4\pi\epsilon_0$ for non-SI unit of charge, the final result is the total sum of the real-space term, the reciprocal-space term, the self term, and the surface term.

$$\begin{aligned} E_{Ewald} &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N q_i q_j \left(\sum'_{|m| \neq 0} \frac{\text{erfc}(\kappa |\mathbf{r}_{ij} + \mathbf{m}L|)}{|\mathbf{r}_{ij} + \mathbf{m}L|} \right. \\ &\quad \left. + \frac{1}{\pi L^3} \sum_{k \neq 0} \frac{4\pi^2}{k^2} \exp(-\mathbf{k}^2/4\kappa^2) \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \right) \\ &\quad - \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^N q_i^2 + \frac{2\pi}{3L^3} \left| \sum_{i=1}^N q_i \mathbf{r}_i \right|^2 \\ &\equiv E_{real} + E_{recip} - E_{self} + E_{sur}. \end{aligned} \quad (2.32)$$

Chapter 3

Fundamental of Artificial Neural Networks

In the 1950s, Arthur Samuel created the first computer program which can self-learn, and the term "machine learning (ML)" was coined concurrently with the demonstration of the basic concept of *artificial intelligence*. We can understand ML as a science of programming computer that is used to make decisions or predictions based on training data without building a specific algorithm for each task. Due to great capability, nowadays, the work on ML becomes widespread in the last two decades with many applications in different fields such as image recognition, robot control, etc. However, the accuracy of ML crucially depends on data, which means it can not give the expected result when there is a lack of data in training data set or bad data chosen.

ML can be classified in many different ways such as the basement on the task that it solves, the difference in its approach, etc. The common types of ML are the following:

- Supervised learning: The training data contain both the inputs and the desired outputs called labels. There are two typical tasks: classification and regression. In the former algorithm, the outputs are limited to a set of "class" while in the later, the outputs have any value predicted from the mapping inputs.
- Unsupervised learning: The training data are unlabeled. Alternatively, the computer tries to analyze the input data in order to find the rules or detect patterns.
- Semi-supervised learning: Falling between two types of above, in this learning, the training data are able to contain both labeled data and unlabeled data.
- Reinforcement learning: The learning system called "agent" studies from the environment, and makes the actions that can get the "reward" or "penalty". This process repeats

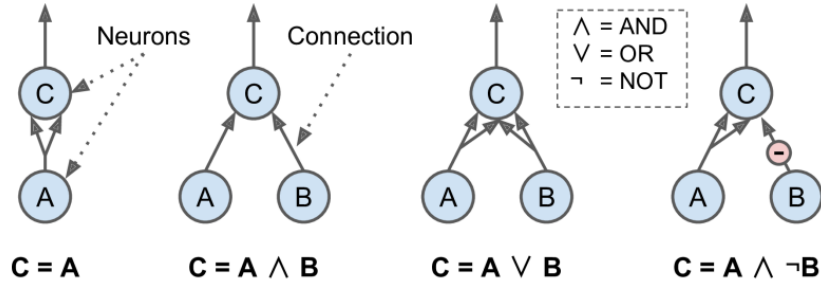


Figure 3.1: ANNs performing simple logical computations (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).

until the agent learns all the possible states.

One of the branches in ML is *deep learning* which is based on artificial neural networks (ANNs). In deep learning, the information is transformed from input to output, and this chain is called "credit assignment path". In the feed-forward NN, for example, the depth of credit assignment path (the number of hidden layers plus output layer) is larger than two. In other words, the NN needs to have two or more hidden layers. Deep learning can be supervised learning, unsupervised learning, or semi-supervised learning [11].

3.1 Perceptron

ANNs inspired by our brains are the core of deep learning. The very first precursor of ANNs is the network of *artificial neurons* performing simple logical computations. In 1943, Warren McCulloch and Walter Pitts introduced the artificial neuron containing only one output, and it will be activated when more than a number of binary inputs are activated. The perceptron algorithm will converge if the two data layers are linearly separable. However, it is quickly proven to be impossible to solve simple problems. Fig. 3.1 shows examples of ANNs that perform simple logical computations with artificial neurons.

Over a decade later, an upgrade version of the artificial neuron called "threshold logic unit" or "linear threshold unit" was invented by Frank Rosenblatt, which is the fundamental element of "perceptron". Frank Rosenblatt added a weight which is the value to accentuate the role of each input for associated output [12]. The structure of the threshold logic unit is shown in Fig. 3.2a. The net of inputs is calculated by multiplying the input x_i with the weight w_i and then add them up as follow:

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n = \mathbf{xw}. \quad (3.1)$$

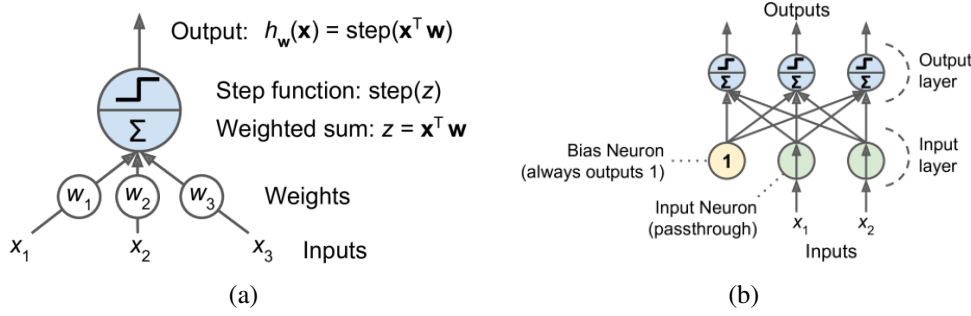


Figure 3.2: a) Threshold logic unit and b) single layer perceptron (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).

The output value is generated after applying z into an activation function. An alternative activation function for perceptron is a *Heaviside step function*

$$h(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} . \quad (3.2)$$

The perceptron models are used in binary classification problems, if the linear combination of inputs exceeds a threshold the output is either a positive or negative instance. Fig. 3.2b presents a *single layer perceptron* with an additional bias neuron. It allows shifting the decision boundary to the left or right, away from the origin, and independent of input values. The summation in the Eq. 3.1 can be written as

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = \mathbf{xw} + \mathbf{b}, \quad (3.3)$$

where $x_0 = 1$. There might be more complicated problems that require a complex relationship between the inputs and outputs. Therefore, it is necessary to insert one or more hidden layers between them. This kind of structure called "multi-layer perceptron" as demonstrated in Fig. 3.3. The biases are included in every layer except for the output layer, and each layer is fully connected with the next layer from the input through the output layer. The *backpropagation* algorithm introduced by David Rumelhart, Geoffrey Hinton, and Ronald Williams in 1986 is used to train the multi-layer perceptron model. The backpropagation computes the gradient of the loss function corresponding to the weights of the network. The optimization method uses this gradient to update the weights and minimize the loss function.

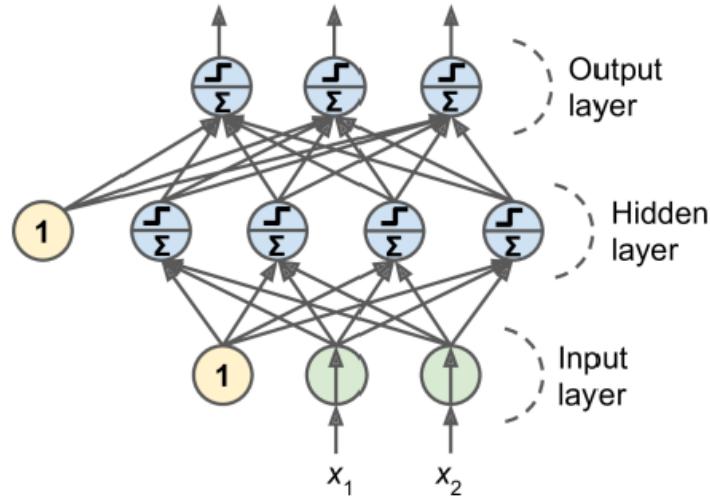


Figure 3.3: Multi-layer perceptron.

3.2 Feed-forward neural network

The simplest type of neural network is a *feed-forward neural network* in which the information moves in one direction from the input layer to output layer. Between the input and output layer are hidden layers, and each layer contains nodes which are connected by the weights. A particular layer just has connections with the adjacent layers and each node in a layer fully connects to all nodes in the subsequent layer. All of the layers and the number of nodes per layer define the architecture of the NN. An example of the FFNN 2-4-4-1 is shown in Fig. 3.4. The node number i in the layer j is computed by equation:

$$y_i^j = b_i^j + \sum_{k=1}^{N_{j-1}} w_{k,i}^{j-1,j} \cdot y_k^{j-1}, \quad (3.4)$$

where N_{j-1} is the number of nodes in layer $j-1$, and $w_{k,i}^{j-1,j}$ is the weight connecting the node i in layer j with the node k in the previous layer $j-1$. This equation is the linear combination of all nodes in the previous layer, which is shifted by the bias weight b_i^j . For instance, we can evaluate each node i in the first and second hidden layers as

$$y_j^1 = b_j^1 + \sum_{i=1}^2 w_{ij}^{01} \cdot x_i^0 = \mathbf{b}^1 + \mathbf{w}^{01} \mathbf{x}, \quad (3.5)$$

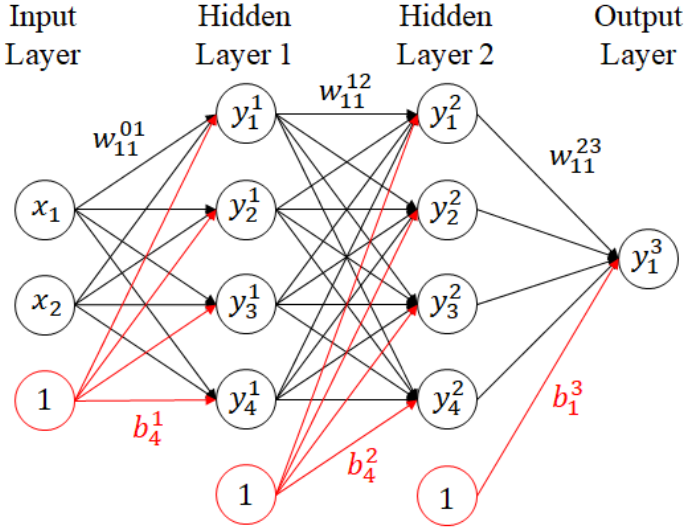


Figure 3.4: Feed-forward neural network with the only one output value.

$$\begin{aligned}
 y_k^2 &= b_k^2 + \sum_{j=1}^4 w_{kj}^{12} \cdot y_j^1 \\
 &= b_k^2 + \sum_{j=1}^4 w_{jk}^{12} \cdot \left(b_j^1 + \sum_{i=1}^2 w_{ij}^{01} \cdot x_i^0 \right) \\
 &= \mathbf{b}^2 + \mathbf{w}^{12} (\mathbf{b}^1 + \mathbf{w}^{01} \mathbf{x}) = (\mathbf{w}^{12} \mathbf{w}^{01}) \mathbf{x} + (\mathbf{w}^{12} \mathbf{b}^1 + \mathbf{b}^2).
 \end{aligned} \tag{3.6}$$

Then, the output is

$$\begin{aligned}
 y_1^3 &= b_1^3 + \sum_{k=1}^4 w_{k1}^{23} \cdot y_k^2 \\
 &= b_1^3 + \sum_{k=1}^4 w_{k1}^{23} \cdot \left(b_k^2 + \sum_{j=1}^4 w_{jk}^{12} \cdot \left(b_j^1 + \sum_{i=1}^2 w_{ij}^{01} \cdot x_i^0 \right) \right) \\
 &= \mathbf{b}^3 + \mathbf{w}^{23} [(\mathbf{w}^{12} \mathbf{w}^{01}) \mathbf{x} + (\mathbf{w}^{12} \mathbf{b}^1 + \mathbf{b}^2)] \\
 &= (\mathbf{w}^{23} \mathbf{w}^{12} \mathbf{w}^{01}) \mathbf{x} + (\mathbf{w}^{23} \mathbf{w}^{12} \mathbf{b}^1 + \mathbf{w}^{23} \mathbf{b}^2 + \mathbf{b}^3).
 \end{aligned} \tag{3.7}$$

As we can see, all three equations above have the linear form $\mathbf{W}\mathbf{x} + \mathbf{b}$. The reason is that the hidden values are the linear function of the inputs and the outputs are also linear function of the hidden values. Hence, for any values of the weights, the outputs just yield the linearity. However, we can overcome this limitation by applying the *nonlinear activation function* to each hidden node values after the linear transformation of each layer. Some common activation functions will be introduced in the next subsection 3.3. Eq. 3.5 can be generalized

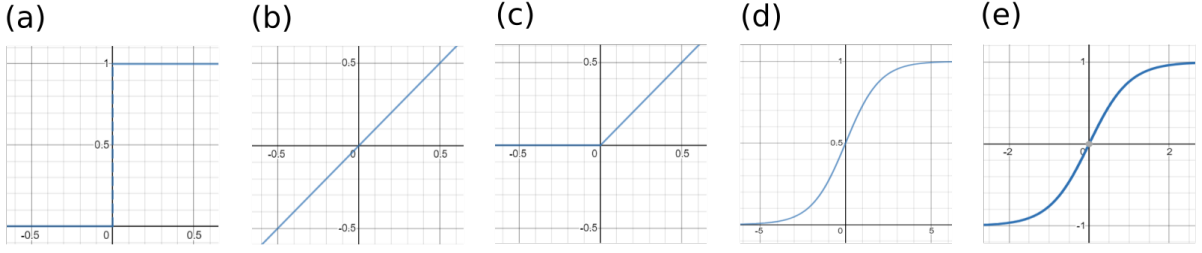


Figure 3.5: Activation function: (a) Binary step function, (b) Linear activation function, (c) ReLU activation function, (d) Sigmoid activation function, (e) Hyperbolic Tangent activation function.

as

$$y_i^j = f_i^j \left(b_i^j + \sum_{k=1}^{N_{j-1}} w_{k,i}^{j-1,j} \cdot y_k^{j-1} \right). \quad (3.8)$$

The output of example NN turns out

$$y_1^3 = f_1^3 \left(b_1^3 + \sum_{k=1}^4 w_{k1}^{23} \cdot f_k^2 \left(b_k^2 + \sum_{j=1}^4 w_{jk}^{12} \cdot f_j^1 \left(b_j^1 + \sum_{i=1}^2 w_{ij}^{01} \cdot x_i^0 \right) \right) \right). \quad (3.9)$$

3.3 Activation function

3.3.1 Binary step function

$$h(z) = \begin{cases} 0 & \text{if } z < \text{threshold} \\ 1 & \text{if } z \geq \text{threshold} \end{cases} \quad (3.10)$$

The Heaviside step function introduced in Section 3.1 is a kind of binary step function. If the input value is above the threshold, the node is activated. The drawback of this function is that it cannot provide multiple value of outputs, which means that it is not available for the multinomial classification problems.

3.3.2 Linear activation function

$$f(z) = z \quad (3.11)$$

The output of this function is not confined between any range and proportional to the input. However, the derivative of the linear function is a constant, so it is not possible to use backpropagation to train the NN since there is no relation between the gradient and input.

Furthermore, as discussed above, without activation function, no matter how many layers in the NN, the output layer is always the linear function of the input layer. Thus, the NN with the linear activation function is just a linear regression model.

3.3.3 Non-linear activation functions

ReLU activation function

$$ReLU(z) = \max(z, 0) \quad (3.12)$$

ReLU is the abbreviation of the *rectified linear unit* which is the most common activation function used in the NN. The ReLU returns output z if z is positive and 0 otherwise. It is a non-linear function and have the derivative allowing for backpropagation. Furthermore, because the ReLU includes simple mathematical operations so it can converge quickly. Nevertheless, there is a restriction that the gradient can be zero for negative z , this is so-called "Dying ReLU". If the nodes turn zero, there is no meaning for the activation in the next layer and the weights are not updated with the gradient descent.

Sigmoid activation function

$$y = f(x) = \frac{1}{1 + e^{-x}} \quad (3.13)$$

It is similar to step function but looks more smooth and it can prevent suddenly "jump" in the output values. The boundary of the outputs is from 0 to 1 because the outputs are normalized. The small changes in x lead to significant change in y , and this can be clearly observed in Fig. 3.5d. For the x values from -2 to 2, the corresponding Y values are very steep. Therefore, the sigmoid function tends to bring y values to the edge of the curve, which means 1 or 0. So it allows clear predictions or distinctions. On the other side, due to the barely change of y values when x values go toward either end of the sigmoid function, the gradients at these regions are small. For that reason, the network denies to learn further or takes a long time to achieve accurate results.

Tanh/Hyperbolic Tangent activation function

$$y = f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = 2\text{sigmoid}(2x) - 1 \quad (3.14)$$

The tangent function is a scaled sigmoid function, so both are similar and can be derived from each other. The differences are that its derivatives are steeper, and it is zero centered (the boundary is from -1 to 1). Both sigmoid and tangent functions are widely used, but the ReLU function is preferred because tanh is computationally expensive.

3.4 Backpropagation

Backward propagation of error or for short "backpropagation" is a popular algorithm for training the FFNN under supervised learning using gradient descent. In this method, the gradient of the error function with respect to the weights of NN is calculated for the final layer first and then for the first layer. That means the process is "backward" through the network. The main target of training progress is to find the best weights and biases for hidden layer nodes [13].

For clearly, the notations of FFNN are rewritten as following:

- w_{ij}^{kl} : weight connecting the node i in layer k with the node j in the next layer $l = k + 1$
- b_i^k : bias for the node i in layer k
- a_i^k : bias plus product of summation for node i in the layer k
- o_i^k : output for the node i in layer k
- σ : activation function for the hidden layer nodes
- σ_{out} : activation function for the output layer nodes.

The bias b_i^k can be analyzed as the multiple of the weight $w_{0i}^{k-1,k}$ with a fixed output $o_0^{k-1} = 1$ for the node 0 in the layer $k - 1$. Hence, $b_i^k = w_{0i}^{k-1,k} \cdot 1 = w_{0i}^{k-1,k}$. The output for the node i in layer k before passing to the activation function is:

$$a_i^k = b_i^k + \sum_{j=1}^{N_{k-1}} w_{ji}^{k-1,k} \cdot o_j^{k-1} = \sum_{j=0}^{N_{k-1}} w_{ji}^{k-1,k} \cdot o_j^{k-1}. \quad (3.15)$$

The dataset of NN includes m input-output pairs $X = \{(\vec{x}_1, \vec{y}_1), \dots, (\vec{x}_m, \vec{y}_m)\}$, where \vec{x}_i is the input and \vec{y}_i is the output label. Considering the NN which has only one output, the output label is not a vector, so the input-output pairs will have the form (\vec{x}_i, y_i) . An *error function* $E(X, \theta)$ describes the error between predicted output and the output label for a set of input-output pairs X and a value of parameter θ . In general, training NN with backpropagation means the calculation of the gradient of the error function $E(X, \theta)$ with respect to the weights and biases. The next step is updating the parameters at each iteration of gradient descent with the learning rate α ,

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta}. \quad (3.16)$$

The error function over m training examples minimized by backpropagation is *mean square error* (MSE):

$$E(X, \theta) = MSE = \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2, \quad (3.17)$$

where \hat{y}_i is the predicted output of NN, and the derivative of it can be computed as the sum over individual error terms for each individual input-output pair

$$\frac{\partial E(X, \theta)}{\partial w_{ij}^{k-1,k}} = \frac{1}{m} \sum_{n=1}^m \frac{\partial}{\partial w_{ij}^{k-1,k}} \left(\frac{1}{2} (\hat{y}_n - y_n)^2 \right) = \frac{1}{m} \sum_{n=1}^m \frac{\partial E(X, \theta)_n}{\partial w_{ij}^{k-1,k}}. \quad (3.18)$$

Then, the error function for each input-output pair is

$$E = \frac{1}{2} (\hat{y} - y)^2. \quad (3.19)$$

Applying chain rule to estimate the partial derivative of the error function

$$\frac{\partial E}{\partial w_{ij}^{k-1,k}} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^{k-1,k}} = \delta_j^k o_i^{k-1}, \quad (3.20)$$

in which $\delta_j^k = \frac{\partial E}{\partial a_j^k}$ and $\frac{\partial a_j^k}{\partial w_{ij}^{k-1,k}} = \frac{\partial}{\partial w_{ij}^{k-1,k}} \left(\sum_{l=0}^{N_{k-1}} w_{lj}^{k-1,k} \cdot o_l^{k-1} \right) = o_i^{k-1}$. The product of error δ_j^k and output o_i^{k-1} implies the weight $w_{lj}^{k-1,k}$ connecting the input node j in layer k and the output node i in layer $k-1$.

Considering that there is only one output with activation function in the final layer, we can evaluate the error function as

$$E = \frac{1}{2} (\hat{y} - y)^2 = \frac{1}{2} (\sigma_{out}(a_0^{out}) - y)^2. \quad (3.21)$$

The error for the node in the final layer is

$$\begin{aligned} \delta_0^{out} &= \frac{\partial E}{\partial a_0^{out}} = \frac{\partial}{\partial a_0^{out}} \left[\frac{1}{2} (\sigma_{out}(a_0^{out}) - y)^2 \right] \\ &= (\sigma_{out}(a_0^{out}) - y) \sigma'_{out}(a_0^{out}) = (\hat{y} - y) \sigma'_{out}(a_0^{out}). \end{aligned} \quad (3.22)$$

From the Eqs. 3.20 and 3.22, the partial derivative of the error function with respect to a weight in the final layer can be expressed as

$$\frac{\partial E}{\partial w_{i0}^{out-1,out}} = \delta_0^{out} o_i^{out-1} = (\hat{y} - y) \sigma'_{out}(a_0^{out}) o_i^{out-1}. \quad (3.23)$$

Considering the error δ_j^k in the layers $1 \leq k \leq out$,

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{N_{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} = \sum_{l=1}^{N_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}. \quad (3.24)$$

Since the bias o_0^k relating to $w_{0j}^{k,k+1}$ is fixed, its value is independent of the outputs of the previous layer. So l does not have value 0. With the appearance of activation function,

$$a_l^{k+1} = \sum_{j=1}^{N_k} w_{jl}^{k+1} \sigma(a_j^k). \quad (3.25)$$

Thus,

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} \sigma'(a_j^k). \quad (3.26)$$

Substituting Eq. 3.26 into Eq. 3.24, the error term in the hidden layers is

$$\delta_j^k = \sum_{l=1}^{N_{k+1}} \delta_l^{k+1} w_{jl}^{k+1} \sigma'(a_j^k) = \sigma'(a_j^k) \sum_{l=1}^{N_{k+1}} w_{jl}^{k+1} \delta_l^{k+1} \quad (3.27)$$

Hence, the partial derivative of the error function with respect to a weight in the hidden layers $1 \leq k \leq out$ has the form

$$\frac{\partial E}{\partial w_{ij}^{k-1,k}} = \delta_j^k o_i^{k-1} = \sigma'(a_j^k) o_i^{k-1} \sum_{l=1}^{N_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}. \quad (3.28)$$

In summary, there are five equations required in the backpropagation algorithm:

- The partial derivatives: $\frac{\partial E}{\partial w_{ij}^{k-1,k}} = \delta_j^k o_i^{k-1}$
- The error in the final layer (for NN having only one output): $\delta_1^{out} = \sigma'_{out}(a_1^{out}) (\hat{y} - y)$
- The error in the hidden layers: $\delta_j^k = \sigma'(a_j^k) \sum_{l=1}^{N_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}$
- The partial derivatives for each input-output pair: $\frac{\partial E(X, \theta)}{\partial w_{ij}^{k-1,k}} = \frac{1}{m} \sum_{n=1}^m \frac{\partial}{\partial w_{ij}^{k-1,k}} \left(\frac{1}{2} (\hat{y}_n - y_n)^2 \right) = \frac{1}{m} \sum_{n=1}^m \frac{\partial E(X, \theta)_n}{\partial w_{ij}^{k-1,k}}$
- The equation for updating the weights: $\Delta w_{ij}^{k-1,k} = -\alpha \frac{\partial E(X, \theta)}{\partial w_{ij}^{k-1,k}}$

The process of backpropagation can be concluded as the following steps,

1. Forward pass: for initial random parameters and input nodes, calculate and store the value of nodes in the hidden layers until getting the predicted output value \hat{y}_i .
2. Backward pass:
 - Estimating the error term δ_1^{out} for the output layer
 - Backpropagating the error terms δ_j^k for the hidden layers
 - Evaluating the partial derivatives of the individual error with respect to the weights
 - Combining the individual gradients to obtain the total gradient $\frac{\partial E(X, \theta)}{\partial w_{ij}^{k-1, k}}$ for the set of input-output pairs.
3. Updating weights: using the total gradient and the learning rate to update the next weights.

Chapter 4

Constructing Long-range Neural Network Potentials

4.1 Conventional NNPs

The conventional NNPs use feed-forward NNs to define the relation between energy and atomic configurations. The input layer contains a vector of input coordinates $\mathbf{R} = \{R_i\}$, and the energy is obtained in the node of the output layer as in Fig. 4.1. In principle, optimizing the weights in the conventional NNPs are similar to feed-forward NNs. The initial choices of the weight parameters can be the random numbers and the number of weights need to be optimized is

$$N_w = \sum_{k=1}^{N_{HL}+1} (N_{k-1} \cdot N_k + N_k) \quad (4.1)$$

where N_{HL} is the number of hidden layers and N_k is the number of node in the layer k .

The conventional NNPs soon reveals its drawbacks. First, it performs well for a small system, but if the system contains too much input node then many weights need to be defined. Therefore, the computational cost is expensive, and it is inefficient to obtain energy. Another problem is for exchanged atoms, for example, a water monomer with the exchanging of two hydrogen atoms. the potential energies in both cases need to be the same since the structure has not changed. However, exchanging atomic positions accidentally changes the order of the input coordinates, which yields different energies for the same atomic configurations. A similar problem occurs with translating and rotating system, distinct energies are given due to the change in input coordinates. Finally, if we add or remove an atom from the system, the weight parameters for that atom are not available or ill-defined and the conventional NNPs

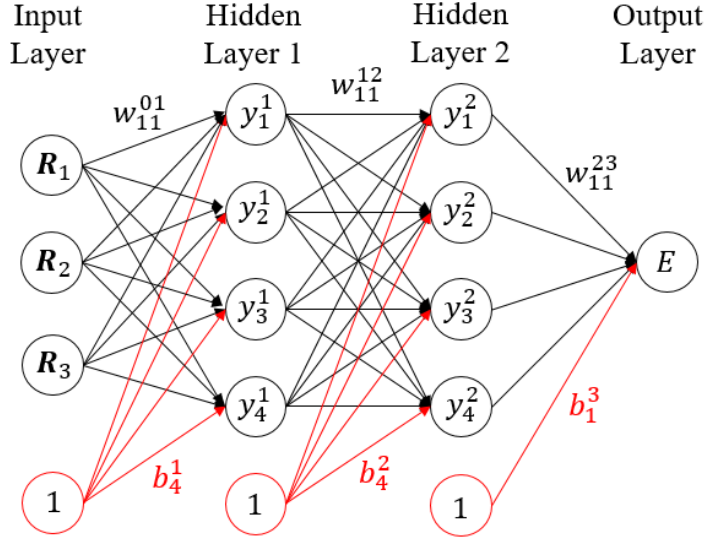


Figure 4.1: Conventional NNPs.

need to be trained again [4].

4.2 High-dimensional NNPs

In 2007, Jörg Behler and Michele Parrinello introduced the high-dimensional NNPs which contribute the short-range energy as a function of all atomic positions. They propose these positions will be transformed into a set of symmetry functions that can describe the local environment of an atom i . The total energy is the summation of all atomic energies.

The structure in high dimensions is shown in Fig. 4.2. The $\{R_i^\alpha\}$ represents the Cartesian coordinates α of the atom i . Then, these coordinates will be converted to a set of symmetry function values $\{G_i^\mu\}$ for each atom. These symmetry function values which play an important role to describe the local environment of each atom are the inputs for the subnet S_i . The subnet S_i is the conventional NNPs, and the subnets are the same for all atoms of the same element. For each element, there is a distinct subnet architecture. After optimizing the weights, the subnet S_i gives atomic energies E_i . Summing these energies we can obtain the total energy of the system,

$$E = \sum_i E_i. \quad (4.2)$$

This kind of NN can overcome the limitations of conventional NNPs. The properties of symmetry functions help the total energy be invariant with permutation of atoms of the same element and translation or rotation of the system. Moreover, the size of the system is easy to

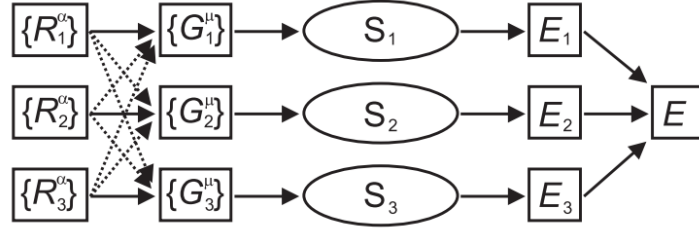


Figure 4.2: Structure of high-dimensional NNPs for system containing three atoms.

vary since each atom has its own scheme.

Since the energy of the system crucially depends on the atomic configurations, a suitable set of coordinates need to be selected. The Cartesian coordinates are not possible, because it is not invariant with respect to translation and rotations of the system. An alternative solution is transforming all atomic positions into many body functions inside the cutoff spheres, these transformed coordinates called "symmetry functions". There are some properties that symmetry functions must have, first, the vector of symmetry function values must be unaffected by the permutation of atoms of the same element and also the rotation and translation of the system. The symmetry functions are able to identify the different atomic environment and give similar coordinates for identical atomic configurations, that means it provides a unique description of the atomic environment. Finally, the number of symmetry functions independent of the number of atom inside the cutoff spheres [14].

The symmetry functions are formulated from cutoff functions multiplied by radial or angular functions. The cutoff function neglects the interaction which yields the interatomic distance R_{ij} larger than the cutoff radius R_c . It has the form

$$f_c(R_{ij}) = \begin{cases} 0.5 \left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1 \right] & \text{for } R_{ij} \leq R_c \\ 0 & \text{for } R_{ij} > R_c \end{cases}, \quad (4.3)$$

which is the monotonically decreasing part of a cosine function. If R_{ij} increases, the function will be decreased, this reflects the strength of the interactions between the atoms vary with the distance.

The simplest radial symmetry function is the sum of the cutoff functions for all neighboring atoms j inside the cutoff sphere,

$$G_i^1 = \sum_j f_c(R_{ij}). \quad (4.4)$$

A set of these functions can be constructed by using different cutoff radii which yield different

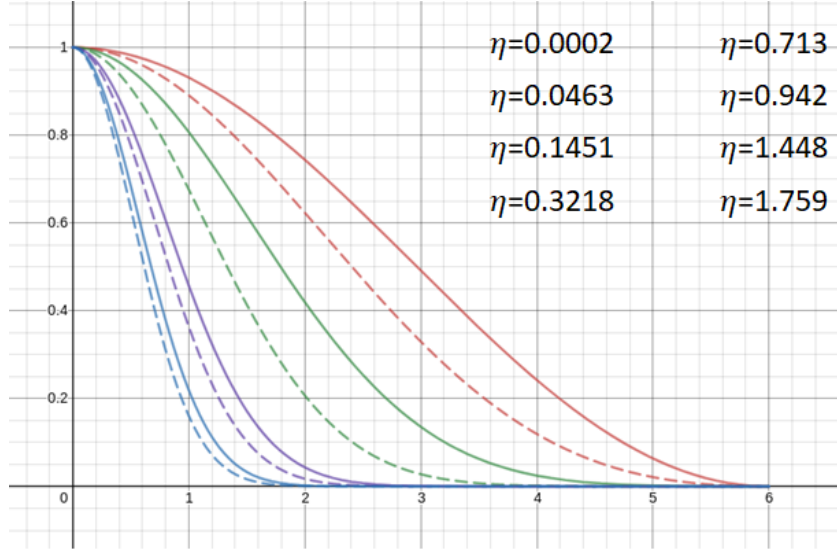


Figure 4.3: Example of an initial radial symmetry function set G^2 with cutoff radius $R_c = 6\text{\AA}$ and $R_s = 0$.

spatial extensions. Another choice is a sum of Gaussians multiplied by cutoff functions,

$$G_i^2 = \sum_j e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}), \quad (4.5)$$

where the width of Gaussian is controlled by the parameter η and the Gaussians can be shifted to a certain radial distance by the parameter R_s . For small values of η and $R_s = 0$ function G^2 reduces to function G^1 . An example of an initial radial symmetry function type G^2 is presented in Fig. 4.3. Finally, the radial function G^3 represents damped cosine functions,

$$G_i^3 = \sum_j \cos(\kappa R_{ij}) \cdot f_c(R_{ij}), \quad (4.6)$$

where the period length adjusted by parameter κ . Since there are positive and negative function values, the neighboring atoms at different distances can cancel each other's contributions to the summation. Thus, this function needs to use with another function.

There are two types for angular symmetry function,

$$G_i^4 = 2^{1-\zeta} \sum_{j,k \neq i}^{all} (1 + \lambda \cos \theta_{ijk})^\zeta \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}), \quad (4.7)$$

and

$$G_i^5 = 2^{1-\zeta} \sum_{j,k \neq i}^{all} (1 + \lambda \cos \theta_{ijk})^\zeta \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}). \quad (4.8)$$

The angle $\theta_{ijk} = \arccos\left(\frac{\mathbf{R}_{ij} \cdot \mathbf{R}_{ik}}{R_{ij} \cdot R_{ik}}\right)$ is centered at the central atom i and is enclosed by the two interatomic distances R_{ij} and R_{ik} . They are different in the radial parts, and have the same angular part. The parameter $\lambda = \pm 1$ shifts the maxima of the cosine function to $\theta_{ijk} = 0^\circ$ with $\lambda = 1$ or $\theta_{ijk} = 180^\circ$ with $\lambda = -1$. The distribution of angle can be investigated by using different values ζ , while the normalization factor $2^{1-\zeta}$ ensures that the range of values is independent of the choice of exponent. The value of function G^5 is larger than G^4 since both $f_c(R_{jk})$ and $e^{-\eta R_{jk}^2}$ are smaller than one, so G^5 is more useful for larger atomic separations. In addition, there is no constraint on the distance between atom i and atom j , thus a significantly number of angles is added in the function.

The parameters R_c , η , R_s , ζ , λ , and κ which define the spatial shape of the symmetry functions are fixed during the optimization. The set of symmetry functions should cover the configuration space in an unbiased way. The spatial extension of the function with the smallest effective range should be selected based on the shortest interatomic distances present in the data set. For each element, the width of the Gaussian with the largest η value should correspond to the shortest interatomic distance and vice versa. If the difference between the smallest and largest symmetry function value, is too small, the symmetry function is not contributing to the distinction of different structures. Thus, for each symmetry function, the range of values should be analyzed. It is good for computational cost if the number of symmetry functions is kept as small as possible. However, it must be large enough for the NNP to distinguish distinct atomic structures.

4.3 Long-range neural network potentials

The long-range NNPs is constructed with the short-range part from high-dimensional NNPs and the long-range part is calculated using the Ewald summation. The reason for choosing the Ewald sum is because it provides high accuracy and feasible computational speed when estimating the long-range potential, due to the rapid convergence between long-range contribution in reciprocal space and the short-range contribution in real space.

The total energy E_{total} of the system is the sum of short-range energy and long-range energy. The former is calculated as the combination of atomic energies, and the latter is

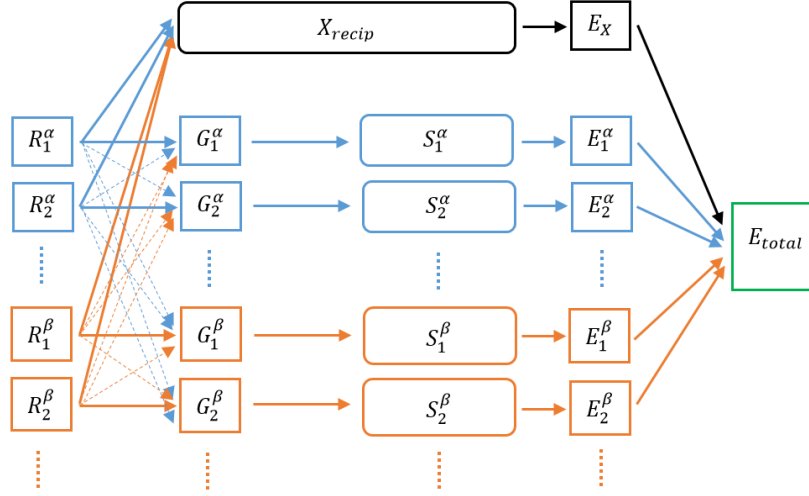


Figure 4.4: Structure of long-range NNPs.

obtained from the reciprocal part in the Ewald sum:

$$E_{total} = E_{SR} + E_{LR} = \sum_i E_i + E_X. \quad (4.9)$$

The structure of this NN is shown in Fig. 4.4. The R_i^α represents the Cartesian coordinates of the atom i^{th} of molecule α , and similar for R_i^β . Then, these coordinates will be converted to a set of symmetry function values for each atom (G_i^α, G_i^β). These symmetry function values which play an important role to describe the local environment of each atom are the inputs for the subnet (S_i^α, S_i^β). In addition, the original Cartesian positions are the input of reciprocal-subnet (X_{recip}). In other words, the short-range atomic potentials depend on the symmetry function values, while the long-range reciprocal part is directly expressed in regard to Cartesian coordinates.

The Ewald sum is the total sum of the real-space term, the reciprocal-space term, the self term and the surface term,

$$\begin{aligned} E_{Ewald} &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N q_i q_j \left(\sum'_{|m| \neq 0} \frac{\text{erfc}(\kappa |\mathbf{r}_{ij} + \mathbf{mL}|)}{|\mathbf{r}_{ij} + \mathbf{mL}|} \right. \\ &\quad \left. + \frac{1}{\pi L^3} \sum_{\mathbf{k} \neq 0} \frac{4\pi^2}{k^2} \exp(-k^2/4\kappa^2) \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \right) \\ &\quad - \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^N q_i^2 + \frac{2\pi}{3L^3} \left| \sum_{i=1}^N q_i \mathbf{r}_i \right|^2 \\ &\equiv E_{real} + E_{recip} - E_{self} + E_{sur}. \end{aligned} \quad (4.10)$$

Pretermittting the energy in real-space and concentrate on the last three terms,

$$\begin{aligned}
E_X &= E_{recip} - E_{self} + E_{sur} \\
&= \frac{1}{2L^3} \sum_{i=1}^N \sum_{j=1}^N q_i q_j \sum_{k \neq 0} \frac{4\pi}{k^2} \exp(-k^2/4\kappa^2) \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \\
&\quad - \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^N q_i^2 + \frac{2\pi}{3L^3} \left| \sum_{i=1}^N q_i \mathbf{r}_i \right|^2 \\
&\equiv E_1 + E_2 + E_3.
\end{aligned} \tag{4.11}$$

We will analyze this potential as the sum of $q_a q_a X_1 + q_a q_b X_2$. Now, a and b indicate two distinct molecules, N_a and N_b are number of atom type a and b in the system, respectively.

$$\begin{aligned}
\text{For } E_1 : \quad & \frac{1}{2L^3} \sum_{i=1}^N \sum_{j=1}^N q_i q_j \sum_{k \neq 0} \frac{4\pi}{k^2} \exp(-k^2/4\kappa^2) \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \\
&= \frac{1}{2L^3} \sum_{k \neq 0} \frac{4\pi}{k^2} \exp(-k^2/4\kappa^2) \sum_{i=1}^N \sum_{j=1}^N q_i q_j \exp(-i\mathbf{k} \cdot \mathbf{r}_{ij}) \\
&= \frac{1}{2L^3} \sum_{k \neq 0} \frac{4\pi}{k^2} \exp(-k^2/4\kappa^2) \left[q_a q_a \sum_{i=1}^{N_a} \sum_{j=1}^{N_a} \exp(i\mathbf{k} \cdot \mathbf{r}_{ij}^a) \right. \\
&\quad + q_b q_b \sum_{i=1}^{N_b} \sum_{j=1}^{N_b} \exp(i\mathbf{k} \cdot \mathbf{r}_{ij}^b) \\
&\quad \left. + 2q_a q_b \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \exp(-i\mathbf{k} \cdot \mathbf{r}_i^a) \exp(i\mathbf{k} \cdot \mathbf{r}_j^b) \right] \\
\text{For } E_2 : \quad & \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^N q_i^2 = q_a q_a \frac{\kappa}{\pi^{1/2}} \sum_{i=1}^{N_a} 1 \\
\text{For } E_3 : \quad & \frac{2\pi}{3L^3} \left| \sum_{i=1}^N q_i \mathbf{r}_i^a \right|^2 = \frac{2\pi}{3L^3} \left| q_a \sum_{i=1}^{N_a} \mathbf{r}_i^a + q_b \sum_{j=1}^{N_b} \mathbf{r}_j^b \right|^2 \\
&= \frac{2\pi}{3L^3} \left[q_a q_a \left| \sum_{i=1}^{N_a} \mathbf{r}_i^a \right|^2 + q_b q_b \left| \sum_{j=1}^{N_b} \mathbf{r}_j^b \right|^2 + 2q_a q_b \left| \sum_{i=1}^{N_a} \mathbf{r}_i^a \sum_{j=1}^{N_b} \mathbf{r}_j^b \right| \right].
\end{aligned}$$

Thus,

$$\begin{aligned}
E_X &= q_a q_a \left[\frac{1}{2L^3} \sum_{i=1}^{N_a} \sum_{j=1}^{N_a} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(i\mathbf{k} \cdot \mathbf{r}_{ij}^a)} - \sum_{i=1}^{N_a} \frac{\kappa}{\pi^{1/2}} + \frac{2\pi}{3L^3} \left| \sum_{i=1}^{N_a} \mathbf{r}_i^a \right|^2 \right] \\
&\quad + q_a q_b \left[\frac{1}{L^3} \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(-i\mathbf{k} \cdot \mathbf{r}_i^a)} e^{(i\mathbf{k} \cdot \mathbf{r}_j^b)} + \frac{4\pi}{3L^3} \left| \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \mathbf{r}_i^a \mathbf{r}_j^b \right|^2 \right].
\end{aligned} \tag{4.12}$$

For example, applying Eq. () for the NaCl ionic system,

$$\begin{aligned}
E_X &= q_{Na}q_{Na} \left[\frac{1}{2L^3} \sum_{i=1}^{N_{Na}} \sum_{j=1}^{N_{Na}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(i\mathbf{k} \cdot \mathbf{r}_{ij}^{Na})} - \sum_{i=1}^{N_{Na}} \frac{\kappa}{\pi^{1/2}} + \frac{2\pi}{3L^3} \left| \sum_{i=1}^{N_{Na}} \mathbf{r}_i^{Na} \right|^2 \right] \\
&+ q_{Cl}q_{Cl} \left[\frac{1}{2L^3} \sum_{i=1}^{N_{Cl}} \sum_{j=1}^{N_{Cl}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(i\mathbf{k} \cdot \mathbf{r}_{ij}^{Cl})} - \sum_{i=1}^{N_{Cl}} \frac{\kappa}{\pi^{1/2}} + \frac{2\pi}{3L^3} \left| \sum_{i=1}^{N_{Cl}} \mathbf{r}_i^{Cl} \right|^2 \right] \\
&+ q_{Na}q_{Cl} \left[\frac{1}{L^3} \sum_{i=1}^{N_{Na}} \sum_{j=1}^{N_{Cl}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(-i\mathbf{k} \cdot \mathbf{r}_i^{Na})} e^{(i\mathbf{k} \cdot \mathbf{r}_j^{Cl})} + \frac{4\pi}{3L^3} \left| \sum_{i=1}^{N_{Na}} \sum_{j=1}^{N_{Cl}} \mathbf{r}_i^{Na} \mathbf{r}_j^{Cl} \right|^2 \right] \\
&\equiv q_{Na}q_{Na}X_1 + q_{Cl}q_{Cl}X_2 + q_{Na}q_{Cl}X_3 \\
&\equiv \theta_1X_1 + \theta_2X_2 + \theta_3X_3.
\end{aligned} \tag{4.13}$$

The potential E_X becomes the linear combination of parameters θ_i multiplying with feature values X_i . We treat the feature values X_i , which directly depend on the atomic positions, as the nodes and parameters θ_i , which are trained with the whole model, as the weights. Finally, computing E_X together with short-range atomic energy from Behler-Parrinello type we can obtain the complete model as in Fig. 4.4.

Chapter 5

Training Neural Network Potentials for NaCl

5.1 Preparing for the model

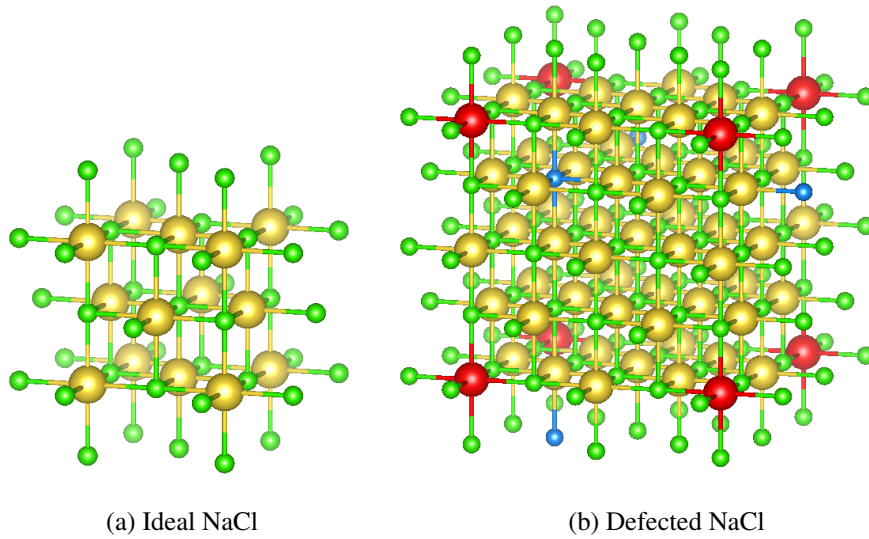


Figure 5.1: Ideal NaCl and defected NaCl.

The initial data sets were obtained from simulating cubic crystal NaCl with a lattice constant of 5.64 \AA at different temperatures. For both ideal and defected systems, in total 6000 structures and corresponding energies were given, 5400 of which were used for optimizing the NN, and 600 as a validation set. The validation set was used to investigate the predictive capability of the model and not included in the training set. The Fig. [5.1a](#) shows the supercell $1 \times 1 \times 1$ of

Table 5.1: Symmetry function parameters for G^2 and G^5 .

G^2		G^5											
η		η	ζ	λ	η	ζ	λ	η	ζ	λ	η	ζ	λ
0.0002	0.713	0.0001	1.0	-1.0	0.2825	1.0	-1.0	0.8193	1.0	-1.0	1.3115	1.0	-1.0
0.0463	0.942	0.0001	1.0	1.0	0.2825	1.0	1.0	0.8193	1.0	1.0	1.3115	1.0	1.0
0.1451	1.448	0.0001	2.0	-1.0	0.2825	2.0	-1.0	0.8193	2.0	-1.0	1.3115	2.0	-1.0
0.3218	1.759	0.0001	2.0	1.0	0.2825	2.0	1.0	0.8193	2.0	1.0	1.3115	2.0	1.0

ideal NaCl system and the Fig. 5.1b show the supercell 2x2x2 of defected NaCl system, in which the red Na atoms and the blue Cl atoms are missing.

Following Behler's suggestions [14], we decided to use symmetry function type $G^2 = \sum_j e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij})$ since it does not need to use with another radial functions as $G^3 = \sum_j \cos(\kappa R_{ij}) \cdot f_c(R_{ij})$ and it can reduce to function $G^1 = \sum_j f_c(R_{ij})$ for small η and $R_s = 0$. An equidistant set of radial functions as shown in Fig. 4.3 were used in which the smallest value of η correspond to the largest interatomic distance. Because there is no restriction on the distance between atom j and k , the angular symmetry function type $G^5 = 2^{1-\zeta} \sum_{j,k \neq i}^{all} (1 + \lambda \cos \theta_{ijk})^\zeta \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik})$ can give larger range of angle than $G^4 = 2^{1-\zeta} \sum_{j,k \neq i}^{all} (1 + \lambda \cos \theta_{ijk})^\zeta \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk})$. Thus, we prefer symmetry function G^5 to G^4 . A set of 8 radial functions and 16 angular functions are used with radius cutoff of 6 Å, and the parameters are listed in Table 5.1. The value of symmetry functions can have different range, in which the biggest value can have significant effect on the model, while the smallest value does not contribute to the model. For this reason, the symmetry functions are required to be rescaled to make their range in a particular interval. The symmetry functions in this thesis are scaled to be in the range of $[-1, 1]$ by applying

$$G_i^{scaled} = \frac{2(G_i - G_{i,min})}{G_{i,max} - G_{i,min}} - 1, \quad (5.1)$$

where $G_{i,min}$ and $G_{i,max}$ are the smallest and largest values of symmetry functions in the data set.

5.2 Optimizer and learning rate

The backpropagation algorithm mentioned in section 3.4 computes the gradient of the loss function with respect to each parameter in order to update the weight parameters. Sometimes the gradients become smaller as the processes pass to the lower layers, this leads to tiny changes on the weights and the training will not convert to an expected solution. This is called the "vanishing gradients" problem. Reversely, when the gradients increase, the weights get large update and we call this "exploding gradients". There are many better and faster solutions that will be introduced in this section such as Momentum optimization, AdaGrad, RMSProp, Adam optimizations.

First, the most basic form for updating the weight vector θ is $\theta_{new} \leftarrow \theta_{old} - \eta \nabla_{\theta} J(\theta)$ where η is the learning rate and this form has been seen in the backpropagation method. The next optimizer is *Momentum*. Suppose we have a ball rolling down on a smooth surface, it will be slow at first, but it then picks up momentum and gains the final velocity. It is the basic idea of Momentum optimizer. Large slopes accelerate the process of convergence and prevents fluctuation at the end of the process. The momentum vector \mathbf{m} is changed at each iteration, $\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta)$, and this vector is added into the weights for updating $\theta \leftarrow \theta + \mathbf{m}$. β is called the momentum hyperparameter, which must be set between 0 (high friction) and 1 (no friction). A typical momentum value is 0.9 [15].

A little bit different from Momentum optimizer, which calculates the derivative at the current position for updating momentum and make a jump based on the previous momentum vector, the *Nesterov accelerated gradients* rely on the old momentum vector to calculate the next position and then using the gradient at the new positions to update [15]. The process is:

1. $\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta + \beta \mathbf{m})$
2. $\theta \leftarrow \theta + \mathbf{m}$.

Unlike the previous methods, the learning rate is almost the same for training, *AdaGrad optimizer* considers learning rate as a parameter. The AdaGrad algorithm steps are:

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$
2. $\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

The symbol \otimes and \oslash present the element-wise multiplication and division. The first step is equivalent to compute $s_i \leftarrow s_i + (\partial J(\theta) / \partial \theta_i)^2$, each element s_i of the vector \mathbf{s} accumulates the squares of the partial derivative of the cost function with regards to parameter θ_i . In the second step the gradient vector is reduced by a factor of $\sqrt{s + \epsilon}$ where ϵ is a smoothing term to avoid division by zero, typically set to 10^{-10} . An advantage of AdaGrad is to avoid adjusting the

learning rate manually, usually set it to 0.01 and then the algorithm will automatically adjust. One drawback of AdaGrad is that the learning rate gets scaled down very fast, making training become frozen before reaching the global optimum [16].

RMSProp optimizer appears to fix AdaGrad's problem by accumulating only the gradients from the most recent iterations [16]. It does so by using exponential decay in the first step of the process:

1. $\mathbf{s} \leftarrow \beta \mathbf{s} + (1 - \beta) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$
2. $\theta \leftarrow \theta - \frac{\eta}{\sqrt{\mathbf{s} + \varepsilon}} \nabla_{\theta} J(\theta)$

The decay rate β is typically set to 0.9.

Finally, *Adam optimizer* is the combination of Momentum optimization and RMSProp. It maintains the average of the past slope, like the Momentum, and it keeps the mean square of the past slope, just like RMSProp:

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\theta} J(\theta)$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$
3. $\hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1 - \beta_1^t}$
4. $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \beta_2^t}$
5. $\theta \leftarrow \theta + \eta \frac{\hat{\mathbf{m}}}{\sqrt{\hat{\mathbf{s}} + \varepsilon}}$

In steps 3 and 4, \mathbf{m} and \mathbf{s} will be biased toward 0 at the beginning of training since \mathbf{m} and \mathbf{s} are initialized at 0, so these two steps will help boost \mathbf{m} and \mathbf{s} at the beginning of the learning process. The hyperparameter β_1 , which is often initialized to 0.9, is momentum decay, while the scaling decay hyperparameter β_2 is often initialized to 0.999. As mentioned before, the smoothing term ε is usually initialized to a tiny number such as 10^{-7} . The Adam is adaptive learning rate algorithm like AdaGrad and RMSProp, so we can often use the default value $\eta = 0.001$, making Adam easier to be used than others [17]. In this thesis, we use Adam optimizer for the training process.

The learning rate used in the optimization method also needs to be considered. If it is too high, the training can diverge. If it is a little bit high, it will make progress very quickly at first, but it will end up moving around the optimum. Using an adaptive learning rate, optimization algorithms such as AdaGrad and RMSProp can improve the optimization process, but it still wastes the computation time. If the learning rate is too low, training will converge to the optimum, but it will take very long time. We can find a good learning rate by training the NN

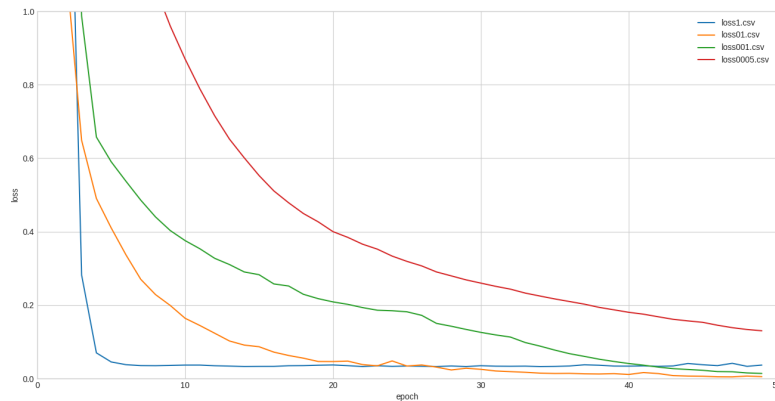


Figure 5.2: Learning curves for various learning rates.

several times during just a few epochs using various learning rates and comparing the learning curves. For the NaCl, the learning curves are shown in the Fig. 5.2, and the orange and green lines seem good. However, with just a few epochs we do not know whether the loss value will converge or diverge after that. Another way is to start with a high learning rate and then reduce it until it stops making fast progress, the strategy to reduce the learning rate during training is called *learning schedules*. The most common schedule is *exponential scheduling*. If we set the learning rate to $\eta(t) = \eta_0 0.1^{t/s}$, the learning rate will drop by a factor of 10 every s steps. The exponential scheduling is easy to adjust and fast converges to the optimal value. Since it is supported by Keras, we use this method and start with a learning rate of 0.01.

5.3 Overfitting problem

After determining the learning algorithm, we need to train the model on some data and sometimes we will confront the "bad data" problem. Deep neural networks with so many parameters have a huge amount of freedom and can fit an enormous variety of complex datasets. However, this flexibility also leads to "overfitting" problem. Overfitting is a common problem in machine learning that occurs when the model performs well on the training data but it fails to fit new data. The possible solution is regularization which can constrain a model to make it simpler and reduce the risk of overfitting [18]. In the next subsection, we will introduce two regularization ways:

- regularizing the weights of the model
 - Ridge Regression

- Lasso Regression
- Elastic Net
- regularizing the iterative learning algorithms
 - Early stopping

5.3.1 Ridge Regression

The weights are constrained by adding a regularization term $\alpha \sum_{i=1}^n \theta_i^2$ to the cost function during training. The full expression of the Ridge Regression cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2. \quad (5.2)$$

The hyperparameter α determines how much you want to regularize the model. If $\alpha = 0$, the Ridge Regression becomes Linear Regression. If α goes to infinity, all the weights almost turn to zero and the result is a straight line going through the mean of data. In this cost function the bias term θ_0 is not considered since the regularization term starts at $i = 0$. Denote that the norm l_k of a vector \mathbf{v} containing n elements is $\|\mathbf{v}\|_k = \left(|v_0|^k + |v_1|^k + \dots + |v_n|^k\right)^{\frac{1}{k}}$. The norm l_0 gives the number of non-zero elements in the vector, and l_∞ gives the maximum absolute value in the vector. If the vector of feature weights (θ_1 to θ_n) is \mathbf{w} , the regularization term is $\frac{1}{2} (\|\mathbf{w}\|_2)^2$, where the norm l_2 of the weight vector is $\|\mathbf{w}\|_2$. The penalty hyperparameter sets the type of regularization term equal to a half of the square of the magnitude of the weight vector [19].

5.3.2 Lasso Regression

Lasso Regression is very similar to Ridge Regression. It also adds a regularization term to the cost function, but it adds penalty equivalent to absolute value of the magnitude of the weight vector. The Lasso Regression cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|. \quad (5.3)$$

The Lasso Regression tends to completely remove the least essential weight features by setting them to zero. It automatically performs feature selection and outputs a sparse model. The Lasso cost function is not differentiable at $\theta_i = 0$ (for $i = 1, 2, \dots, n$), but Gradient Descent

still works if a subgradient vector \mathbf{g} is used instead when $\theta_i = 0$. One possible subgradient vector equation which can be used for Gradient Descent with the Lasso cost function is [20]

$$g(\theta, J) = \nabla_{\theta} MSE(\theta) + \alpha \begin{pmatrix} \text{sign}(\theta_1) \\ \text{sign}(\theta_2) \\ \vdots \\ \text{sign}(\theta_n) \end{pmatrix}, \text{ where } \text{sign}(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}. \quad (5.4)$$

5.3.3 Elastic Net

Elastic Net is a middle ground between Ridge Regression and Lasso Regression. The regularization term is a mix of both Ridge and Lasso's regularization terms, with mix ratio r . The Elastic Net cost function is

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2. \quad (5.5)$$

If $r = 0$, Elastic Net is equivalent to Ridge Regression, and if $r = 1$, it is equivalent to Lasso Regression.

It is supposed to be used a little bit of regularization to avoid plain Linear Regression. Ridge Regression is majorly used to prevent overfitting. Since it includes all the features, it is not very useful in the case of the high number of features. Lasso Regression provides a sparse solution, and it has the computational advantage since the features with zero coefficients can simply be ignored. Lasso Regression can be unreliable when the number of features is greater than the number of training instances or when several features are strongly correlated. In this case, the Elastic Net is an alternative solution [21].

5.3.4 Early Stopping

Early stopping is used to constrain the iterative learning of the model to stop training as soon as the validation error reaches a minimum. the Fig. 5.3 shows an example of a model which is trained using Batch Gradient Descent. The algorithm learns and its prediction error (RMSE) on the training set goes down as the increase of epochs, and so does its prediction error on the validation set. The model will be stopped training as soon as the validation error reaches the minimum [22].

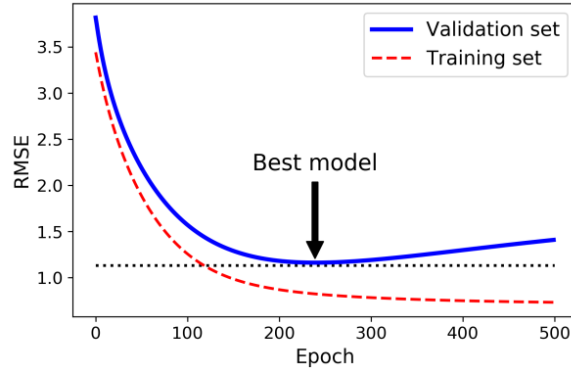


Figure 5.3: Early stopping regularization (Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow. O'Reilly Media, second edition, 2019).

5.4 Constraint in the model weights

Since the sodium chloride with and without vacancy are in a neutral system and the weight parameters in the long-range NNPs represent the charge, we need to constrain the model weights to satisfy the neutrality condition. The long-range part for NaCl is

$$\begin{aligned} E_X &= \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 \\ &= q_{Na} q_{Na} X_1 + q_{Cl} q_{Cl} X_2 + q_{Na} q_{Cl} X_3. \end{aligned}$$

In neutral system, $|q_{Na}| = |q_{Cl}|$ and $q_{Na} + q_{Cl} = 0$, so

$$\begin{aligned} (q_{Na} + q_{Cl})^2 &= q_{Na} q_{Na} + 2q_{Na} q_{Cl} + q_{Cl} q_{Cl} = 0 \\ \theta_1 + \theta_2 + 2\theta_3 &= 0 \\ \theta_1 + \theta_2 &= -2\theta_3 \end{aligned}$$

Because $q_{Na} q_{Na} = q_{Cl} q_{Cl}$, we have $\theta_1 = \theta_2$ and thus

$$\begin{aligned} E_X &= \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 \\ &= \theta_1 (X_1 + X_2) + \theta_3 X_3 \\ &= \theta_1 (X_1 + X_2 - X_3). \end{aligned}$$

In the beginning, the NNs start with random weights, and after training the weight parameters have arbitrary values. Hence, to make the model physically meaningful we analyze the long-range part as shown in Fig. 5.4. The number of weight parameters is reduced from 3 to 1. The weight θ_1 need to be constrained to be positive since it is the square of the charge q_{Na} and its value should be close to 1 as the oxidation number of Na is $+1e$.

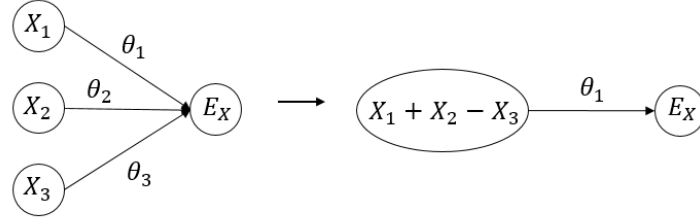


Figure 5.4: Constrain on the long-range part.

5.5 Results and discussion

In this thesis, we improved the high-dimensional NNPs by adding the long-range term from the Ewald sum to obtain the total energy of the system. The reliability of the model is tested in NaCl crystal with an ideal crystalline system and a defected system missing one atom Na and one atom Cl. To find the atomic structures and relevant energies we simulate cubic crystal NaCl at different temperatures.

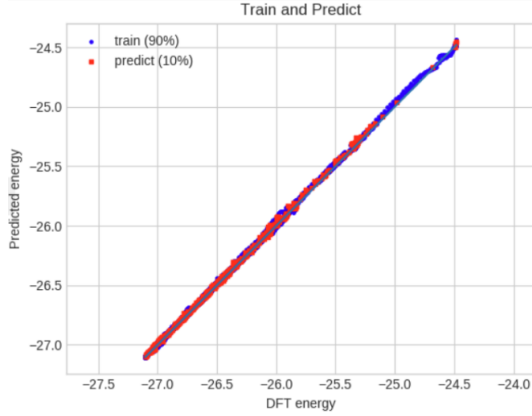
The Behler-Parrinello method was implemented to deal with short-range energy calculations. The symmetry functions with various parameters were used as the input of NNPs. The long-range part of Ewald sum was analyzed as the linear combination of parameters q_i multiplied by the feature values X_i ,

$$\begin{aligned}
 E_X &= q_{\text{Na}}q_{\text{Na}} \left[\frac{1}{2L^3} \sum_{i=1}^{N_{\text{Na}}} \sum_{j=1}^{N_{\text{Na}}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(i\mathbf{k} \cdot \mathbf{r}_{ij}^{\text{Na}})} - \sum_{i=1}^{N_{\text{Na}}} \frac{\kappa}{\pi^{1/2}} + \frac{2\pi}{3L^3} \left| \sum_{i=1}^{N_{\text{Na}}} \mathbf{r}_i^{\text{Na}} \right|^2 \right] \\
 &+ q_{\text{Cl}}q_{\text{Cl}} \left[\frac{1}{2L^3} \sum_{i=1}^{N_{\text{Cl}}} \sum_{j=1}^{N_{\text{Cl}}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(i\mathbf{k} \cdot \mathbf{r}_{ij}^{\text{Cl}})} - \sum_{i=1}^{N_{\text{Cl}}} \frac{\kappa}{\pi^{1/2}} + \frac{2\pi}{3L^3} \left| \sum_{i=1}^{N_{\text{Cl}}} \mathbf{r}_i^{\text{Cl}} \right|^2 \right] \\
 &+ q_{\text{Na}}q_{\text{Cl}} \left[\frac{1}{L^3} \sum_{i=1}^{N_{\text{Na}}} \sum_{j=1}^{N_{\text{Cl}}} \sum_{k \neq 0} \frac{4\pi}{k^2} e^{(-k^2/4\kappa^2)} e^{(-i\mathbf{k} \cdot \mathbf{r}_i^{\text{Na}})} e^{(i\mathbf{k} \cdot \mathbf{r}_j^{\text{Cl}})} + \frac{4\pi}{3L^3} \left| \sum_{i=1}^{N_{\text{Na}}} \sum_{j=1}^{N_{\text{Cl}}} \mathbf{r}_i^{\text{Na}} \mathbf{r}_j^{\text{Cl}} \right|^2 \right] \\
 &\equiv q_{\text{Na}}q_{\text{Na}}X_1 + q_{\text{Cl}}q_{\text{Cl}}X_2 + q_{\text{Na}}q_{\text{Cl}}X_3 \\
 &\equiv \theta_1X_1 + \theta_2X_2 + \theta_3X_3.
 \end{aligned}$$

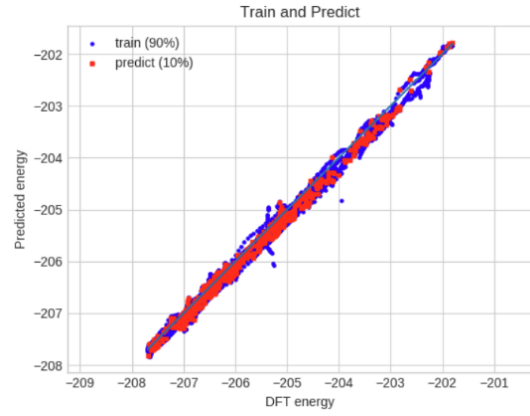
Unlike the short-range part which is transformed to a set of symmetry functions, the long-range term is computed directly from the atomic positions. The feature values X_i are treated as the nodes and the parameters θ_i , which are trained with the whole model, as the weights. To construct the model NNPs, we used the hyperparameter following Table 5.2.

Table 5.2: The hyperparameters of the model NNPs. The learning rate was optimal using exponential scheduling method.

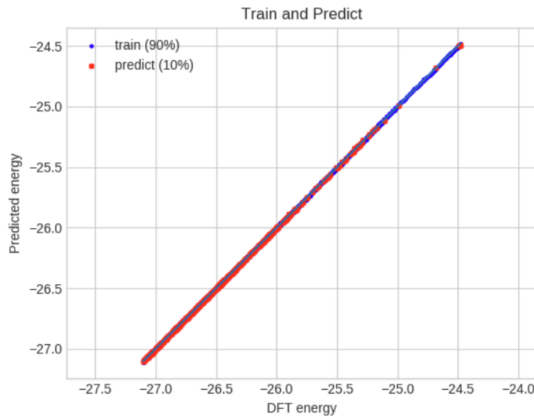
	Number of hidden Layers	Number of nodes per hidden layer	Optimizer	Learning rate	Epochs
With vacancy	2	80	Adam	0.01	10000
Without vacancy	2	80	Adam	0.001	12000



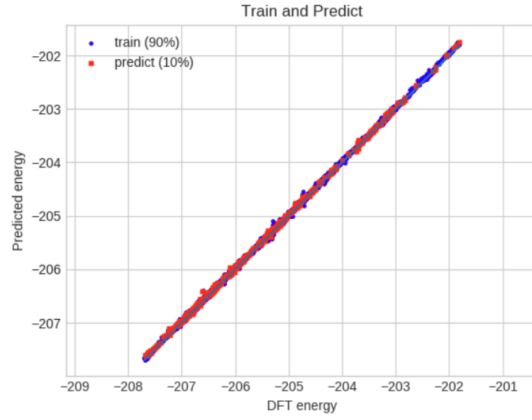
(a) High-dimensional NNPs without vacancy.



(b) High-dimensional NNPs with vacancy.



(c) Long-range NNPs without vacancy.



(d) Long-range NNPs with vacancy.

Figure 5.5: Comparison of DFT energy and predicted energy with the high-dimensional NNPs and long-range NNPs applied to ideal and defected sodium chloride systems.

The high-dimensional NNPs and long-range NNPs are performed as shown in Fig. 5.5. The Behler-Parrinello model gives the RMSE of 19.8 meV for the ideal NaCl system, while the RMSE in long-range NNPs is 4.9 meV. For the defected NaCl system, the NNPs with long-range part is more reliable than the short-range NNPs: the RMSE of the former is 41.2 meV and that of the latter is 89.1 meV. Moreover, the long-range NNPs also meet the physical meaning since we applied the constraints on the model. The number of weight parameters

Table 5.3: Long-range fitting parameters and RMSE for the ideal NaCl and the defected NaCl.

	$\theta_1 (e^2)$	RMSE (meV/atom)
Ideal NaCl	0.9028	4.9
Defected NaCl	0.8436	41.2

in the long-range part is reduced to 1 as the long-range potential equals to $\theta_1 (X_1 + X_2 - X_3)$, after testing θ_1 is positive and its value close to 1, as shown in Table 5.3.

The NNPs computation is faster than the corresponding *ab initio* energy calculation that takes a few days to finish. Table 5.4 represents the difference in computational cost between high-dimensional NNPs and long-range NNPs. Because the number of epochs (the number times that the learning algorithm will work through the entire training dataset) tested for the defected system is larger than that for the ideal system, the computational time in divacancy case is longer than ideal case. The long-range NNPs model takes more time than the high-dimensional NNPs model since the long-range NNPs model needs time to compute the value of X_i from Ewald summation, however this difference is not significant.

Table 5.4: Comparison of the computational times (unit: second) of the high-dimensional NNP and the long-range NNP for the ideal NaCl and defected NaCl.

	High-dimensional NNPs	Long-range NNPs
Ideal NaCl	1851	2136
Defected NaCl	1993	3015

Chapter 6

Conclusion

In conclusion, we constructed the long-range NNPs which are the combination of the short-range energy from the Behler-Parrinello type and the long-range energy from Ewald summation. The short-range energy depends on the local environment of atoms, which is defined by a cutoff radius. There are separate sub-NNs for the atoms of different elements, but the atoms of the same element have identical sub-NNs. The positions of the atoms with respect to the central atom are transferred to a vector of symmetry function as the input of sub-NNs. The long-range energy is added to the total energy by using the reciprocal part of Ewald summation. The product of the charges is supposed to be the weight parameters which are optimized during the training process.

The capability of long-range NNPs has been demonstrated for sodium chloride cubic system. The Behler-Parrinello type can give high accuracy for the ideal cubic NaCl system with a cutoff radius, but it can not give the expected exactness for the defected NaCl system with a divacancy of Na and Cl. Meanwhile, the long-range NNPs not only inherit the success but also overcome the transferability confinement of the high-dimensional NNPs. The long-range NNPs can be numerically very accurate on both cases: For ideal system, this model shows the RMSE of about four times as small as the high-dimensional NNPs and for the defected case, this error is just a half. Moreover, the long-range term of the model also fulfills the physical meaning since the constraint on the weight is applied. The data sets were obtained from *ab initio* simulations at different temperatures, and we developed the Python program with the TensorFlow package to train NNPs and fit the weights to the data sets. The computational time of NNPs is obviously faster than *ab initio* method. Besides, compared to the Behler-Parrinello model, the additional computational cost of the long-range NNPs is insignificant.

References

- [1] F Ercolessi and J. B Adams. Interatomic Potentials from First-Principles Calculations: The Force-Matching Method. *Europhysics Letters (EPL)*, 26(8):583–588, June 1994.
- [2] I. A. Courtney, J. S. Tse, Ou Mao, J. Hafner, and J. R. Dahn. Ab initio calculation of the lithium-tin voltage profile. *Physical Review B*, 58(23):15583–15588, December 1998.
- [3] Frederico V. Prudente, Paulo H. Acioli, and J. J. Soares Neto. The fitting of potential energy surfaces using neural networks: Application to the study of vibrational levels of H₃⁺. *The Journal of Chemical Physics*, 109(20):8801–8808, November 1998.
- [4] Jörg Behler and Michele Parrinello. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters*, 98(14):146401, April 2007.
- [5] Norman S. Ham and Klaus Ruedenberg. Mobile Bond Orders in Conjugated Systems. *The Journal of Chemical Physics*, 29(6):1215–1229, December 1958.
- [6] Monika Thol, Gabor Rutkai, Roland Span, Jadran Vrabec, and Rolf Lustig. Equation of State for the Lennard-Jones Truncated and Shifted Model Fluid. *International Journal of Thermophysics*, 36:25–43, November 2015.
- [7] Jiri Kolafa and John W. Perram. Cutoff Errors in the Ewald Summation Formulae for Point Charge Systems. *Molecular Simulation*, 9(5):351–368, January 1992.
- [8] Abdulnour Y. Toukmaji and John A. Board. Ewald summation techniques in perspective: a survey. *Computer Physics Communications*, 95(2):73–92, June 1996.
- [9] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh Ewald: An $N \log(N)$ method for Ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, June 1993.

- [10] Henry David Hecce, Angel Enrique Garcia, and Thomas Darden. The electrostatic surface term: (I) Periodic systems. *The Journal of Chemical Physics*, 126(12):124106, March 2007.
- [11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.
- [12] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
- [13] Pushparaja Murugan. Feed Forward and Backward Run in Deep Convolution Neural Network. *ArXiv*, abs/1711.03278, 2017. arXiv: 1711.03278.
- [14] Jörg Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *The Journal of Chemical Physics*, 134(7):074106, February 2011.
- [15] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. page 14.
- [16] Mahesh Chandra Mukkamala and Matthias Hein. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. page 9.
- [17] S V G Reddy, K Thammi Reddy, and V ValliKumari. Optimization of Deep Learning using various Optimizers, Loss functions and Dropout. 7(4):8, 2018.
- [18] Xue Ying. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 1168:022022, February 2019.
- [19] Ali Muayad and A. N. Irtefaa. Ridge Regression using Artificial Neural Network. *Indian Journal of Science and Technology*, 9(31), August 2016.
- [20] Caihao Cui and Dianhui Wang. High dimensional data regression using Lasso model and neural networks with random weights. *Inf. Sci.*, 372:505–517, 2016.
- [21] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, April 2005.
- [22] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. page 7.