universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„A neural network potential for the water-vapor interface“

verfasst von / submitted by

Oliver Wohlfahrt, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 876
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Physik
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Christoph Dellago

Abstract

Artificial neural networks can be trained on results from complex and computationally very expensive numerical calculations for the quantum mechanical determination of the energy and resultant forces of a collection of atoms in order to approximate these energies and forces. Subsequently, by means of such trained artificial neural networks, it is possible to perform molecular dynamics simulations. In recent years this has been realized for liquid water and ice, among many other examples.

In this work the same method was applied to the liquid-vapor interface of water, whereas the so-called density functional theory (RPBE-D3 functional) was employed to calculate the reference energies and forces. Furthermore, we examined what reliability of results from simulations of such interfaces using artificial neural networks can be expected. For this purpose empirical water models were employed, which are by several orders of magnitude more efficient than quantum mechanical calculations. This efficiency made it possible to compare results from simulations based on an empirical water model, on the one hand, and on artificial neural networks trained on this empirical water model, on the other hand. The results are found to agree well with each other. We thus presume that simulations of interfaces by means of artificial neural networks, especially performed at lower temperatures, can accurately reproduce results of quantum mechanical simulations which are computationally much more expensive.

Contents

1. Introduction

1.1. Applications of artificial neural networks

In many applications one has given a set of input and corresponding output values. However, an exact relation or algorithm how the output can be obtained from a certain input may be missing. An example is the task of image classification, where a list of objects present in an image shall be produced [1]. The pixel values of the image and the list of objects are the input and output, respectively. It is a very hard task to find reliable mathematical models describing such a relation between pixel values and objects. Instead, it can be more convincing to construct a function by a fitting process. This can be achieved by choosing an appropriate model function and fitting this model together with its parameters to the given set of input and associated output data by an optimization process.
In contrast, some well-defined mathematical models or algorithms may be very costly in terms of computing time. In these cases the replacement by a less expensive algorithm can be desirable. Such a replacement can, of course, only be an approximation and it needs to be computationally more efficient than its original counterpart in order to be useful. An example in physics where such a procedure is used concerns potential energy surfaces (PES). The PES is the relationship between the energy of a collection of atoms and its positions [2]. Theoretically, one can solve the Schroedinger equation for a given set of atoms and their positions to calculate single points of the PES. If many of these points were needed, such as during molecular dynamics (MD) simulations [3] or Monte Carlo simulations [4], the computational costs would be by far too high in most cases. Employing density-functional theory (DFT) [5] and other efficient electronic structure methods are thus the only alternatives for this purpose. Even then, so-called ab initio MD simulations are restricted to small systems and short simulation times [6]. Here, it can be desirable to create an approximation function, which allows for faster evaluation of energies and forces without sacrificing too much accuracy. This work is concerned with artificial neural networks (ANN)[1] which are capable of fulfilling such tasks.
NNs have already gotten a lot of attention because of their successful usage in image and pattern recognition [7, 8] as well as speech recognition [9]. In high energy physics, NNs have already been used in 1989 for event filtering [10] or, in mathematical physics, for the approximation of functions and their derivatives [11]. Also

[1] In this work the expression "neural network" (NN) is usually used in place of ANN

quite unexpected applications have been found, like the automated search for new quantum experiments [12].

1.2. Neural network potentials for water

The application we are concerned with is, as mentioned previously, fitting NNs to data from PESs obtained by DFT calculations. The resulting approximation function is referred to as neural network potential (NNP) and can be used in MD simulations[2] to accelerate energy and force calculations at the cost of a certain degree of accuracy. Compared with quantum mechanical calculations, this loss of accuracy shall ideally be almost negligible. There are many possible strategies for the implementation of such a NNP. Probably the most successful and convenient approach is that published by Behler and Parrinello in 2007 [13]. By means of this approach, physical properties of many different materials have already been investigated with DFT accuracy. A comprehensive list of these investigations until the year 2017 can be found in a review of Behler [14].
Our focus is on the work of Morawietz *et al.*, where the significance of van der Waals (vdW) interactions for the thermodynamic anomalies of water has been shown [15]. They have employed the NNP method of Behler and Parrinello [13] and used two different DFT functionals, the Becke-Lee-Yang-Parr (BLYP) [16, 17] and the revised Perdew-Burke-Ernzerhof (RPBE) [18] functional. These density functionals are not capable of correctly describing vdW forces. However, vdW correction schemes [19] can be applied to compensate for this. With the aid of the computational efficiency of NNPs, Morawietz *et al.* have pointed out that including the vdW correction scheme leads to significantly less deviations from experimental results. Particularly, the density maximum and melting temperature of water does crucially depend on vdW forces. Thus, rather weak forces originating from vdW interactions appear to be indispensable for the explanation of the anomalous properties of water. In this thesis, we examine the NNP approach of Behler and Parrinello and apply it to the water-vapor interface. To this purpose, we use the n2p2-package, written by Andreas Singraber[3]. Even though the method of NNPs has been proven to perform well in the case of bulk water, there are potential pitfalls for interfaces. The goal of this thesis is to obtain values for the surface tension, the density profiles and some information about the microscopic structure of the water-vapor interface by performing NNP simulations based on DFT training data. It is yet unclear what accuracy can be expected for the so-obtained ensemble averages of observables. To estimate the reliability of these results from NNP simulations of interfaces, we first test the whole procedure on empirical potentials, since we can easily compare quantities from both,

[2] The resulting simulations are often referred to as NNP simulations.
[3] University of Vienna

simulations employing the empirical potential and simulations utilizing NNPs based on exactly this empirical potential.

1.3. Structure of the thesis

In Chapter 2 we discuss NNs in general. After a short introduction about the origin of artificial neurons some basic mathematical concepts such as backpropagation are covered. Subsequently, this Chapter deals with optimization algorithms for the training of NNs with a focus on the Kalman filter method. Chapter 3 presents the idea of NNPs and symmetry functions (SF) and an attempt to classify them. In Chapter 4 the NNP method of Behler and Parrinello is described in detail. In Chapter 5 the empirical SPC/F water potential as well as a description of DFT are presented. Following this, Chapter 6 includes the results of some fits of NNs to simple low-dimensional functions and to the energy of systems containing only one and two water molecules. After that, potential problems of NNP simulations of interfaces as well as typical observables are discussed in Chapter 7. Chapter 8 presents an attempt of applying the method of NNPs to the liquid-vapor interface of argon, employing the well-known Lennard-Jones (12,6) potential. Finally, Chapters 9 and 10 are dedicated to the presentation and discussions of the NNP simulation results of the water-vapor interface based on the empirical SPC/F potential and on DFT calculations.

Plots of this thesis have been created by *matplotlib* [20] and the package *tikz* [21]. Many of the following computational results have been achieved using the *Vienna scientific cluster* (VSC).

2. Introduction to artificial neural networks

2.1. Origin of the Perceptron

In biology and neuroscience, neurons are cells specialized for sending and receiving chemically mediated electrical signals [22]. Artificial neurons are simplified and idealized models of these biological neurons and were first described by McCulloch and Pitts in 1943 [23]. As their biological counterparts, they receive and accumulate signals from other neurons they are connected with. The sum of all incoming signals is called activation, the value of which is modified by a function to finally produce an output signal. The resulting output is forwarded to another set of neurons. McCulloch and Pitts did not employ a model of artificial neurons in a modern sense but before them nobody had used the mathematical notion of computation to establish a theory of mind and brain [24]. A very important contribution to today's understanding of neurons was made by Hebb [25], who initiated the introduction of weight parameters in neuron models, characterizing the connections between them. On the basis of the work of McCulloch, Pitts and Hebb, Rosenblatt developed the Perceptron. In 1962 he provided a detailed description of the Perceptron in his book, "The Principles of Neurodynamics"[26]. Following Rosenblatt's theory, a Perceptron is composed of sensory units, association units and response units, or S-units, A-units and R-units, respectively. Today, a Perceptron is mostly understood as a single neuron, as depicted in Fig. 2.1. Such a neuron can be defined by the number of incoming connections, the set of weights determining the strength of each connection and the activation function f. Each incoming connection transfers a value x_i from an input neuron i, multiplied by the weight parameter w_i. There is also a bias node[4], which is an input node of constant value. The value the neuron yields, based on the inputs and weight parameters, is

$$f(b \cdot w_0 + \sum_{i=1}^{n} x_i \cdot w_i). \tag{2.1}$$

Just the most basic Perceptron Rosenblatt described, the simple Perceptron [26], can be interpreted as a single neuron with preceding input neurons. It is composed of three layers, a sensory unit layer, an association unit layer and a third layer containing only one response unit. S-units are only connected to A-units and all A-units are connected to the final R-unit. Each connection is characterized by its

[4] In the context of NNs the expressions "nodes" and "neurons" are often used interchangeably.

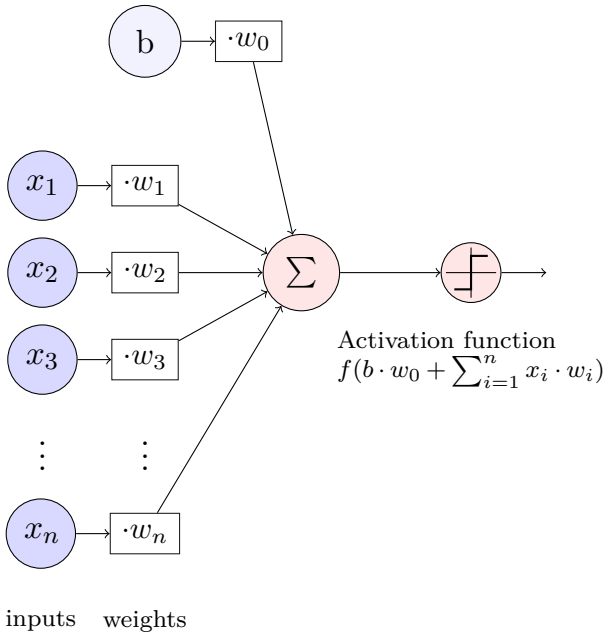


Figure 2.1.: Artificial neuron or Perceptron with bias node b.

weight parameter, but only weight parameters of connections between A-units and R-units are considered to be variables, whereas those between S-units and A-units are fixed. Since there are no variable parameters before the A-unit layer, the S- and A-unit layer can be merged together to form one single input layer. Thus, we ultimately obtain the simple network structure of Fig. 2.1, except for the bias node.

2.2. Multilayer Perceptron

In 1969 Minsky and Papert proved that the capabilities of the Perceptron were limited [27]. They showed that there is a wide range of problems, which can not be solved by a Perceptron, such as learning non-linearly separable patterns. These limitations can be overcome by utilizing multilayer Perceptrons (MLP). This class of neural networks is divided into minimum three layers. There is one input and one output layer, as is the case for the single-layer Perceptron. Additionally, in between, an additional layer, commonly called hidden layer, is present. Each neuron of the hidden layer receives signals from incoming connections from neurons of the input layer and transfers information to neurons of the output layer by outgoing connections. A MLP can also have more than one hidden layer. In this case, the layers are numbered and connected as depicted in Fig. 2.2. Another feature of a MLP are fully-connected layers. This means, all neurons of one layer are connected with all neurons of the subsequent one. A MLP containing only one hidden layer is very similar to Rosenblatt's simple Perceptron. However, there is one crucial difference.

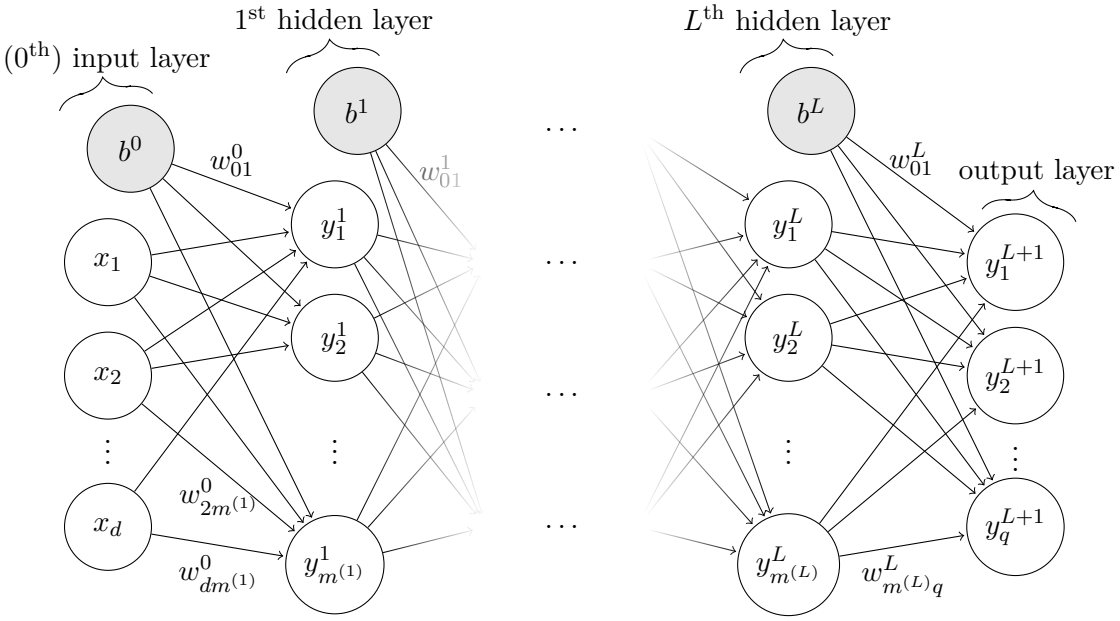


Figure 2.2.: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ neurons.

Weights connecting neurons of the input layer (sensory unit layer) with neurons of the hidden layer (association unit layer) are adjustable and not fixed. At the time Rosenblatt published his descriptions of the Perceptron, no learning algorithm was known for fast and successfully adjusting the variable weight parameters. This changed after Werbos analyzed the backpropagation algorithm in his PhD thesis in 1974 [28]. This algorithm was originally developed in the field of control theory, but Werbos was the first who suggested its application to weight adjustment in neural networks. After more than ten years Rumelhart *et al.* succeeded in showing experimentally that, by applying the backpropagation algorithm, a neural network with hidden layers can solve problems of the type a single-layer Perceptron failed at [29].

In 1991 Hornik proved the *universal approximation theorem* [30]. It states that for every function, $g : \mathbb{R}^n \to \mathbb{R}^m$, and an arbitrarily small real number $\epsilon > 0$, there exists a MLP with at least one hidden layer, consisting of neurons with a bonded, continuous and non-constant activation function f, so that $|G(x) - g(x)| < \epsilon$, where $x \in \mathbb{R}^n$ and $G(x)$ is the approximate representation of the function, $g(x)$, given by the MLP. This theoretical result is a very strong statement since it implies that a MLP with only one hidden layer could approximate every arbitrary function. The significance of this theorem remains rather theoretical than practical, though, because the hidden layer may be unfeasibly large and the algorithm determining the weights may fail to learn and generalize correctly [31].

The MLP is a subtype of a neural network class, which is the class of feedforward neural networks (FFNNs). Other NN types of this class are the aforementioned

single-layer Perceptron, the radial basis function network [32] and convolutional neural networks (CNN) [33], although CNNs are a very specialized kind of FFNN. The key feature of FFNNs is that information only moves in one direction, from one layer to the subsequent one, as opposed to recurrent neural networks, where connections can also form circles [34]. In this thesis we only consider FFNNs and will thus not go into detail about other NN classes.

2.3. Mathematical basics of multilayer Perceptrons

Before we discuss different activation functions, backprogagation and other training algorithms, we first introduce some notations. The depth of the MLP is the number of hidden layers L. The width $m^{(l)}$ is the number of neurons of the lth layer, ignoring the additional bias node. The output layer is always defined as the $(L+1)$th layer. The weight associated with the connection, which points from neuron a of layer $l-1$ to neuron b of layer l, is denoted by w_{ab}^{l-1}. The output value of the kth node of the lth hidden layer is denoted by y_k^l and x_k is the value of the kth node of the input layer. The input layer is also called the zeroth layer and instead of x_k we can also write y_k^0. Except for the input layer, y_k^l is calculated by

$$y_k^l = f_k^l\left(u_k^l\right) = f_k^l\left(w_{0k}^{l-1} + \sum_{s=1}^{m^{(l-1)}} w_{sk}^{l-1} y_s^{l-1}\right), \tag{2.2}$$

where f_k^l is the activation function of the kth node of the lth layer. Each layer other than the output layer contains a bias node, which is always treated as the zeroth neuron of a constant value $b^l = y_0^l = 1$. The activation of a neuron is the sum of all weighted inputs. To be more precise, the activation u_k^l of the kth neuron of layer l is defined by

$$u_k^l = w_{0k}^{l-1} + \sum_{s=1}^{m^{(l-1)}} w_{sk}^{l-1} y_s^{l-1}\,. \tag{2.3}$$

A training instance r is a single pair of an input vector $\boldsymbol{X}_r \in \mathbb{R}^d$ and its corresponding output vector $\boldsymbol{Y}_r \in \mathbb{R}^q$, where d and q are the number of input nodes (exclusively bias node) and output nodes, respectively. For the training of a MLP, we have to introduce a cost function or error function Γ. In many applications the mean square error (MSE) of the sum of all instances is used as error function. It is defined as

$$\Gamma_{\text{MSE}} = \frac{1}{N}\sum_{r=1}^{N}\sum_{i=1}^{q}\left(Y_{i,r} - y_{i,r}^{(L+1)}\right)^2, \tag{2.4}$$

where $Y_{i,r}$ is the ith element of the desired output vector $\boldsymbol{Y}_r$ of training instance r, $y_{i,r}^{(L+1)}$ is the corresponding output of the neural network and N is the number

of training instances. Many training algorithms require the gradients of the error function with respect to the weight parameters. A very efficient way of obtaining this gradient is backpropagation [28].

2.3.1. Backpropagation

We will now derive the well-known backpropagation algorithm for MLPs. The aim of backpropagation is to calculate the derivative of the error function with respect to all weights of the MLP, formally $g_{ij}^l = \partial\Gamma \Big/ \partial w_{ij}^l \ , \forall\ i, j, l$.
Formally, the MLP is a function $\boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x})$, where $\boldsymbol{\Phi} : \mathbb{R}^d \to \mathbb{R}^q$, $\boldsymbol{w}$ is the vector of all weight parameters of the NN and $\boldsymbol{x}$ is the vector of input values. This function can also be seen as a nested function

$$\boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{\varphi}^L \left(\boldsymbol{w}^L, \boldsymbol{\varphi}^{L-1} \left(\boldsymbol{w}^{L-1}, \boldsymbol{\varphi}^{L-2} \left(..., \boldsymbol{\varphi}^0 \left(\boldsymbol{w}^0, \boldsymbol{x} \right) ... \right) \right) \right), \tag{2.5}$$

where $\boldsymbol{\varphi}^l : \mathbb{R}^{m^{(l)}} \to \mathbb{R}^{m^{(l+1)}}$ and $\boldsymbol{w}^l$ is the vector of all weights between layer l and l+1. Each function $\boldsymbol{\varphi}^l$ maps the output values of the neurons of layer l, $\{y_i^l\}$, onto the values of the neurons of the next layer l+1, $\{y_i^{l+1}\}$. The error function Γ for a single training instance depends then, for a fixed input vector and target output vector, on $\boldsymbol{\varphi}^L$ and all its arguments and nested arguments $\boldsymbol{\varphi}^0, \boldsymbol{\varphi}^1, ..., \boldsymbol{\varphi}^{L-1}$ as well as $\boldsymbol{w}^0, \boldsymbol{w}^1, ..., \boldsymbol{w}^L$ and $\boldsymbol{x}$:

$$\Gamma = \Gamma \left(\boldsymbol{\varphi}^L \left(\boldsymbol{w}^L, \boldsymbol{\varphi}^{L-1} \left(..., \boldsymbol{\varphi}^0 \left(\boldsymbol{w}^0, \boldsymbol{x} \right) ... \right) \right) \right). \tag{2.6}$$

To calculate all derivatives of the error function with respect to the weights g_{ij}^l, the chain rule must be applied.
The backpropagation algorithm solves this problem iteratively. We start by computing the derivative $g_{\alpha\beta}^L = \partial\Gamma \Big/ \partial w_{\alpha\beta}^L$. Therefore, we write

$$\varphi_k^L = f_k^{L+1} \left(w_{0k}^L + \sum_{i=1}^{m^{(L)}} w_{ik}^L y_i^L \right) \tag{2.7}$$

$$= f_k^{L+1} \left(w_{0k}^L + \sum_{i=1}^{m^{(L)}} w_{ik}^L \ f_i^L \left(w_{0i}^{L-1} + \sum_{j=1}^{m^{(L-1)}} w_{ji}^{L-1} y_j^{L-1} \right) \right), \tag{2.8}$$

where φ_k^L is the kth component of $\boldsymbol{\varphi}^L$ and f_k^{L+1} is the activation function of neuron k of layer L+1. For the sake of convenience, we change the notation of Eq. 2.7 to

$$\varphi_k^l = f_k^{l+1} \left(\sum_{i=0}^{m^{(l)}} w_{ik}^l y_i^l \right) \tag{2.9}$$

with $y_0^l = 1$. For weights between the last hidden layer and the output layer, by applying the chain rule, we obtain

$$g_{\alpha\beta}^L = \frac{\partial \Gamma}{\partial w_{\alpha\beta}^L} = \frac{\partial \Gamma}{\partial f_\beta^{(L+1)}} \frac{\partial f_\beta^{L+1}}{\partial w_{\alpha\beta}^L}. \tag{2.10}$$

Again applying the chain rule to the expression $\partial f_\beta^{L+1} \Big/ \partial w_{\alpha\beta}^L$ leads to

$$\frac{\partial \Gamma}{\partial w_{\alpha\beta}^L} = \frac{\partial \Gamma}{\partial f_\beta^{(L+1)}} \frac{\partial f_\beta^{L+1}}{\partial u_\beta^{L+1}} \frac{\partial u_\beta^{L+1}}{\partial w_{\alpha\beta}^L} \tag{2.11}$$

and since $u_\beta^{L+1} = \sum_{i=0}^{m^{(L)}} w_{i\beta}^L y_i^L$, we get $\partial u_\beta^{L+1} \Big/ \partial w_{\alpha\beta}^L = y_\alpha^L$ and thus

$$g_{\alpha\beta}^L = \frac{\partial \Gamma}{\partial f_\beta^{(L+1)}} \frac{\partial f_\beta^{L+1}}{\partial u_\beta^{L+1}} y_\alpha^L. \tag{2.12}$$

For $\alpha = 0$, we have the special case of bias nodes $y_0^l = b^l = 1$ for $0 \leq l \leq L$. These derivatives, g_{0k}^l, will also be referred to as h_k^l. Now we will calculate derivatives for the weights between layers L and $L-1$,

$$g_{\alpha\beta}^{L-1} = \sum_{i=0}^{m^{(L+1)}} \frac{\partial \Gamma}{\partial f_i^{(L+1)}} \frac{\partial f_i^{L+1}}{\partial u_i^{L+1}} \frac{\partial u_i^{L+1}}{\partial f_\beta^L} \frac{\partial f_\beta^L}{\partial u_\beta^L} \frac{\partial u_\beta^L}{\partial w_{\alpha\beta}^{L-1}} \tag{2.13}$$

and with $\partial u_i^{L+1} \Big/ \partial f_\beta^L = w_{\beta i}^L$ and $\partial u_\beta^L \Big/ \partial w_{\alpha\beta}^{L-1} = y_\alpha^{L-1}$ we get

$$g_{\alpha\beta}^{L-1} = \sum_{i=0}^{m^{(L+1)}} \frac{\partial \Gamma}{\partial f_i^{(L+1)}} \frac{\partial f_i^{L+1}}{\partial u_i^{L+1}} w_{\beta i}^L \frac{\partial f_\beta^L}{\partial u_\beta^L} y_\alpha^{L-1}. \tag{2.14}$$

We can rewrite Eq. 2.14 by using the definition of $g_{0ß}^L = h_i^L$:

$$\frac{\partial \Gamma}{\partial w_{\alpha\beta}^{L-1}} = g_{\alpha\beta}^{L-1} = \sum_{i=0}^{m^{(L+1)}} h_i^L w_{\beta i}^L \frac{\partial f_\beta^L}{\partial u_\beta^L} y_\alpha^{L-1}. \tag{2.15}$$

Then, it can be easily shown that for all $l \leq L-1$:

$$g_{\alpha\beta}^{l-1} = \sum_{i=0}^{m^{(l+1)}} h_i^l w_{\beta i}^l \frac{\partial f_\beta^l}{\partial u_\beta^l} y_\alpha^{l-1}. \tag{2.16}$$

Proof:
For $l = L-1$, we have

$$g_{\beta\zeta}^l = \sum_{i=0}^{m^{(l+2)}} h_i^{l+1} w_{\zeta i}^{l+1} \frac{\partial f_\zeta^{l+1}}{\partial u_\zeta^{l+1}} y_\beta^l. \tag{2.17}$$

From applying the chain rule and the structure of Eq. 2.8, we can conclude for $g_{\alpha\beta}^{l-1}$:

$$g_{\alpha\beta}^{l-1} = \sum_{i=0}^{m^{(l+2)}} h_i^{l+1} \sum_{j=0}^{m^{(l+1)}} w_{ji}^{l+1} \frac{\partial f_j^{l+1}}{\partial u_j^{l+1}} w_{\beta j}^{l} \frac{\partial f_\beta^l}{\partial u_\beta^l} y_\alpha^{l-1}. \tag{2.18}$$

By changing the position of the sum over index j, we find

$$g_{\alpha\beta}^{l-1} = \sum_{j=0}^{m^{(l+1)}} \underbrace{\sum_{i=0}^{m^{(l+2)}} h_i^{l+1} w_{ji}^{l} \frac{\partial f_j^{l+1}}{\partial u_j^{l+1}}}_{h_j^l} w_{\beta j}^{l} \frac{\partial f_\beta^l}{\partial u_\beta^l} y_\alpha^{l-1} \tag{2.19}$$

and thus we finally get

$$g_{\alpha\beta}^{l-1} = \sum_{j=0}^{m^{(l+1)}} h_j^l w_{\beta j}^{l} \frac{\partial f_\beta^l}{\partial u_\beta^l} y_\alpha^{l-1}. \tag{2.20}$$

□

Therefore, we see that all $g_{\alpha\beta}^{l}$ can be obtained by iterating through all layers, from the last hidden layer to the input layer. Only the output layer is treated in a special way in order to start the iteration. If we refer to backpropagation, we only mean the method of obtaining the gradient of the error function w.r.t. the weights but originally backpropagation also included a rule for changing the weights based on this gradient. The way we modify the weights is covered in the Section 2.4.

2.3.2. Activation functions

So far, we have not yet discussed the shape of the activation function f. Only in the context of the universal approximation theorem we have mentioned some restrictions on the activation function of neurons of the necessary hidden layer. The activation function of neurons of the hidden layer needs to be bounded, continuous and non-constant to fulfill the requirements of the theorem [30]. However, it shall be noted that also activation functions which do not meet these criteria can be successfully used.

It is common to use the same activation function within one layer but vary the type of function from layer to layer. Neurons of the output layer do mostly just apply the identity function

$$f_{\mathrm{id}}(x) = x. \tag{2.21}$$

A bounded activation function would force the neural network function to yield numbers within a certain interval, which is not desired in most cases. Very popular activation functions are the sigmoid function

$$f_{\text{sig}}(x) = \frac{1}{1 + e^{-x}} \tag{2.22}$$

and the hyperbolic tangent function

$$f_{\tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.23}$$

These two functions are depicted in Fig. 2.3. An example for a non-bonded activation

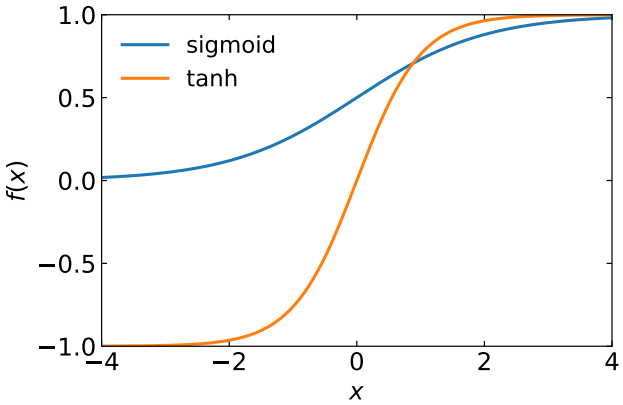


Figure 2.3.: Sigmoid and hyperbolic tangent function.

function is the rectifier function (Fig. 2.4)

$$f_{\text{rect}}(x) = \max(0, x). \tag{2.24}$$

The rectifier function does not meet all criteria of the universal approximation theorem. Nevertheless, it can outperform other activation functions [35], which are consistent with the universal approximation theorem. Of course, also the identity function is a non-bonded function but its usage is usually limited to the output layer which is not contradictory to the universal approximation theorem. The question which arises here is: What activation function should be used for a specific problem? This can not be answered in a generally accepted way. Although one can compare different activation functions for a given training data set and network structure, the success of an activation function might also highly depend on the optimization algorithms and its settings for finding the weights. Each activation function might have essential drawbacks. For instance, consider the rectifier function and an optimization algorithm which uses components of the gradient of the error function g_{ij}^{l} for updating the weight w_{ij}^{l}. If the weights are adjusted during the optimization process or initialized in such a way that a neuron of the first hidden layer always receives a negative activation for every training instance, the weights of incoming connections of

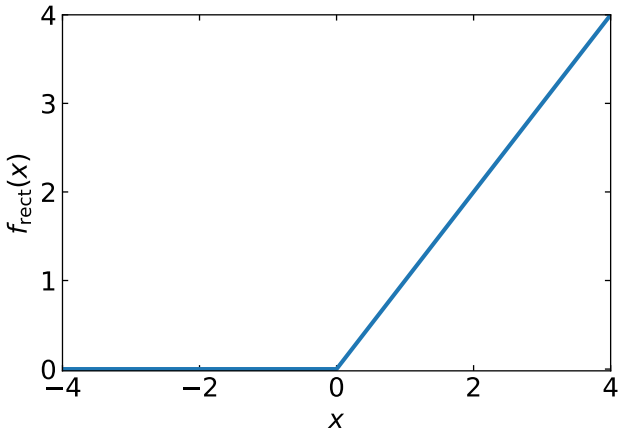


Figure 2.4.: Rectifier function.

this neuron can not change anymore (see Eq. 2.16), because the derivative $\partial f_\beta^l \big/ \partial u_\beta^l$ will always be zero. However, also the sigmoid and hyperbolic tangent function have similar drawbacks. If the activation of neurons is very small (negative) or large, the abovementioned derivative will almost vanish since these activation functions take on saturated values of -1, 0 or 1. The consequence is that the optimization algorithm could be vastly decelerated.

2.4. Optimization algorithms

First of all, we make a distinction between weight parameters and hyperparameters. Weight parameters are real numbers specifying the strength of the connection between neurons. As opposed to them, the number of hidden layers, the number of neurons in each hidden layer and all parameters needed for controlling the optimization algorithm are called hyperparameters. If we refer to optimization of a NN, we mean optimization of weight parameters for a given network structure and training algorithm.

2.4.1. Minimizing the error function

The key feature of optimizing NNs is the scalar-valued error function Γ. It is a measure of how well the NN reaches the goal of approximating the reference function. The error function can be written as

$$\Gamma = \Gamma\left(\boldsymbol{x}_1, ..., \boldsymbol{x}_P, \boldsymbol{Y}_1, ..., \boldsymbol{Y}_P, \boldsymbol{w}\right), \tag{2.25}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{Y}_i$ are the input and desired output vector of the i-th training instance, respectively, P is the number of training instances and $\boldsymbol{w}$ is the vector of all weight parameters. Another way of writing Eq. 2.25 is

$$\Gamma = \bar{\Gamma}\left(\boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x}_1), ..., \boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x}_P), \boldsymbol{Y}_1, ..., \boldsymbol{Y}_P, \boldsymbol{w}\right), \tag{2.26}$$

where $\boldsymbol{\Phi}(\boldsymbol{x}_p, \boldsymbol{w})$ is the NN function. The weight vector $\boldsymbol{w}$ is not just an argument for the NN function but also a separate argument of $\bar{\Gamma}$ since there are error functions which contain a regularization term [36] in order to, for instance, keep weight parameters small. Many error functions are the sum of separate terms without a regularization term

$$\Gamma = \sum_{p=1}^{P} \Gamma_p\left(\boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x}_p), \boldsymbol{y}_i\right). \tag{2.27}$$

This is the type of error function we are using in this work. To be more precise, we want to minimize

$$\Gamma = \sum_{p=1}^{P} ||\boldsymbol{\Phi}(\boldsymbol{w}, \boldsymbol{x}_p) - \boldsymbol{y}_p||^2. \tag{2.28}$$

2.4.2. Gradient-based optimization

The most basic and intuitive gradient-based algorithm for optimizing the weights of a neural network is probably gradient descent (GD). By using the gradient $\nabla_{\boldsymbol{w}}\Gamma$ we can update the weights according to

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \gamma \cdot \nabla_{\boldsymbol{w}}\Gamma, \tag{2.29}$$

where γ is the learning rate and t is number of completed updates. For $t = 0$ an initial guess $\boldsymbol{w}_{(0)}$ has to be made and γ does not need to be a constant but can be varied from update to update. It shall be emphasized that γ can play a critical role in the algorithm, often determining whether the algorithm converges or diverges [37]. One intuitive implementation for defining γ at each iteration step is to find a solution to

$$\min_{\gamma>0}\left\{\Gamma\left(\boldsymbol{w}_{(t)} - \gamma \cdot \nabla_{\boldsymbol{w}}\Gamma\right)\right\}. \tag{2.30}$$

For the sake of simplicity, we suppressed other arguments than $\boldsymbol{w}$ in Γ. This method of determining the learning rate at every iteration step is called line search since it is a one-dimensional search for the lowest value of the error function Γ. However, computational costs of finding the minimum of Eq. 2.30 can be quite high and a rough one-dimensional grid search as well as an upper bound for γ may be employed.

Once the method of determining γ at each iteration step is chosen, GD is purely deterministic since the computation of the error function and its gradient does not involve random processes. A popular variant of GD is stochastic gradient descent (SGD) [38]. The main difference between GD and SGD is the error function used for updating the weights. For SGD only one randomly drawn training instance is used per weight update. If the error function can be decomposed as a sum of individual errors, Γ_i, from every training instance i (see Eq. 2.27), SGD can be easily defined as

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \gamma \cdot \nabla_{\boldsymbol{w}} \Gamma_i, \tag{2.31}$$

where index i is sampled from the interval $[1, N]$. By this modification, the sequence $\left(\boldsymbol{w}_{(t)}\right)_{t\in\mathbb{N}}$ becomes a random process and the direction of the next weight update is not determined before index i is sampled. One advantage of SGD over GD obviously is its lower computational effort per weight update. The stochastic nature of SGD also enables the iteration to escape from local minima of Γ that might trap GD. Nevertheless, the best performance of algorithms of this type might be possible when the computation of the gradients are based on more than one but considerably less than all available training instances. This approach is referred to as mini-batch gradient descent (MBGD). The update equation of MBGD is

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \gamma \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{w}} \Gamma_i \left(\boldsymbol{w}_{(t)}\right). \tag{2.32}$$

The n indices per iteration are to be randomly chosen from the interval $[1, N]$ and thus $\left(\boldsymbol{w}_{(t)}\right)_{t\in\mathbb{N}}$ is again a random process.

2.4.3. Optimization using second order derivatives

So far we have just employed information from first order derivatives, namely $\nabla_{\boldsymbol{w}}\Gamma$. The performance of methods using higher order derivatives may be better. A well-known example, Newton's method (NM), utilizes first and second order derivatives of the error function with respect to weight parameters. The idea behind this is to construct a quadratic approximation of the objective error function at the current point of the optimization space that matches the first and second derivatives and then this quadratic function is to be minimized [39]. The so-obtained point of the optimization space is used as the starting position of the next iteration step. Using $\boldsymbol{w}_{(t)}$ as starting point in iteration step $t+1$ we get

$$\begin{aligned}\Gamma^*(\boldsymbol{w}) = \Gamma(\boldsymbol{w}_{(t)}) + (\boldsymbol{w} - \boldsymbol{w}_{(t)})^{\mathrm{T}} \nabla_{\boldsymbol{w}} \Gamma(\boldsymbol{w}_{(t)})(\boldsymbol{w}_{(t)}) \\ + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{(t)})^{\mathrm{T}} \boldsymbol{H}(\boldsymbol{w}_{(t)})(\boldsymbol{w} - \boldsymbol{w}_{(t)}),\end{aligned} \tag{2.33}$$

where $\Gamma^*(\boldsymbol{w})$ is the quadratic approximating polynomial of $\Gamma(\boldsymbol{w})$ and $\boldsymbol{H}(\boldsymbol{w}_{(t)})$ is the Hessian matrix at $\boldsymbol{w}$=$\boldsymbol{w}_{(t)}$. The jth element of the ith row of the Hessian matrix, H_{ij}, is

$$H_{ij} = \frac{\partial^2 \Gamma}{\partial w_i \partial w_j}. \tag{2.34}$$

A necessary condition for a minimum of $\Gamma^*(\boldsymbol{w})$ is $\partial\Gamma^*/\partial w_i = 0$ for all i. Therefore, we get

$$0 = \nabla_{\boldsymbol{w}}\Gamma(\boldsymbol{w}_{(t)}) + \boldsymbol{H}(\boldsymbol{w}_{(t)})(\boldsymbol{w} - \boldsymbol{w}_{(t)}). \tag{2.35}$$

If $H(\boldsymbol{w}_{(t)})$ is positive definite, a minimum exists at

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \boldsymbol{H}(\boldsymbol{w}_{(t)})^{-1}\ \nabla_{\boldsymbol{w}}\Gamma(\boldsymbol{w}_{(t)}). \tag{2.36}$$

Eq. 2.36 is called Newton's iteration or NM. This recursive algorithm has one essential drawback. If $\boldsymbol{H}(\boldsymbol{w}_{(t)})$ is not positive definite, $\Delta\boldsymbol{w} = \boldsymbol{w}_{(t+1)} - \boldsymbol{w}_{(t)}$ does not necessarily point into the direction of decreasing $\Gamma(\boldsymbol{w})$ [39]. This potential misbehavior of the algorithm can be improved by the Levenberg-Marquardt modification of NM. Therefore, $\boldsymbol{H}(\boldsymbol{w}_{(t)})$ is replaced by a matrix $\boldsymbol{M}(\boldsymbol{w}_{(t)})$

$$\boldsymbol{M}(\boldsymbol{w}_{(t)}) = \boldsymbol{H}(\boldsymbol{w}_{(t)}) + \mu\boldsymbol{I}, \tag{2.37}$$

where $\mu \in \mathbb{R}^+$ and I is the identity matrix.

$$\boldsymbol{M}(\boldsymbol{w}_{(t)}) = \boldsymbol{H}(\boldsymbol{w}_{(t)}) + \mu\boldsymbol{I}. \tag{2.38}$$

To see this, we let λ_i be the i-th eigenvalue of $\boldsymbol{H}(\boldsymbol{w}_{(t)})$ and $\boldsymbol{v}_i$ the corresponding eigenvector. We consider the product of $\boldsymbol{M}(\boldsymbol{w}_{(t)})$ and $\boldsymbol{v}_i$

$$\begin{aligned}\boldsymbol{M}(\boldsymbol{w}_{(t)})\boldsymbol{v}_i &= \boldsymbol{H}(\boldsymbol{w}_{(t)})\boldsymbol{v}_i + \mu\boldsymbol{I}\boldsymbol{v}_i \\ &= \lambda_i\boldsymbol{v}_i + \mu\boldsymbol{v}_i \\ &= (\lambda_i\boldsymbol{v}_i + \mu)\boldsymbol{v}_i.\end{aligned} \tag{2.39}$$

If μ is sufficiently large, we obtain a strictly positive definite matrix $\boldsymbol{M}(\boldsymbol{w}_{(t)})$. The Levenberg-Marquardt modification of NM is represented by the recursive equation

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \Big(H(\boldsymbol{w}_{(t)}) + \mu\boldsymbol{I}\Big)^{-1} \nabla_{\boldsymbol{w}}\Gamma(\boldsymbol{w}_{(t)}). \tag{2.40}$$

An advantage of optimization algorithms using second order derivatives are their possibly better convergence properties (once $\boldsymbol{w}_{(t)}$ is close to the minimum) compared against algorithms that employ only gradients [39]. For a very complex ob-

jective error function and an initial guess far away from the minimum, convergence properties in the vicinity of the minimum are not the only criterion. The need of calculating second order derivatives of the error function and, thus, also of the NN function w.r.t. the weights, as well as the necessary inversion of the Hessian matrix, are drawbacks since the computational load is considerably higher than that of GD. That is why often an approximation of the Hessian matrix is used. Such an approximation can be found by starting from Eq. 2.28. For the sake of clarity we rewrite this equation to

$$\frac{1}{2}\Gamma(\boldsymbol{w}) = \frac{1}{2}\sum_{p=1}^{P} ||\boldsymbol{\Phi}(\boldsymbol{x}_p, \boldsymbol{w}) - \boldsymbol{Y}_p||^2 = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{N}(\Phi_i(\boldsymbol{x}_p, \boldsymbol{w}) - Y_{p,i})^2 = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{N}\delta_{p,i}^2\,, \tag{2.41}$$

where $\delta_{p,i} = \Phi_i(\boldsymbol{x}_p, \boldsymbol{w}) - Y_{p,i}$ and Φ_i is the ith component of the NN function. $Y_{p,i}$ is the ith component of the output of the reference function for training instance p. We also introduced a factor $1/2$. The derivative of Γ w.r.t. weight w_k then is

$$\frac{\partial\Gamma}{\partial w_k} = \sum_{p=1}^{P}\sum_{i=1}^{N}\delta_{p,i}\frac{\partial\delta_{p,i}}{\partial w_k} \tag{2.42}$$

and the kl-th component of the Hessian matrix $\boldsymbol{H}$ is given by

$$H_{kl} = \frac{\partial^2\Gamma}{\partial w_k \partial w_l} = \sum_{p=1}^{P}\sum_{i=1}^{N}\left(\frac{\partial\delta_{p,i}}{\partial w_k}\frac{\partial\delta_{p,i}}{\partial w_l} + \delta_{p,i}\frac{\partial^2\delta_{p,i}}{\partial w_k \partial w_l}\right). \tag{2.43}$$

An approximative Hessian matrix $\bar{\boldsymbol{H}}$, sometimes referred to as Gauss-Newton Hessian, can is defined as [40]

$$\bar{H}_{kl} = \sum_{p=1}^{P}\sum_{i=1}^{N}\frac{\partial\delta_{p,i}}{\partial w_k}\frac{\partial\delta_{p,i}}{\partial w_l} \tag{2.44}$$

by omitting the second order derivatives in Eq. 2.43. $\bar{\boldsymbol{H}}$ is often written as

$$\bar{\boldsymbol{H}} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}, \tag{2.45}$$

where the Jacobian matrix $\boldsymbol{J}$ is arranged as follows [41]:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial \delta_{1,1}}{\partial w_1} & \frac{\partial \delta_{1,1}}{\partial w_2} & \cdots & \frac{\partial \delta_{1,1}}{\partial w_V} \\ \frac{\partial \delta_{1,2}}{\partial w_1} & \ddots & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial \delta_{1,N}}{\partial w_1} & & \ddots & \frac{\partial \delta_{1,N}}{\partial w_1} \\ \frac{\partial \delta_{2,1}}{\partial w_1} & \ddots & & \\ \frac{\partial \delta_{2,2}}{\partial w_1} & \ddots & & \vdots \\ \vdots & & \ddots & \\ \frac{\partial \delta_{P,N}}{\partial w_1} & \cdots & & \frac{\partial \delta_{P,N}}{\partial w_V} \end{bmatrix}. \tag{2.46}$$

In case of $N = 1$ and $P = 1$, we obtain a row vector. $\bar{\boldsymbol{H}}$ is a positive semidefinite matrix [40]. The well-known **Levenberg-Marquardt algorithm** (LMA) utilizes this approach as follows:

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \left(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} + \mu\boldsymbol{I}\right)^{-1} \boldsymbol{J}^{T}\boldsymbol{\delta}, \tag{2.47}$$

where $\mu > 0$ and $\boldsymbol{\delta}$ is introduced as

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_{1,1} \\ \delta_{1,2} \\ \vdots \\ \delta_{1,N} \\ \delta_{2,1} \\ \vdots \\ \delta_{P,N} \end{bmatrix} \tag{2.48}$$

and thus the kth element of the matrix-vector product $\boldsymbol{J}^{T}\boldsymbol{\delta}$ is the derivative of the error function w.r.t. w_k (see Eq. 2.42), the kth component of $\nabla_{\boldsymbol{w}}\Gamma$. Since $\bar{\boldsymbol{H}} = J^{\mathrm{T}}\boldsymbol{J}$ is positive semidefinite, the matrix $\left(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} + \mu\boldsymbol{I}\right)$ is always invertible for $\mu > 0$. For $\mu << 1$ the LMA approaches NM and for $\mu >> 1$ closely resembles GD with learning rate $\gamma = 1/\mu$.

2.4.4. Kalman filter methods

The method of our choice for weights optimization is the Kalman filter (KF), to be more precise, a version of the Extended Kalman filter (EKF), the multistream extended Kalman filter (MEKF) with forgetting factor.
The KF is a state estimation technique introduced by Kalman in 1960 [42, 43]. The system the state of which, x, is to be estimated is a linear dynamic system perturbed by white noise. The estimate is based on measurements y linearly related

to the state but also corrupted by white noise [44]. The measurement equation can thus be formulated as follows:

$$y_t = \boldsymbol{H}_{t_k} \boldsymbol{x}_{t_k} + \boldsymbol{r}_{t_k}, \tag{2.49}$$

where $\boldsymbol{y}_t$ is a m-vector, $\boldsymbol{H}_t$ is a $m \times n$-matrix and x_t is the n-dimensional state vector at time t_k. The m-vector $\boldsymbol{r}_{t_k}$ represents discrete-time white noise. Therefore, $\boldsymbol{r}_{t_k}$ is a multivariate normal distribution with zero mean and covariance $\boldsymbol{R}_{t_k}$:

$$\boldsymbol{v}_{t_k} \sim \mathcal{N}(0, \boldsymbol{R}_{t_k}). \tag{2.50}$$

The state vector x_t can be propagated in time in a continuous or discrete way. For our purpose, the discretized version is sufficient:

$$x_{t_{k+1}} = \boldsymbol{F}_{t_{k+1}, t_k} \boldsymbol{x}_{t_k} + \boldsymbol{q}_{t_k}. \tag{2.51}$$

The $n \times n$ state transition matrix $\boldsymbol{F}_{t_k}$ is applied to the state vector at time t_k, where $\boldsymbol{q}_{t_k}$ is similarly defined as $\boldsymbol{r}_{t_k}$:

$$\boldsymbol{w}_{t_k} \sim \mathcal{N}(0, \boldsymbol{Q}_{t_k}). \tag{2.52}$$

There are different derivations of the relations of the KF [42, 45]. We will not go into detail but we will summarize the most essential points. In accordance with the derivation in Maybeck [45], the initial state vector estimation $\hat{\boldsymbol{x}}_{t_0}$ is represented by a multivariate Gaussian distribution, which means we do not precisely know the initial state. It can be shown that the conditional mean $\hat{\boldsymbol{x}}_{t_k}$ and covariance matrix $\boldsymbol{P}_{t_k}$ of this distribution can be expressed at every time step, conditioned on the complete measurement history $\{\boldsymbol{y}_{t_0}, \boldsymbol{y}_{t_1}, ..., \boldsymbol{y}_{t_k}\}$. The conditional mean is also minimizing the MSE [45],

$$\mathbb{E}[(\hat{\boldsymbol{x}}_{t_k} - \boldsymbol{x}_{t_k})^2]. \tag{2.53}$$

The resulting recursion, known as KF, can be summarized as follows:

In a first step the conditional mean and covariance matrix are propagated without any knowledge of the measurement of the current time step, denoted by $\hat{\boldsymbol{x}}_{t_k}^-$ and $\boldsymbol{P}_{t_k}^-$,

$$\hat{\boldsymbol{x}}_{t_k}^- = \boldsymbol{F}_{t_k, t_{k-1}} \hat{\boldsymbol{x}}_{t_{k-1}}, \tag{2.54}$$

$$\boldsymbol{P}_{t_k}^- = \boldsymbol{F}_{t_k, t_{k-1}} \boldsymbol{P}_{t_{k-1}} \boldsymbol{F}_{t_k, t_{k-1}}^{\mathrm{T}} + \boldsymbol{Q}_{t_{k-1}}. \tag{2.55}$$

With the so-called Kalman gain matrix $\boldsymbol{G}_{t_k}$,

$$\boldsymbol{G}_{t_k} = \boldsymbol{P}_{t_k}^{-}\boldsymbol{H}_{t_k}^{\mathrm{T}}\left(\boldsymbol{H}_{t_k}\boldsymbol{P}_{t_k}^{-}\boldsymbol{H}_{t_k}^{\mathrm{T}} + \boldsymbol{R}_{t_k}\right)^{-1}, \tag{2.56}$$

we finally obtain the mean and covariance matrix by incorporating the current measurement $\boldsymbol{y}_{t_k}$:

$$\hat{\boldsymbol{x}}_{t_k} = \hat{\boldsymbol{x}}_{t_k}^{-} + \boldsymbol{G}_{t_k}\left(\boldsymbol{y}_{t_k} - \boldsymbol{H}_{t_k}\boldsymbol{x}_{t_k}^{-}\right), \tag{2.57}$$

$$\boldsymbol{P}_{t_k} = (\boldsymbol{I} - \boldsymbol{G}_{t_k}\boldsymbol{H}_{t_k})\boldsymbol{P}_{t_k}^{-}. \tag{2.58}$$

The KF can also be used for non-linear systems

$$x_{t_{k+1}} = \boldsymbol{f}(\boldsymbol{x}_{t_k}, t_k) + \boldsymbol{q}_{t_k}, \tag{2.59}$$

$$y_t = \boldsymbol{h}(\boldsymbol{x}_{t_k}, t_k) + \boldsymbol{r}_{t_k} \tag{2.60}$$

by linearizing the two non-linear functions $\boldsymbol{f}$ and $\boldsymbol{h}$ around the most recent state estimate $\hat{\boldsymbol{x}}_{t_k}$ to obtain

$$\boldsymbol{f}(\boldsymbol{x}_{t_k}, t_k) = \boldsymbol{f}(\hat{\boldsymbol{x}}_{t_k}, t_k) + \boldsymbol{F}_{t_k,t_{k-1}}(\boldsymbol{x}_{t_k} - \hat{\boldsymbol{x}}_{t_k}) + \epsilon_1, \tag{2.61}$$

$$\boldsymbol{h}(\boldsymbol{x}_{t_k}, t_k) = \boldsymbol{h}(\hat{\boldsymbol{x}}_{t_k}^{-}, t_k) + \boldsymbol{H}_{t_k}(\boldsymbol{x}_{t_k} - \hat{\boldsymbol{x}}_{t_k}^{-}) + \epsilon_2, \tag{2.62}$$

where ϵ_1 and ϵ_2 contain higher order terms and

$$\boldsymbol{F}_{t_k,t_{k-1}} = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, t_k)}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{t_k}}, \tag{2.63}$$

$$\boldsymbol{H}_{t_k} = \frac{\partial \boldsymbol{h}(\boldsymbol{x}, t_k)}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{t_k}^{-}}. \tag{2.64}$$

That is, the ijth element of $\boldsymbol{F}_{t_k,t_{k-1}}$ is the partial derivative of the ith component of $\boldsymbol{f}(\boldsymbol{x}, t_k)$ w.r.t. the jth component of $\boldsymbol{x}$, evaluated at $\boldsymbol{x} = \hat{\boldsymbol{x}}_{t_k}$. By neglecting the higher order terms ϵ_1 and ϵ_2 we obtain the following system state and measurement equations

$$\boldsymbol{x}_{t_{k+1}} = \boldsymbol{F}_{t_k,t_{k-1}}\boldsymbol{x}_{t_k} + \boldsymbol{s}_{t_k} + \boldsymbol{q}_{t_k}, \tag{2.65}$$

$$\boldsymbol{y}_{t_k} = \boldsymbol{H}_{t_k}\boldsymbol{x}_{t_k} + \boldsymbol{r}_{t_k} + \boldsymbol{r}_{t_k}, \tag{2.66}$$

for $\boldsymbol{s}_{t_k}$ and $\boldsymbol{r}_{t_k}$ defined as

$$\boldsymbol{s}_{t_k} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{t_k}, t_k) - \boldsymbol{F}_{t_k,t_{k-1}}\hat{\boldsymbol{x}}_{t_k}, \tag{2.67}$$

$$\boldsymbol{r}_{t_k} = \boldsymbol{h}(\hat{\boldsymbol{x}}_{t_k}^{-}, t_k) - \boldsymbol{H}_{t_k}\hat{\boldsymbol{x}}_{t_k}^{-}. \tag{2.68}$$

The EKF is, then, an exact KF for the linearized state and measurement Eq. 2.65 and Eq. 2.66 [46, 47] and can be written as

$$\hat{\boldsymbol{x}}_{t_k}^- = \boldsymbol{f}(\boldsymbol{x}_{t_{k-1}}, t_k), \tag{2.69}$$

$$\boldsymbol{P}_{t_k}^- = \boldsymbol{F}_{t_k,t_{k-1}} \boldsymbol{P}_{t_{k-1}} \boldsymbol{F}_{t_k,t_{k-1}}^{\mathrm{T}} + \boldsymbol{Q}_{t_{k-1}}, \tag{2.70}$$

$$\boldsymbol{G}_{t_k} = \boldsymbol{P}_{t_k}^- \boldsymbol{H}_{t_k}^{\mathrm{T}} \left(\boldsymbol{H}_{t_k} \boldsymbol{P}_{t_k}^- \boldsymbol{H}_{t_k}^{\mathrm{T}} + \boldsymbol{R}_{t_k} \right)^{-1}, \tag{2.71}$$

$$\hat{\boldsymbol{x}}_{t_k} = \hat{\boldsymbol{x}}_{t_k}^- + \boldsymbol{G}_{t_k} \left(\boldsymbol{y}_{t_k} - \boldsymbol{h}(\boldsymbol{x}_{t_k}^-, t_k) \right), \tag{2.72}$$

$$\boldsymbol{P}_{t_k} = (\boldsymbol{I} - \boldsymbol{G}_{t_k} \boldsymbol{H}_{t_k}) \boldsymbol{P}_{t_k}^-. \tag{2.73}$$

Hence, the EKF is still an optimal estimator in the sense of the linearized Eqs. 2.65 and 2.66, but it is no longer an optimal estimator for the original non-linear set of Eqs. 2.59 and 2.60. This also implicates that $\hat{\boldsymbol{x}}_{t_k}$ and $\boldsymbol{P}_{t_k}$ are just an approximate conditional mean [46] and that the EKF is, unlike its linear counterpart, not minimizing the MSE in the sense of Eq. 2.53 for the non-linear system.
Now we will show how the EKF can be utilized for the optimization of weight parameters of a NN. Therefore, it is important to realize that the NN training procedure can be seen as a special case of the non-linear system Eqs. 2.59 and 2.60:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{q}_k, \tag{2.74}$$

$$\boldsymbol{y}_k = \boldsymbol{\Phi}(\boldsymbol{x}_k, \boldsymbol{w}_k) + \boldsymbol{r}_k. \tag{2.75}$$

The state of the system which shall be estimated is the vector of all weight parameters. The non-linear function $\boldsymbol{f}(\boldsymbol{w}_k, k)$ is replaced by the identity function $\boldsymbol{f}(\boldsymbol{w}_k, k)$=$\boldsymbol{w}_k$ and $\boldsymbol{h}(\boldsymbol{w}_k, k)$ is represented by the NN function $\boldsymbol{\Phi}(\boldsymbol{x}_k, \boldsymbol{w}_k)$. The EKF for NN weight optimization can be condensed to the following equations:

$$\boldsymbol{G}_k = \boldsymbol{P}_k \boldsymbol{H}_k^{\mathrm{T}} \left(\boldsymbol{H}_k \boldsymbol{P}_k \boldsymbol{H}_k^{\mathrm{T}} + \boldsymbol{R}_k \right)^{-1}, \tag{2.76}$$

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{G}_k \boldsymbol{\xi}_k, \tag{2.77}$$

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{G}_k \boldsymbol{H}_k) \boldsymbol{P}_k + \boldsymbol{Q}_k, \tag{2.78}$$

where $\boldsymbol{\xi}_k = \boldsymbol{y}_k - \boldsymbol{\Phi}(\boldsymbol{x}_k, \boldsymbol{w}_k)$ and $\boldsymbol{H}_k$ is now

$$\boldsymbol{H}_k = \left. \frac{\partial \boldsymbol{\Phi}(\boldsymbol{x}_k, \boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}=\hat{\boldsymbol{w}}_k}. \tag{2.79}$$

At each time instant k we choose an input-output pair $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$, calculate $\boldsymbol{\Phi}(\boldsymbol{x}_k, \hat{\boldsymbol{w}}_k)$, $\boldsymbol{H}_k$ and use the equations of the EKF (Eqs. 2.76-2.78) in order to obtain $\hat{\boldsymbol{w}}_{k+1}$.
It shall be noted that the LMA is just a particular case of the EKF for NNs under some simplifying assumptions [48]. Hence, for our purposes, the EKF is the method we choose, because, by varying the parameters of the EKF, the performance of the LMA should be, at least, achieved or even surpassed.

2.4.5. Batch-like extended Kalman filter

Kalman filtering is a technique made for sequential updating. In terms of optimizing weights of a NN this means training patterns are processed instance-by-instance by the EKF algorithm. This can be a clear disadvantage compared to other optimization algorithms, which take multiple training patterns into account for one single weight update. One might average the quantities over several training instances to construct $\boldsymbol{H}_k$ and $\boldsymbol{\xi}_k$ before applying the EKF equations, as it is comparably done in the (mini) batch gradient descent algorithm. However, for EKF training this is incorrect [49]. A more sophisticated method, the multi-stream extended Kalman filter (MSEKF) [49, 50], can circumvent this problem. It only requires slight modifications of the EKF. As for mini-batch training in the case of GD a set of n training instances I_k=$\{i_0, i_1, ..., i_n\}$ must be chosen at every iteration step k. Here, n is referred to as the number of streams. For each training instance i of iteration step k the quantities $\boldsymbol{H}_k^i$ and $\boldsymbol{\xi}_k^i$ are computed. By concatenating matrices $\boldsymbol{H}_k^i$ vertically matrix $\boldsymbol{H}_k$ is formed:

$$\boldsymbol{H}_k = \left((\boldsymbol{H}_k^{i_0})^T, (\boldsymbol{H}_k^{i_1})^T, ..., (\boldsymbol{H}_k^{i_n})^T\right)^T. \tag{2.80}$$

Similarly, we construct $\boldsymbol{\xi}_k$:

$$\boldsymbol{\xi}_k = \left((\boldsymbol{\xi}_k^{i_0})^T, (\boldsymbol{\xi}_k^{i_1})^T, ..., (\boldsymbol{\xi}_k^{i_n})^T\right)^T. \tag{2.81}$$

With this augmented matrix and vector $\boldsymbol{H}_k$ and $\boldsymbol{\xi}_k$, respectively, Eqs. 2.76-2.78 can be applied straightforwardly. The basic idea behind the MSEKF is to replace the original NN (with N outputs) with a set of n identical copies of the original NN. In each iteration every copy processes input data of one training instance and the n resulting outputs of size N are considered to be the output of a NN with $N \cdot n$ outputs.

There is another modification of the EKF for the purpose of weight optimization, namely the EKF with **forgetting factor** [51, 52]. The idea behind this method is that early optimization steps tend to be very inaccurate because the initial guess will most probably be widely separated from an acceptable local minimum [51]. The forgetting strategy reduces the influence of previous updates. As the optimization proceeds the algorithm will hopefully reach a point where the level of error is ac-

ceptable and in that case it is advantageous to make use of all available information [51]. Such an effect can be achieved by modifying Eq. 2.76-2.78 in the following way:

$$\lambda_k = \mu\lambda_{k-1} + 1 - \mu, \tag{2.82}$$

$$\boldsymbol{G}_k = \lambda_k^{-1}\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}}\left(\lambda_k^{-1}\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}} + \boldsymbol{R}_k\right)^{-1}, \tag{2.83}$$

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{G}_k\boldsymbol{\xi}_k, \tag{2.84}$$

$$\boldsymbol{P}_{k+1} = \lambda_k^{-1}(\boldsymbol{I} - \boldsymbol{G}_k\boldsymbol{H}_k)\boldsymbol{P}_k + \boldsymbol{Q}_k, \tag{2.85}$$

where $0 \leq \mu \leq 1$ and $0 < \lambda_0 < 1$. This means, we divide $\boldsymbol{P}_k$ at each iteration step k by λ_k. Even though the modifications in Eqs. 2.82-2.85 arise from a very distinct approach [51], we prove in Appendix A that this scheme can also be written in a different way, which we find to be more insightful:

$$\lambda_k = \mu\lambda_{k-1} + 1 - \mu, \tag{2.86}$$

$$\alpha_k = \alpha_{k-1}\lambda_k, \tag{2.87}$$

$$\boldsymbol{G}_k = \boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}}\left(\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}} + \alpha_k\boldsymbol{R}_k\right)^{-1}, \tag{2.88}$$

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{G}_k\boldsymbol{\xi}_k, \tag{2.89}$$

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{G}_k\boldsymbol{H}_k)\boldsymbol{P}_k + \alpha_k\boldsymbol{Q}_k, \tag{2.90}$$

where $\alpha_0 = 1$ and again, $0 \leq \mu \leq 1$ and $0 < \lambda_0 < 1$. Since $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ represent randomness in the optimization process and α_k decreases with increasing k, the forgetting factor EKF can be understood as an EKF with stepwise decreasing level of randomness. To run the iteration we have to choose values for $\boldsymbol{P}_0$, for all $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$, λ_0, and μ. In our applications we set all elements of $\boldsymbol{Q}_k$ to zero, $\boldsymbol{R}_k$ to the unity matrix, λ_0 to a value between 0.9 and 0.99, μ between 0.95 and 0.999 and $\boldsymbol{P}_0$ to the unity matrix multiplied by a value between 100 and 100000. We often applied the MSEKF, where the number of streams was mostly chosen to be between 4 and 16.

3. Neural network potential energy surfaces in chemistry

In the last chapter the basics of MLPs are discussed. The technique we use in this work, the Behler-Parrinello method [13], utilizes MLPs and has not been the first attempt of fitting a NN to the PES of a system of multiple atoms. An extensive summary of efforts in this field can be found in the work of Handley and Popelier [53] as well as Behler [6].
The application of NNs for approximating PESs is not a straightforward task because there are several issues which need to be considered before a NN can be used for this purpose. Among these, we find:

1. In the absence of external forces the NNP should be **invariant with respect to rotation or translation** of the whole system.

2. The NNP should be **invariant with respect to the order of atoms** of the same element.

3. The NNP should be **flexible regarding the number of atoms** the simulated system contains.

The first two points arise from the fact that a PES also shows these properties. A NNP without exhibiting these features could still work but it is unsatisfactory if very basic properties as rotational and translational invariance and symmetries regarding the order of the atoms are not preserved by the NNP. The third criterion is important, if we want to simulate systems of different size or if we want to perform simulations in the grand canonical ensemble but it is also not mandatory. Alternatively, a separate NNP can be trained for each system size. As we have already mentioned, there are several different approaches to creating NNPs. Even though these approaches differ in many details, there are two main and very essential distinctions. These distinctions concern the number of NNs used for the approximation and the type of the so-called symmetry functions (SF).

3.1. Structures of neural network potentials

3.1.1. Potentials using a single neural network

The most intuitive strategy is the representation of the PES by one single NN. That is

$$E_{\mathrm{NN}} = \Phi(\boldsymbol{w}, \boldsymbol{q}), \tag{3.1}$$

where $\boldsymbol{w}$ is the vector of all weight parameters, $\boldsymbol{q}$ is a d-dimensional vector of input coordinates determining all necessary information about the atomic positions of the configuration, which energy we want to describe by the NN and Φ is the NN function, $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}$. The simplest choice for the input coordinates, in the case of N_{atom} atoms, is to use the complete set of $3N_{\mathrm{atom}}$ Cartesian coordinates $\{x_i\}$. Such a NN is depicted in Fig. 3.1 for the example of one H_2O molecule. However, it is obvious that this approach does not provide a NN that is invariant with respect to the order of the hydrogen atoms or translation and rotation of the molecule. Of course, to some extent, the NNP can learn these properties, especially if the training instances are randomly varied with respect to the order of atoms of the same element and if the the system is randomly translated and rotated. Agrawal *et al.* investigated the dissociation of SiO_2 with the help of NNPs [54]. Instead of applying a special symmetrization procedure, they only used an input vector consisting of the three interatomic distances but they extended the training data set by copying the original configurations with exchanged Si atoms. The drawbacks are a higher demand of training examples and that the NNP will not acquire these properties in a perfect but only approximate manner. Thus, it gained acceptance to use SFs in order to transform the set of Cartesian input coordinates to a set of values which show invariance with respect to the order of atoms of the same element and with respect to translations and rotations of the whole system. We will cover the topic of SFs in Section 3.2. A more sophisticated technique is the method of using multiple NNs.

3.1.2. Atomic neural network potentials

For this technique a separation of the total energy,

$$E_{\mathrm{NN}} = \sum_{i=1}^{N_{\mathrm{atom}}} E_i = \sum_{i=1}^{N_{\mathrm{atom}}} \Phi^{(i)}(\boldsymbol{w}^{(i)}, \boldsymbol{q}^{(i)}), \tag{3.2}$$

is applied, where E_i is determined as the atomic energy. The atomic energy is just a means of reducing the complexity and increasing the flexibility of a NNP because in quantum mechanical calculations of the potential energy such a quantity is not defined. For many empirical potentials a summation of individual atomic energy

contributions is performed and thus atomic NNPs resemble empirical potentials on this point. The energy of atom i is represented by an own NN function $\Phi^{(i)}$ with arguments $\boldsymbol{q}^{(i)}$, a vector of input values computed as a function of the positions of a certain set of atoms depending on atom i as well as $\boldsymbol{w}^{(i)}$, a set of weights of the NN associated with atom i. Within the approach of Behler and Parrinello [13] all atoms of the same element are represented by an identical NN, having the same weight parameters and an input vector depending on the positions of all neighbors relative to atom i. Atomic NNPs have essential advantages over potentials using a single NN.

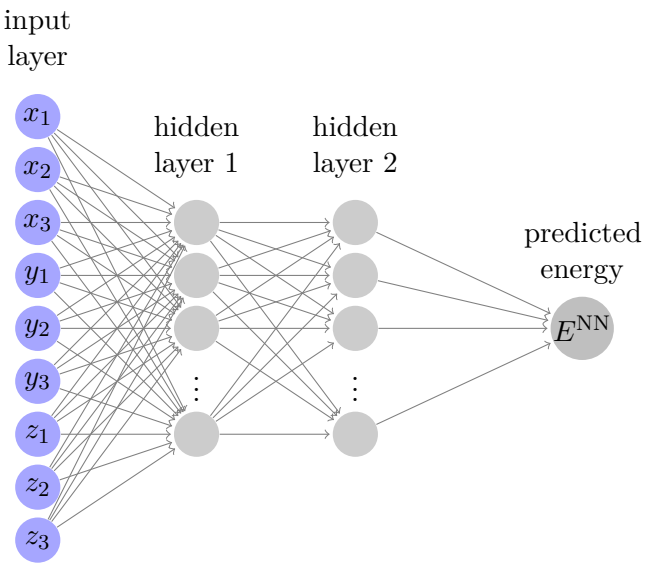


Figure 3.1.: Structure of a NN used for a fit to the energy of one water molecule based on Cartesian coordinates. 9 Cartesian coordinates of the H_2O molecule are forwarded to the input layer.

First of all, a single NN typically has a fixed number of input values. If each input value is computed by using the location of a certain atom or the locations of a certain number of atoms, it is not possible to apply the NNP consisting of a single NN to configurations with different numbers of atoms. It would be impractical to train multiple NNPs for different system sizes. The same argument is valid if we replace one atom of a certain element by another atom of a different element. Additionally, the complexity of a NNP using a single NN will grow rapidly with system size.
However, regardless if we use a single NN or multiple atomic NNs in order to satisfy the aforementioned invariance requirements, we need to employ SFs.

3.2. Symmetry functions

As mentioned previously, a NNP should not only approximate the reference PES but also satisfy some key features of a PES such as invariance with respect to rotations, translations and invariance with respect to the permutation of atoms or molecules

without changing the energy of the system [55]. When we consider, for instance, two water molecules (O_1, H_{11}, H_{12}), (O_2, H_{21}, H_{22}), we can write the PES as

$$E_0 = E(\vec{x}_{O_1}, \vec{x}_{H_{11}}, \vec{x}_{H_{12}}, \vec{x}_{O_2}, \vec{x}_{H_{21}}, \vec{x}_{H_{22}}). \tag{3.3}$$

The function value must be the same when we change the position vectors of two hydrogen atoms of the same molecule $\vec{x}_{H_{11}}$, $\vec{x}_{H_{12}}$:

$$E_0 = E(\vec{x}_{O_1}, \vec{x}_{H_{12}}, \vec{x}_{H_{11}}, \vec{x}_{O_2}, \vec{x}_{H_{21}}, \vec{x}_{H_{22}}). \tag{3.4}$$

Overall there are 8 different permutations (see table 3.1) which are energetically equivalent. One must be aware that this is only true, if the potential energy function distinguishes between bonded and unbonded atoms. Otherwise we will find 48 different permutations of the input vectors of the same energy (2 permutations for oxygen atoms and 24 for hydrogen atoms). If we use only interatomic distances, we can not expect to obtain a NNP which preserves invariance with respect to all energetically equivalent permutations. For demonstration purposes, we use the example of two rigid water molecules (8 permutations) to introduce the group of global SFs in the next section.

3.2.1. Global symmetry functions

Global SFs are just a means of transforming the $3N_{\text{atom}}$ Cartesian coordinates of a system containing N_{atom} atoms into a another set of coordinates. This new set shall satisfy the aforementioned invariance criteria and still give a precise description of the system. We call this transformation global SFs, because it does not focus on the local environment of single atoms but rather on the whole structure of the system. An interesting first approach was found by Gassner *et al.* In this work a NN was used to fit the three-body interaction energies in the system $H_2O - Al^{3+} - H_2O$ [56]. They used functions of interatomic distances of certain pairs of atoms as input values for the NN to obtain a NNP that is invariant with respect to energetically equivalent permutations of the order of the atoms. Here we will show how we can achieve that in the case of two rigid water molecules in a similar manner as Gassner *et al.*. First of all, we note that for rigid water molecules 6 interatomic distances would suffice to describe the geometry of the system, but we recommend to use more than 6 symmetrized input coordinates for the NN since the symmetrization can lead to a loss of information. We can choose from three different groups, that is one O-O distance, four O-H distances and four H-H distances. Each group shall be symmetrized separately to reduce the complexity. We will demonstrate the symmetrization for the group of O-H distances. Four symmetrized expressions $G^{(1)}_{\text{O-H}}$, $G^{(2)}_{\text{O-H}}$, $G^{(3)}_{\text{O-H}}$, $G^{(4)}_{\text{O-H}}$ as functions of the interatomic O-H distances $R_{O_1\text{-}H_{21}}$, $R_{O_1\text{-}H_{22}}$, $R_{O_2\text{-}H_{11}}$, $R_{O_2\text{-}H_{12}}$ shall be found.

We start by taking the absolute value of the sum or difference of two interatomic distances,

$$G^{(*)}_{\text{O-H}} = |R_{\text{O}_1\text{-H}_{21}} - R_{\text{O}_1\text{-H}_{22}}|\,.$$

Now we look at the same expression after interchanging atoms, H_{21} and H_{22}:

$$G^{(**)}_{\text{O-H}} = |R_{\text{O}_1\text{-H}_{22}} - R_{\text{O}_1\text{-H}_{21}}|\,.$$

Since we have taken the absolute value, this expression is already invariant with respect to this permutation. So we don't need to change our expression for $G^{(*)}_{\text{O-H}}$ so far. Another energetically equivalent permutation from Table 3.1 is

$$\text{O}_1 \longleftrightarrow \text{O}_2,\ \text{H}_{21} \longleftrightarrow \text{H}_{11},\ \text{H}_{22} \longleftrightarrow \text{H}_{12}. \tag{3.5}$$

By executing this permutation the value of $G^{(*)}_{\text{O-H}}$ becomes

$$G^{(***)}_{\text{O-H}} = |R_{\text{O}_2\text{-H}_{12}} - R_{\text{O}_2\text{-H}_{11}}|\,,$$

which is obviously not the same as $G^{(*)}_{\text{O-H}}$. To preserve the invariance of $G^{(*)}_{\text{O-H}}$ with respect to this permutation, we need to combine $G^{(*)}_{\text{O-H}}$ and $G^{(***)}_{\text{O-H}}$ as the sum or the absolute value of the difference of those.

$$\begin{aligned} G^{(1)}_{\text{O-H}} &= |R_{\text{O}_2\text{-H}_{12}} - R_{\text{O}_2\text{-H}_{11}}| + |R_{\text{O}_1\text{-H}_{21}} - R_{\text{O}_1\text{-H}_{22}}| \\ G^{(2)}_{\text{O-H}} &= \Big|\, |R_{\text{O}_2\text{-H}_{12}} - R_{\text{O}_2\text{-H}_{11}}| - |R_{\text{O}_1\text{-H}_{21}} - R_{\text{O}_1\text{-H}_{22}}| \,\Big|. \end{aligned}$$

Now this expression remains unchanged for all energetically equivalent permutations and we have thus simultaneously found two symmetrized functions. By applying exactly the same procedure, we can find a third and fourth expression starting with $|R_{\text{O}_1\text{-H}_{21}} + R_{\text{O}_1\text{-H}_{22}}|$. This gives us

$$\begin{aligned} G^{(3)}_{\text{O-H}} &= |R_{\text{O}_2\text{-H}_{12}} + R_{\text{O}_2\text{-H}_{11}}| + |R_{\text{O}_1\text{-H}_{21}} + R_{\text{O}_1\text{-H}_{22}}| \\ G^{(4)}_{\text{O-H}} &= \Big|\, |R_{\text{O}_2\text{-H}_{12}} + R_{\text{O}_2\text{-H}_{11}}| - |R_{\text{O}_1\text{-H}_{21}} + R_{\text{O}_1\text{-H}_{22}}| \,\Big|. \end{aligned}$$

For the case of H-H distances we could proceed in a similar way.
The problem with this approach is that the number of atoms must remain constant. If the system size is varying, the same NN can not be used. Additionally, it can be very complex to create symmetrized functions when there are many, say N_{atom}, atoms of the same element which can all be interchanged with each other without changing the energy. To be more precise, each position vector of these N_{atom} atoms must enter each symmetrized function via at least one interatomic distance. When one of the atoms is not used in a symmetrized function of this group, the function

1.	2.	3.	4.	5.	6.	7.	8.
O_1	O_1	O_1	O_1	O_2	O_2	O_2	O_2
H_{11}	H_{12}	H_{11}	H_{12}	H_{21}	H_{22}	H_{21}	H_{22}
H_{12}	H_{11}	H_{12}	H_{11}	H_{22}	H_{21}	H_{22}	H_{21}
O_2	O_2	O_2	O_2	O_1	O_1	O_1	O_1
H_{21}	H_{21}	H_{22}	H_{22}	H_{11}	H_{11}	H_{12}	H_{12}
H_{22}	H_{22}	H_{21}	H_{21}	H_{12}	H_{12}	H_{11}	H_{11}

Table 3.1.: Eight energetically equivalent permutations of the position vectors of the PES of two water molecules $E(\vec{x}_{O_1}, \vec{x}_{H_{11}}, \vec{x}_{H_{12}}, \vec{x}_{O_2}, \vec{x}_{H_{21}}, \vec{x}_{H_{22}})$. If the functional form of the PES did not distinguish between bonded and unbonded atoms, we could even interchange each arbitrary pair of atoms of the same element. In that case 48 permutations of the same energy could be found.

can, per construction, not be invariant with respect to permutations in which this atom changes place with another atom being used.
A similar effect could also be achieved by another kind of global symmetrization. Raff *et al.* [57] described a Si_5 system by using 4 Si-Si bond distances, 3 Si-Si-Si angles and 2 dihedral angles. To prevent altering energies when two Si atoms are exchanged, the four bond distances are listed in increasing order of their deviation from the Si-Si equilibrium bond length. The order of these bonds also determines the input vector components for the bond angles and dihedral angles [57]. This procedure ensures that the input vector remains unaffected when two Si atoms are exchanged.
The SFs we have presented so far can be efficiently used for systems containing a small and fixed number of atoms and they are the ideal choice for NNPs using a single NN. If we use SFs that scan the chemical environment of a particular atom, the approach of atomic NNPs of Eq. 3.2 seems more reasonable.

3.2.2. Local symmetry functions

As opposed to global SFs, the intention behind local SFs[5] is to separately describe the chemical environment of each single atom, called the central atom. Therefore, for a system of N_{atom} atoms, we need N_{atom} sets of SFs. Even though we could forward all values of all sets of SFs to a single NN in a particular order, we would still face the problem that exchanging two atoms would lead to a different input vector and, consequently, to another output. Additionally, this approach using only one NN could still not handle a variable number of atoms. Only if we combine the concept of local SFs with that of atomic NNPs, we obtain a powerful method which can solve these problems. In order to obtain a NNP, which is invariant with respect to energetically equivalent permutations, the set of SFs must be calculated in an identical way for atoms of the same element.

[5]Local SFs will sometimes also be referred to as atom-centered SFs.

Two different approaches of atom-centered SFs have been formulated. On the one hand, there are SFs that always process information from a fixed number of neighbors of the central atom but the number of SFs itself is varying, depending on the environment of the central atom. In other words, each SF processes only information from a single group of atoms, e.g. a pair, a triple, a quartet or groups of bigger size. If the number of possible groups present in the environment of the central atom varies, then, also the number of SFs does. On the other hand, SFs which incorporate and aggregate information from a variable number of groups were developed and so the number of SFs can be held constant. The latter type of atom-centered SFs, which we use in this work, was devised by Behler and Parrinello [13].

Local single group SFs

The first SFs of this type were developed by Hobday *et al.* [58] and in a similar way reused and enhanced by Bholoa *et al.* [59]. The implementation of Hobday *et al.* processed information from all triples the central atom could form with its first neighbors, e.g. the bond lengths, the bond angle and discrete information about the type of elements of the involved atoms in this triple. Similarly, Bholoa *et al.* described quartets of atoms $\{i, j, k, l\}$, where a quartet was formed if each atom was a first neighbor of another atom of the quartet, starting with the central atom i and $i{\neq}j{\neq}k{\neq}l$. Each atom was attributed to its own (atomic) NN and the size of the input vector of these NNs was determined by the number of available triples or quartets including atom i. Such an input vector consisted of a subset of values and each value of this subset was calculated by a function of the relative atomic coordinates of the respective triple or quartet. The architecture of an atomic NN employed, together with this type of symmetry functions as well as a more detailed description of its architecture, can be found in Fig. 3.2. It is important to emphasize that this approach can only work if the underlying atomic NNs are of variable size. A way of avoiding this is the usage of local accumulative SFs, which we cover in the next section.

Local accumulative SFs

Behler and Parrinello used the concept of atomic NNPs and supplemented it by the development of certain SFs which allow for atomic NNs of constant size, even if the number of atoms in the system varies or if the environment of the central atom essentially changes. Just like local single group SFs, this class of SFs derives a value from the relative coordinates of a group (pairs, triples, quartets,...) of atoms involving the central atom. However, the difference between them is that local accumulative SFs sum these values over multiple groups of this kind for one central atom. To clarify this, we give an example. A local single group SF is a function, $f_2(R_{ij})$, of

the distance between central atom i and neighboring atom j, R_{ij}. By a summation over all neighbors of atom i,

$$G_i^{(2)} = \sum_{j \neq i}^{N_{\text{atom}}} f_2(R_{ij}), \tag{3.6}$$

we obtain a local accumulative SF. Similarly, this works for triples

$$G_i^{(3)} = \sum_{j \neq i}^{N_{\text{atom}}} \sum_{\substack{k \neq i \\ k > j}}^{N_{\text{atom}}} f_3(R_{ij}, R_{ik}, R_{jk}). \tag{3.7}$$

In order to keep the NN approach tractable, Behler and Parrinello employed a cutoff radius that limits the information which is to be collected about the environment of the central atom. Neighbors of this atom with a relative distance larger than the cutoff do not influence the values of the SFs. In the following chapter we will extensively describe the NNP Behler and Parrinello presented in 2007 [13].

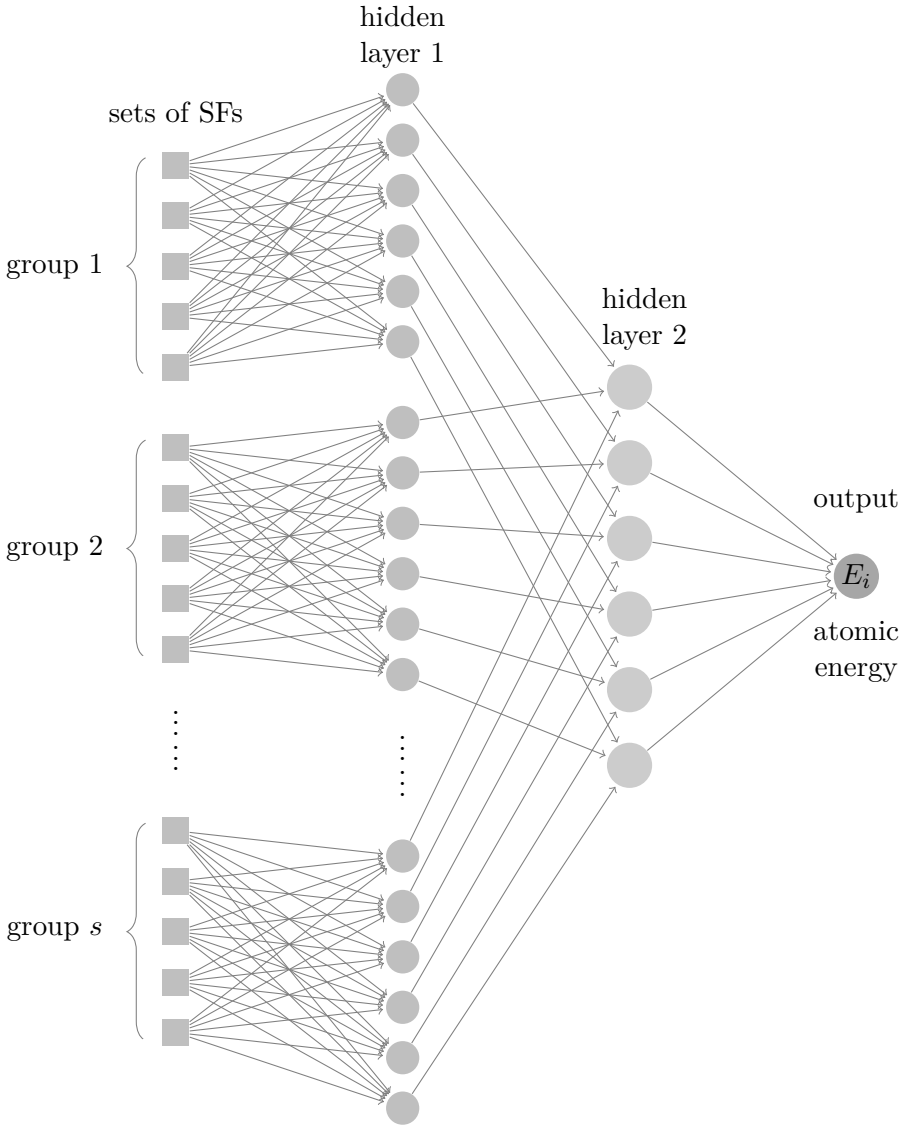


Figure 3.2.: Architecture of the atomic NN for atom i used in the work of Bholoa *et al.* as well as Hobday *et al.* [58, 59]. From each group a number of function values is calculated (in this NN: 5). These s sets of values are forwarded to the same number of sublayers constituting the first hidden layer, where s is the number of quartets including atom i. Thus, the size of the input layer and first hidden layer is not constant, which is a non-standard approach in NN design. The number of neurons of each sublayer of the first hidden layer equals the number of neurons of the second hidden layer. Furthermore, only the kth neuron of each sublayer is connected with the kth neuron of the second hidden layer. In the work of Bholoa *et al.* [59] the weights between the first and second hidden layer are not determined by a training procedure but by evaluation of a screening function, which serves to classify the influence of each quartet on the atomic energy.

4. Neural network potentials of Behler and Parrinello

Like in many empirical potentials, the total energy E_{tot}^{NN} in this approach is constructed as a sum of atomic energies ϵ_i [6] :

$$E_{\text{tot}}^{\text{NN}} = \sum_{i=1}^{N_{\text{atom}}} \epsilon_i^{\text{NN}}. \tag{4.1}$$

Therefore, this method can be classified as atomic NNP, which we have described in section 3.1.2. The input vector of an atomic NN is a vector of values calculated by employing local accumulative SFs. The nth SF of central atom i, $G_{i,n}$, for a system consisting of N_{atom} atoms can be written as a function of $\boldsymbol{X}$, the vector of all $3N_{\text{atom}}$ Cartesian coordinates,

$$G_{i,n} = G_{i,n}(\boldsymbol{X}). \tag{4.2}$$

Furthermore, the atomic NN of atom i is given by

$$\epsilon_i^{\text{NN}} = \Phi^{(Z_i)}(\boldsymbol{w}^{(Z_i)}, \boldsymbol{G_i}), \tag{4.3}$$

where $\Phi^{(Z_i)}$ is the NN function assigned to atom i, with its vector of weight parameters $\boldsymbol{w}^{(Z_i)}$ and $\boldsymbol{G_i}$ is the vector of SFs serving as NN input. The notations $\Phi^{(Z_i)}$ and $\boldsymbol{w}^{(Z_i)}$ indicate that the atomic NN function (its architecture and weight parameters) depends on atom i. To be more precise, it depends on the element of atom i, Z_i. The same is true for the number and type of SFs in the SF set for atom i, $\{G_{i,n}\}$. That means the composition of the central atom's set of SFs depends on its element. The notation we use here, $G_{i,n}$, does not take into account this dependence of the SF on the element of the associated central atom in order to emphasize the index of the central atom the SF is computed for. Exactly these dependences on the element combined with Eq. 4.1 cause the invariance of the NNP w.r.t. the exchange of two atoms of the same element. In case of such an exchange, only the two summands of the respective atoms in the sum of Eq. 4.1 change their places, but the summands itself remain unchanged. To also achieve translational and rotational invariance, the applied SFs must be constructed appropriately. The relations between the quantities in Eq. 4.1-4.3 are depicted in Fig. 4.1. In the following, we will discuss the SFs Behler

and Parrinello developed [13] as well as additional SFs of Behler [60]. We will also suggest two new types of local accumulative SFs.

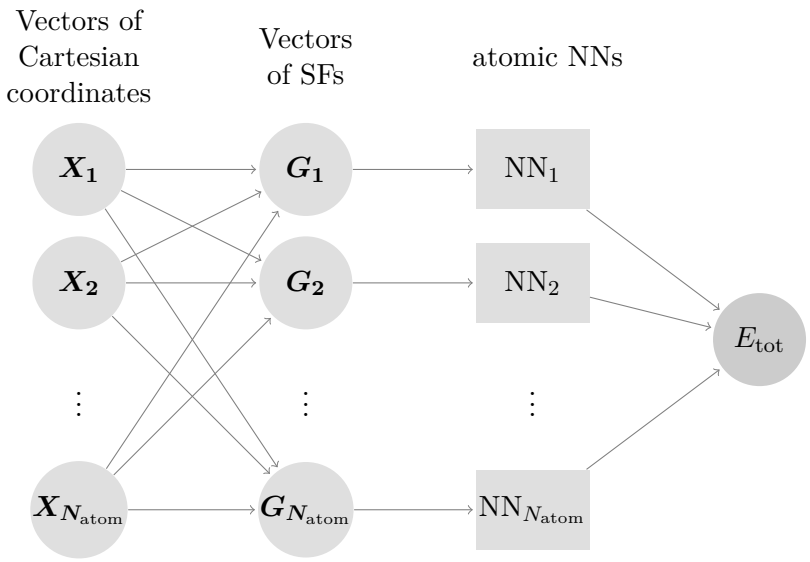


Figure 4.1.: NNP of Behler and Parrinello [6]. The links do not represent weighted connections but indicate that the object on the right-hand side of each link is derived from the object on the left-hand side. From N_{atom} three-dimensional position vectors, $\{\boldsymbol{X_i}\}$, another set of N_{atom} SF vectors, $\{\boldsymbol{G_i}\}$, is derived. These SF vectors are the input for the N_{atom} atomic NNs.

4.1. Behler-Parrinello symmetry functions

For the NNP of Behler and Parrinello the environment of each single atom needs to be described separately. Depending on the specific set of SFs, this can lead to substantial amount of computation time. To reduce the computational complexity a cutoff function is used. Atoms beyond this cutoff distance will not have any impact on the values of the SFs.

4.1.1. Cutoff functions

In the approach of Behler and Parrinello the calculation of the derivatives of each SF with respect to the atomic Cartesian coordinates is required. This is necessary for the calculation of forces. A hard cutoff function (step function) implies a discontinuous derivative for forces at the cutoff radius. Such discontinuities can have negative

effects on the accuracy and energy conservation of a MD simulation [61]. Thus, two different cutoff functions were proposed [13, 60]:

$$f_{\mathrm{c},1} = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_\mathrm{c}}\right) + 1\right], & R_{ij} \leq R_\mathrm{c} \\ 0, & R_{ij} > R_\mathrm{c} \end{cases} \tag{4.4}$$

$$f_{\mathrm{c},2} = \begin{cases} \tanh^3\left[1 - \frac{R_{ij}}{R_c}\right], & R_{ij} \leq R_\mathrm{c} \\ 0, & R_{ij} > R_\mathrm{c} \end{cases}, \tag{4.5}$$

where R_{ij} is the distance between atom i and atom j and R_c is the cutoff radius. As can be seen in Fig. 4.2, $f_{\mathrm{c},1}$, $f_{\mathrm{c},2}$ and their first derivatives have zero value at the cutoff distance R_c.

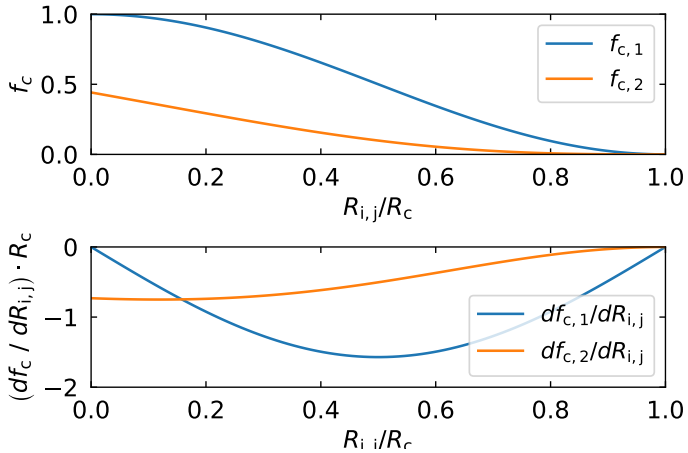


Figure 4.2.: Cutoff functions $f_{\mathrm{c},1}$ and $f_{\mathrm{c},2}$ from Eqs. 4.4 and 4.5 and their derivatives with respect to the interatomic distance $R_{i,j}$ between atom i and j.

If we sum over all pairs of atoms, cutoff functions can also be used as stand-alone SFs:

$$G_i^{(\mathrm{cut})} = \sum_{j \neq i}^{N_\mathrm{atom}} f_\mathrm{c}(R_{ij}). \tag{4.6}$$

The main application of cutoff functions is yet to ensure a smooth decay of all SFs to zero at the cutoff radius. We will show how this is achieved within the next sections, where we describe typical SFs.

4.1.2. Radial symmetry functions

The cutoff functions we have introduced in the last section are monotonically decreasing functions. That means, closely neighboring atoms do have a bigger impact

on the SF value than atoms which are farther apart. However, there should also be SFs which can scan different regions of the vicinity of the central atom. In this way, additional information about the radial arrangement can be collected:

$$G_i^{(\text{radial})} = \sum_{j \neq i}^{N_{\text{atom}}} g_{i,j}^{(\text{radial})} = \sum_{j \neq i}^{N_{\text{atom}}} \exp^{-\eta(R_{ij}-R_s)^2} f_{\text{c}}(R_{ij}). \tag{4.7}$$

The radial SF $G_i^{(\text{radial})}$ is the sum of the products of a cutoff function and Gaussians with parameters η and R_s. Whereas η controls the width of the Gaussians, R_s determines how much the center of the Gaussian is shifted (Fig. 4.3). By multiplying each Gaussian by $f_{\text{c}}(R_{ij})$ we achieve

$$g_{i,j}^{(\text{radial})}(R_{\text{c}}) = 0 \tag{4.8}$$

$$\mathrm{d}g_{i,j}^{(\text{radial})}(R_{ij}) \Big/ \mathrm{d}R_{ij} \Big|_{R_{ij}=R_{\text{c}}} = 0. \tag{4.9}$$

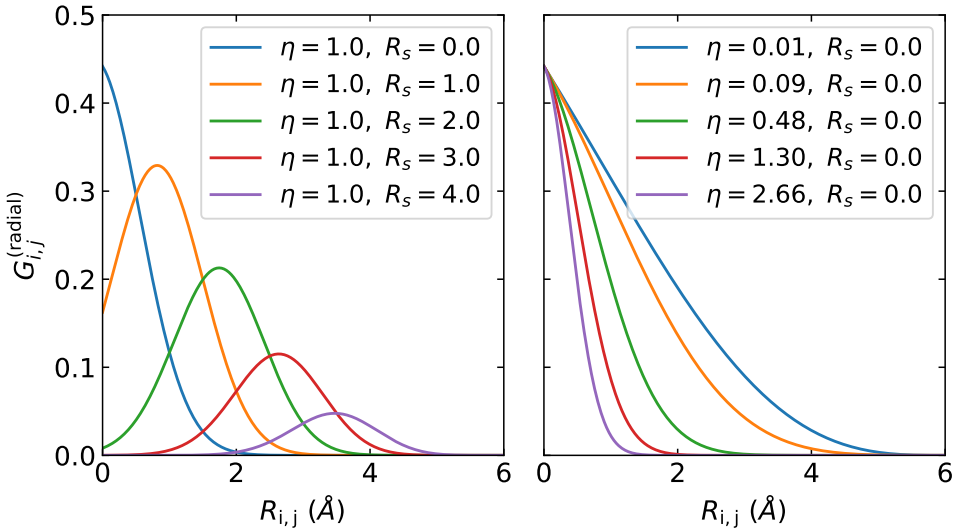


Figure 4.3.: Single summand $g_{i,j}^{(\text{radial})}$ of the radial Symmetry Function $G_i^{(\text{radial})}$ for different paramters η (left), R_s (right) and $R_{\text{c}} = 6.0$Å for central atom i and neighboring atom j.

4.1.3. Angular symmetry functions

So far we have only treated SFs (cutoff and radial functions) which solely process pairwise information of the central atom i and an arbitrary neighboring atom $j \neq i$. To accurately describe the local structure around atom i, we also have to consider information from atom triplets (i, j, k), where $i{\neq}j{\neq}k$. In general, to gather all available information from the system by atom-centered SFs, we would need to use a very large cutoff radius R_{c} and SFs processing the relative positions from atom-groups of all sizes, that means from pairs, triplets, quartets, quintets, etc. However,

in many applications it might be sufficient to stop after SFs for triplets and to choose a suitable cutoff in order to reach a compromise between computational costs and accuracy. Behler and Parrinello introduced the following angular SF [13]:

$$\begin{aligned} G_i^{(\mathrm{ang},3)} &= \sum_{j\neq i}^{N_\mathrm{atom}} \sum_{\substack{k\neq i \\ k>j}}^{N_\mathrm{atom}} g_{i,j,k}^{(\mathrm{ang},3)} = \\ &= 2^{1-\zeta} \sum_{j\neq i}^{N_\mathrm{atom}} \sum_{\substack{k\neq i \\ k>j}}^{N_\mathrm{atom}} \Big[(1+\lambda\cdot\cos\theta_{ijk})^\zeta \, e^{-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)} \\ &\quad \cdot f_\mathrm{c}(R_{ij}) f_\mathrm{c}(R_{ik}) f_\mathrm{c}(R_{jk}) \Big]. \end{aligned} \tag{4.10}$$

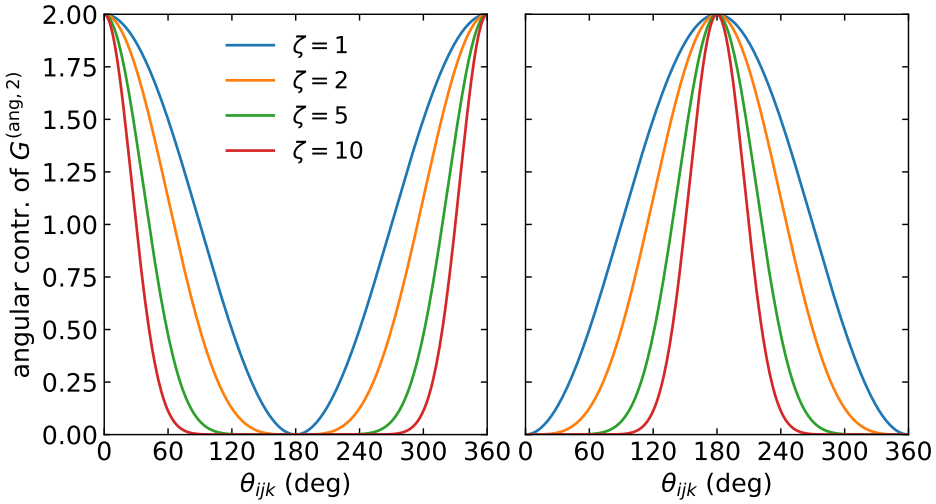


Figure 4.4.: Angular contribution of the angular SFs $G^{(\mathrm{ang},2)}$ and $G^{(\mathrm{ang},3)}$, $2^{1-\zeta}\,(1+\lambda\cdot\cos\theta_{ijk})^\zeta$, for $\lambda = 1$ (left) and $\lambda = -1$ (right).

Behler also described a different version of this function [60]:

$$\begin{aligned} G_i^{(\mathrm{ang},2)} &= \sum_{j\neq i}^{N_\mathrm{atom}} \sum_{\substack{k\neq i \\ k>j}}^{N_\mathrm{atom}} g_{i,j,k}^{(\mathrm{ang},2)} \\ &= 2^{1-\zeta} \sum_{j\neq i}^{N_\mathrm{atom}} \sum_{\substack{k\neq i \\ k>j}}^{N_\mathrm{atom}} \Big[(1+\lambda\cdot\cos\theta_{ijk})^\zeta \, e^{-\eta(R_{ij}^2+R_{ik}^2)} f_\mathrm{c}(R_{ij}) f_\mathrm{c}(R_{ik}) \Big], \end{aligned} \tag{4.11}$$

where the interatomic distance between the two neighbors of atom i is not taken into account. The angular contribution to $G^{(\mathrm{ang},2)}$ and $G^{(\mathrm{ang},3)}$ is shown in Fig. 4.4.

A third form of angular SFs was used by Smith *et al.* [62],

$$G_i^{(\mathrm{ang,mod})} = \sum_{j\neq i}^{N_{\mathrm{atom}}} \sum_{\substack{k\neq i \\ k>j}}^{N_{\mathrm{atom}}} g_{i,j,k}^{(\mathrm{ang,mod})} = \\ 2^{1-\zeta} \sum_{j\neq i}^{N_{\mathrm{atom}}} \sum_{\substack{k\neq i \\ k>j}}^{N_{\mathrm{atom}}} \Bigg[(1+\cos(\theta_{ijk}-\theta_{\mathrm{s}}))^{\zeta} \\ \cdot \exp\left(-\eta \left[\frac{R_{ij}+R_{ik}}{2} - R_{\mathrm{s}}\right]^2\right) f_{\mathrm{c}}(R_{ij}) f_{\mathrm{c}}(R_{ik}) \Bigg]. \tag{4.12}$$

As opposed to the angular SFs of Behler and Parrinello, this version allows for more individual probing of the angular environment by applying parameter θ_{s} and R_{s} serves the same purpose for the radial part [62]. Which of these angular SFs above shall be applied depends on the specific problem.

4.1.4. Legendrian symmetry functions

The drawback of the radial SF $G^{(\mathrm{rad})}$ of Section 4.1.2 is that a set of functions with specific parameters has to be chosen and if these parameters are picked improperly, the accuracy of the NNP is certainly decreased. For this reason we developed a new type of SF that only requires the definition of a cutoff radius. The main idea is that any function $f(x)$ on the intervall $[-1,1]$ can be written as

$$f(x) = \sum_{n=0}^{\infty} a_n P_n(x), \tag{4.13}$$

where P_n is the nth Legendre polynomial with its respective coefficient a_n [63]. The coefficients $\{a_i\}$ can be calculated by

$$a_m = \frac{2m+1}{2} \int_{-1}^{+1} P_m(x) f(x)\, \mathrm{d}x. \tag{4.14}$$

In a similar way, we want to find the coefficients for the distribution of atoms in the neighborhood of the central atom i. Therefore, we assume the following function $f_i(R)$:

$$f_i(R) = \sum_{j\neq i}^{N_{\mathrm{atom}}} f_c(R)\delta(R_{ij}-R). \tag{4.15}$$

The sum in Eq. 4.15 is to be taken over all neighbors of atom i and for a function $h(R)$, $\delta(R_{ij}-R)$ can be defined as

$$\int_0^{R_c} h(R)\delta(R_{ij}-R)\, \mathrm{d}R = h(R_{ij}), \tag{4.16}$$

for $0 \leq R_{ij} \leq R_c$. This type of SF is then defined as

$$G_i^{(Leg,m)} = \sum_{j \neq i}^{N_{\text{atom}}} \int_0^{R_c} f_c(R)\delta(R_{ij} - R) \cdot \left[P_m\left(D(R)\right) + 1\right] \, \mathrm{d}R, \tag{4.17}$$

where $D(R) = 2(R/R_c - 0.5)$. This is necessary to rescale the variable R to the domain $[-1, 1]$ the Legendre polynomials are defined on. We ignored the factor $(2m+1)/2$ in front of the integral (compare Eq. 4.14) because in the NNP algorithm it is standard to rescale all SF values before they are forwarded to the NN and, additionally, we shift the values of the Legendre polynomials to positive numbers. By considering Eq. 4.16, we obtain

$$G_i^{(Leg,m)} = \sum_{j \neq i}^{N_{\text{atom}}} g_{i,j}^{(Leg,m)} = \sum_{j \neq i}^{N_{\text{atom}}} f_c(R_{ij}) \left(P_m(D_{ij}) + 1\right). \tag{4.18}$$

The cutoff function does not only allow for the requirements of Eqs. 4.8 and 4.9, but also avoids an overemphasis on regions farther apart from the central atom, because the number of expected atoms for a homogeneous distribution scales with R^2. In order to compensate for this scaling, each summand of Eq. 4.18 could be multiplied by $1/R_{ij}^2$. However, we do not apply this compensation. Fig. 4.5 shows some Legendrian SFs for $0 \leq m \leq 7$. Besides this type, we also developed a second SF type to collect information from atom triples.

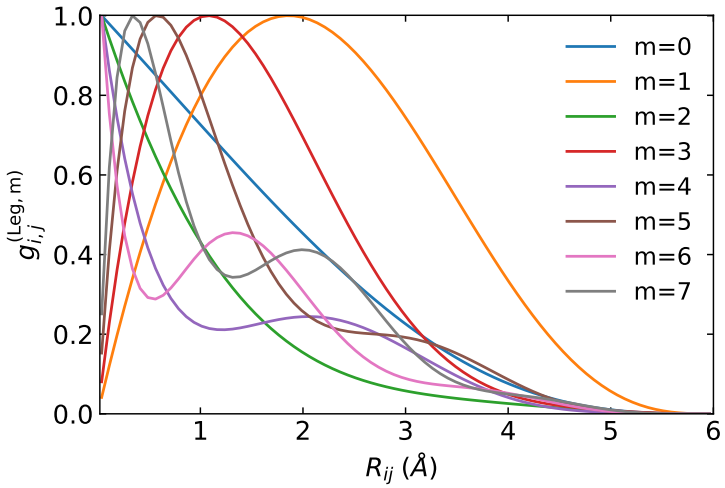


Figure 4.5.: The first 8 Legendrian SFs for cutoff function $f_{c,2}$ (see eq. 4.5) and cutoff radius $R_c = 6$Å. Each curve is normalized by its maximum.

4.1.5. Multi-radial symmetry functions

In order to get rid of one parameter in the angular SFs of Eqs. 4.10 and 4.11 we tried to simplify the concept of these. Based on the radial SF of Section 4.1.2, we defined

$$G_i^{(mr,3)} = \sum_{j\neq i}^{N_{\text{atom}}} \sum_{\substack{k\neq i \\ k>j}}^{N_{\text{atom}}} g_{i,j,k}^{(mr,3)} \tag{4.19}$$

$$= \sum_{j\neq i}^{N_{\text{atom}}} \sum_{\substack{k\neq i \\ k>j}}^{N_{\text{atom}}} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}) e^{-\eta(R_{ij}+R_{ik}+R_{jk}-R_s)}. \tag{4.20}$$

A potential advantage is that this type of SF can also be generalized to more than three atoms. For example, quartets of atoms can be described by 6 interatomic distances. In Fig. 4.6 we illustrate how the central quantity of this SF, namely $R_{ij} + R_{ik} + R_{jk}$, is distributed, using the example of SPC/F water.

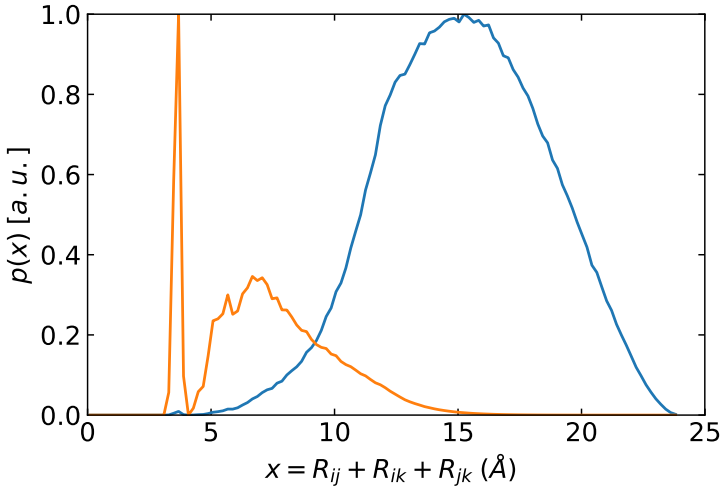


Figure 4.6.: Distribution of the magnitude of the exponent of the multi-radial SF for liquid-vapor water interface configurations and its impact on the SF value. The blue curve shows the probability distribution $p(x)$ of the sum $x = R_{ij}+R_{ik}+R_{jk}$ for three arbitrary atoms i, j, k averaged over 10 slab configurations of SPC/F water, each consisting of 216 water molecules. The orange curve represents $p(x) \cdot \exp\{-\eta\,(x - R_s)\}$ for $\eta = 0.45\text{Å}^{-1}$ and $R_s = 0\text{Å}$.

4.1.6. Element dependency of symmetry function

Many systems are containing atoms of more than one element. In this case it is very important that SFs distinguish between atoms of different elements. So far we have

not taken this fact into account in our description of symmetry functions. A typical SF collecting pairwise information has been formulated as follow:

$$G_i^{(2)} = \sum_{j \neq i}^{N_{\text{atom}}} f_2(R_{ij}). \tag{4.21}$$

When the system under consideration contains atoms of different elements, each symmetry function can be restricted to a certain element:

$$G_i^{(2)}\{\zeta\} = \sum_{j \neq i}^{N_{\text{atom}}} f_2(R_{ij}) \delta_{Z_j}^{\zeta}, \tag{4.22}$$

where Z_j is the element of atom j, ζ is the specific element $G_i^{(2)}\{\zeta\}$ is collecting information about and the Kronecker delta δ_i^j is defined as

$$\delta_i^j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \tag{4.23}$$

For SFs accumulating information of atom triplets, $G_i^{(3)}$, we modify the general form of Eq. 3.7 to

$$G_i^{(3)}\{\zeta, \xi\} = \sum_{j \neq i}^{N_{\text{atom}}} \sum_{\substack{k \neq i \\ k > j}}^{N_{\text{atom}}} f_3(R_{ij}, R_{ik}, R_{jk})(\delta_{Z_j}^{\zeta} \delta_{Z_k}^{\xi} + \delta_{Z_j}^{\xi} \delta_{Z_k}^{\zeta} - \delta_{Z_j}^{\zeta} \delta_{Z_k}^{\xi} \delta_{\zeta}^{\xi}). \tag{4.24}$$

The expression $\nu(\zeta, \xi, Z_j, Z_k) = (\delta_{Z_j}^{\zeta} \delta_{Z_k}^{\xi} + \delta_{Z_j}^{\xi} \delta_{Z_k}^{\zeta} - \delta_{Z_j}^{\zeta} \delta_{Z_k}^{\xi} \delta_{\zeta}^{\xi})$ gives

$$\nu(\zeta, \xi, Z_j, Z_k) = \begin{cases} 1, & (Z_j = \zeta \wedge Z_k = \xi) \vee (Z_j = \xi \wedge Z_k = \zeta) \\ 0, & \text{else} \end{cases}, \tag{4.25}$$

All SFs we use in this work shall be understood as element-dependent. For a system containing only water molecules, a SF $G_i^{(3)}$ can be defined as $G_i^{(3)}\{\text{H}, \text{H}\}$, $G_i^{(3)}\{\text{O}, \text{H}\}$ or $G_i^{(3)}\{\text{O}, \text{O}\}$, where H and O represent hydrogen and oxygen atoms, respectively.

4.1.7. Extrapolation warnings

The vector of SFs $\boldsymbol{G_i}$ is kind of a fingerprint of the chemical environment of central atom i. For central atoms of element ζ, the SF vector $\boldsymbol{G_i}(\zeta)$ contains a certain number of SFs of fixed types. A training set for a NNP contains a limited number of atoms and, therefore, also a limited number of SF vectors. Each of these SF vectors can be seen as a point in a multidimensional space. During a NNP simulation there naturally occur new SF vectors. If the distance from one of these new SF vectors is very large to the nearest SF vectors of the training set, the prediction of the

NNP may be poor. In such a case the NNP could issue a warning to indicate that the probability of a poor NNP prediction is higher than normal. Unfortunately, the procedure of computing the distance of a point to many thousands of other points in a multidimensional space is computationally quite expensive. Moreover, it is unclear what maximum distance to other points should be acceptable.
A related problem, which can be handled more effectively, is extrapolation. As indicated in Figs. 6.2, 6.3 and 9.4, NNs show poor performance when it comes to extrapolation. If at least a single component of a SF vector during a NNP simulation is greater or smaller than the maximum or minimum value of the same component obtained from the training set, the NNP prediction is an extrapolation which is clearly undesired. The algorithm we use for NNP simulations continuously checks if one of the SF vector components is outside the interval between the minimum and maximum value occurred in the training set. In case this happens, a warning is issued during the simulation. If the number of warnings per timestep reaches a high value, the training set should definitely be enlarged by configurations which eliminate these so-called extrapolation errors. However, one should be aware of the fact, that a NNP simulation without extrapolation warnings is not automatically reliable because the issue of extrapolation is only a partial aspect.

4.2. Training

As we have already stated in Eq. 4.1, the total energy of a configuration consisting of N_{atom} atoms is constructed as the sum of atomic energies

$$E^{\text{NN}} = \sum_{i=1}^{N_{\text{atom}}} \epsilon_i^{\text{NN}} = \sum_{i=1}^{N_{\text{atom}}} \Phi^{(Z_i)} \left(\boldsymbol{w}^{(Z_i)}, \boldsymbol{G_i}(\boldsymbol{X}) \right), \tag{4.26}$$

where $\boldsymbol{X}$ is the vector of all $3N_{\text{atom}}$ Cartesian coordinates. The architecture and the weight vector of each atomic NN associated to atom i depend on the element of atom i, Z_i. The output of the NNP of Behler and Parrinello is thus the total energy of a given configuration composed of all atomic energies. Since the underlying PES we want to approximate usually does not provide atomic energies but only the total energy, we cannot individually fit the atomic NN functions $\Phi^{(Z_i)}$. Besides the reference total energy of configuration s, E_s^{ref}, also forces of its atoms are available. Therefore, not only the total energy but also its derivative w.r.t. the components of the atomic position vectors, $F_{\alpha,s}^{\text{ref}} = -\,\partial E_s^{\text{ref}} \big/ \partial X_\alpha$, can be used for the training of the NNP, where X_α is the αth component of $\boldsymbol{X}$. The sth configuration of the training set provides

$$E_s^{\text{ref}}, F_{1,s}^{\text{ref}}, F_{2,s}^{\text{ref}}, ..., F_{3\cdot N_{\text{atom},s}}^{\text{ref}}, \tag{4.27}$$

where $N_{\mathrm{atom},s}$ is the number of atoms in configuration s. The error function can then be defined as

$$\Gamma = \sum_{s=1}^{N_{\mathrm{config}}} \left[\left(E_s^{\mathrm{ref}} - E_s^{\mathrm{NN}} \right)^2 + \beta \sum_{j=1}^{3 \cdot N_{\mathrm{atom},s}} \left(F_{j,s}^{\mathrm{ref}} - F_{j,s}^{\mathrm{NN}} \right)^2 \right]. \tag{4.28}$$

The error function is composed of two different types of quantities, energies and forces. This makes the optimization procedure more complex, because the ratio

$$\Gamma = \frac{\sum_{s=1}^{N_{\mathrm{config}}} \left(E_s^{\mathrm{ref}} - E_s^{\mathrm{NN}} \right)^2}{\sum_{s=1}^{N_{\mathrm{config}}} \sum_{j=1}^{3N_{\mathrm{atom},s}} \left(F_{j,s}^{\mathrm{ref}} - F_{j,s}^{\mathrm{NN}} \right)^2} \tag{4.29}$$

influences the relative importance of energies and forces during the optimization. First of all, the number of forces will naturally be higher than the number of configurations and, additionally, the ratio depends on the choice of units. A bad choice can lead to very unsatisfactory results. For this reason, the parameter β modifies the ratio of Eq. 4.28. The adjustment of β can, for instance, be conducted on a trial-and-error basis.

During the training we try to optimize all weight parameters of the NNP. In the beginning of this chapter we have already discussed that, due to symmetry aspects, atoms of the same element are represented by identical atomic NNs. This means that we do only have one weight vector $\boldsymbol{w}^{(\mathrm{Z}_i)}$ per element Z_i. Overall, the vector of parameters $\boldsymbol{w}$, which is to be optimized during the training of the NNP, can be constructed as

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}^{(\mathrm{Z_1})} \\ \boldsymbol{w}^{(\mathrm{Z_2})} \\ \vdots \\ \boldsymbol{w}^{(\mathrm{Z}_{N_{\mathrm{el}}})} \end{bmatrix}, \tag{4.30}$$

where N_{el} is the number of elements. The weight parameters within one weight vector $\boldsymbol{w}^{(\mathrm{Z}_i)}$ can, for instance, be arranged as

$$\boldsymbol{w}^{(\mathrm{Z}_i)} = [w_{01}^0, w_{02}^0, \ldots, w_{0m^{(1)}}^0, w_{11}^0, \ldots, w_{1m^{(1)}}^0, \ldots, w_{m^{(0)}m^{(1)}}^0, w_{01}^1, \ldots, w_{m^{(L)}m^{(L+1)}}^L]^{\mathrm{T}}, \tag{4.31}$$

where w_{ij}^l is the weight connecting the ith node of layer l with the jth node of layer $l+1$ and $m^{(l)}$ is the number of nodes of layer l (see Chapter 2). It shall be noted that for the atomic NNs under consideration $m^{(L+1)}$ is always 1. When a gradient-based optimization algorithm is used, we need the derivatives $\partial E_s^{\mathrm{NN}} \big/ \partial \boldsymbol{w}^{(\zeta)}$ and $\partial F_{j,s}^{\mathrm{NN}} \big/ \partial \boldsymbol{w}^{(\zeta)}$ for all j, s and ζ. For the sake of simplicity, we will suppress index s in what follows.

4.2.1. Weight derivatives of energies

The gradient we need is

$$\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}} = \left[\left(\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}^{(\boldsymbol{Z_1})}} \right)^{\mathrm{T}}, \left(\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}^{(\boldsymbol{Z_2})}} \right)^{\mathrm{T}}, \ldots, \left(\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}^{\boldsymbol{Z}_{N_{\mathrm{el}}}}} \right)^{\mathrm{T}} \right]^{\mathrm{T}} \tag{4.32}$$

and from Eq. 4.26 we get

$$\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}^{(\zeta)}} = \frac{\partial \sum_{i=1}^{N_{\mathrm{atom}}} \Phi^{(Z_i)} \left(\boldsymbol{G_i}(\boldsymbol{X}), \boldsymbol{w}^{(Z_i)} \right)}{\partial \boldsymbol{w}^{(\zeta)}} = \sum_{i=1}^{N_{\mathrm{atom}}} \delta^{\zeta}_{\mathrm{Z}_i} \frac{\partial \Phi^{(Z_i)}}{\partial \boldsymbol{w}^{(\boldsymbol{Z_i})}} \tag{4.33}$$

and, therefore,

$$\frac{\partial E^{\mathrm{NN}}}{\partial \boldsymbol{w}^{(\zeta)}} = \sum_{\substack{i \\ (\mathrm{Z}_i = \zeta)}} \frac{\partial \Phi^{(Z_i)}}{\partial \boldsymbol{w}^{(\boldsymbol{Z_i})}}. \tag{4.34}$$

The summation is over all atoms of element ζ. By using the notation and results of Section 2.3.1, where we described the backpropagation algorithm, we can directly state the components of this gradient for one summand on the right-hand side of Eq. 4.34. In the following, Φ is an unspecified NN function, so that we do not have to consider Z_i in the subsequent equations. For weights between the last hidden layer L and the output layer $L+1$, we find

$$g^L_{\alpha 1} = \frac{\partial \Phi}{\partial w^L_{\alpha 1}} = \frac{\mathrm{d} f_1^{L+1}}{\mathrm{d} u_1^{L+1}} y^L_\alpha. \tag{4.35}$$

For $0 \leq l \leq L$, we proved that

$$g^{l-1}_{\alpha\beta} = \sum_{i=1}^{m^{(l+1)}} h^l_i w^l_{\beta i} \frac{\mathrm{d} f^l_\beta}{\mathrm{d} u^l_\beta} y^{l-1}_\alpha, \tag{4.36}$$

where $h^l_i = g^l_{0\beta}$.

4.2.2. Neural network potential forces

To calculate force component F^{NN}_j we need to take the derivative of the total energy E^{NN} w.r.t. the jth component of the vector of all atomic positions, X_j.

$$F^{\mathrm{NN}}_j = -\frac{\partial E^{\mathrm{NN}}}{\partial X_j} = -\frac{\partial}{\partial X_j} \sum_{i=1}^{N_{\mathrm{atom}}} \Phi^{(Z_i)} \left(\boldsymbol{w}^{(\boldsymbol{Z_i})}, \boldsymbol{G_i}(\boldsymbol{X}) \right) \tag{4.37}$$

and thus

$$F_j^{\mathrm{NN}} = -\sum_{i=1}^{N_{\mathrm{atom}}} \frac{\partial \Phi^{(Z_i)}}{\partial X_j} = -\sum_{i=1}^{N_{\mathrm{atom}}} \sum_{j=1}^{T_i} \frac{\partial \Phi^{(Z_i)}}{\partial G_{i,j}} \frac{\partial G_{i,j}}{\partial X_j}, \tag{4.38}$$

where T_i is the number of SFs assigned to atom i and $G_{i,j}$ is the jth SF of atom i. In practice the argument list of a SF $G_{i,j}$ does not contain the position of every atom because of a cutoff radius and the fact that SFs normally process the position of certain elements only and so $\partial G_{i,j}/\partial X_j$ equals to zero for many j. By applying the chain rule, we obtain

$$\begin{aligned} \frac{\partial \Phi^{(Z_i)}}{\partial G_{i,k}} =& \frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} \sum_{r=1}^{m^{(L)}} w_{r1}^{L} \frac{\partial f_r^{L}}{\partial u_r^{L}} \\ & \cdot \sum_{s=1}^{m^{(L-1)}} w_{sr}^{L-1} \frac{\partial f_s^{L-1}}{\partial u_s^{L-1}} \ \cdots \sum_{u=1}^{m^{(2)}} \cdots \sum_{v=1}^{m^{(1)}} w_{vu}^{1} \frac{\partial f_v^{1}}{\partial u_v^{1}} \cdot w_{kv}^{0}, \end{aligned} \tag{4.39}$$

where all activation functions, weight parameters and the structure of the NN in the equation above are considered to be that of the NN for the element of atom i.

4.2.3. Weight derivatives of forces

For the adjustment of the weight parameters, we also need the derivative of a single force component w.r.t. weights. This is considerably more involved than the derivative of the energy w.r.t. weight parameters:

$$\frac{\partial F_q}{\partial w_{\alpha\beta}^{l}} = \frac{\partial}{\partial w_{\alpha\beta}^{l}} \sum_{i=1}^{N_{\mathrm{atom}}} \frac{\partial \epsilon_i}{\partial X_q} = \frac{\partial}{\partial w_{\alpha\beta}^{l}} \sum_{i=1}^{N_{\mathrm{atom}}} \sum_{j=1}^{T_i} \frac{\partial \epsilon_i}{\partial G_{i,j}} \frac{\partial G_{i,j}}{\partial X_q}. \tag{4.40}$$

$G_{i,j}$ is not a function of the weight parameters and thus

$$\frac{\partial}{\partial w_{\alpha\beta}^{l}} \frac{\partial G_{i,j}}{\partial X_q} = 0. \tag{4.41}$$

Therefore, we get

$$\frac{\partial F_q}{\partial w_{\alpha\beta}^{l}} = \sum_{i=1}^{N_{\mathrm{atom}}} \sum_{j=1}^{T_i} \frac{\partial G_{i,j}}{\partial X_q} \frac{\partial}{\partial w_{\alpha\beta}^{l}} \frac{\partial E_i}{\partial G_{i,j}} = \sum_{i=1}^{N_{\mathrm{atom}}} \sum_{j=1}^{T_i} \frac{\partial G_{i,j}}{\partial X_q} \gamma_{\alpha\beta}^{l}, \tag{4.42}$$

where we set $\gamma_{\alpha\beta}^{l} = \partial \Big/ \partial w_{\alpha\beta}^{l} \left(\partial E_i / \partial G_{i,j} \right)$. Since $\partial G_{i,j}/\partial X_q$ is to be solved individually, depending on the type of SF, we will focus on the expression

$$\gamma_{\alpha\beta}^{l} = \frac{\partial}{\partial w_{\alpha\beta}^{l}} \frac{\partial E_i}{\partial G_{i,j}}, \tag{4.43}$$

where we do not take into account the index of atom i in $\gamma^l_{\alpha\beta}$. We assume that all activation functions have continuous second order partial derivatives and thus we can interchange them

$$\frac{\partial}{\partial w^l_{\alpha\beta}}\frac{\partial E_i}{\partial G_{i,j}} = \frac{\partial}{\partial G_{i,j}}\frac{\partial E_i}{\partial w^l_{\alpha\beta}}. \tag{4.44}$$

We will develop the right hand side of Eq. 4.44 for a general NN function and therefore we replace $G_{i,j}$ by the jth NN input x_j and E_i by the one-dimensional output of a NN φ^L_1, given by

$$\varphi^L_1 = f^{L+1}_1\left(w^L_{01} + \sum_{r=1}^{m^{(L)}} w^L_{r1} y^L_r\right). \tag{4.45}$$

After this exchanges we arrive at

$$\gamma^l_{\alpha\beta} = \frac{\partial}{\partial G_{i,j}}\frac{\partial E_i}{\partial w^l_{\alpha\beta}} = \frac{\partial}{\partial x_j}\frac{\partial \varphi^L_1}{\partial w^l_{\alpha\beta}}. \tag{4.46}$$

For a weight between the last two layers, $w^L_{\alpha 1}$, we thus find

$$\begin{aligned}
\gamma^L_{\alpha 1} &= \frac{\partial}{\partial x_j}\frac{\partial}{\partial w^L_{\alpha 1}} f^{L+1}_1\left(w^L_{01} + \sum_{r=1}^{m^{(L)}} w^L_{r1} y^L_r\right) \\
&= \frac{\partial}{\partial x_j}\frac{\partial f^{L+1}_1}{\partial u^{L+1}_1} y^L_\alpha \\
&= \frac{\partial^2 f^{L+1}_1}{\partial x_j \partial u^{L+1}_1} y^L_\alpha + \frac{\partial f^{L+1}_1}{\partial u^{L+1}_1}\frac{\partial y^L_\alpha}{\partial x_j} \\
&= \frac{\partial^2 f^{L+1}_1}{\partial \left(u^{L+1}_1\right)^2}\frac{\partial u^{L+1}_1}{\partial x_j} y^L_\alpha + \frac{\partial f^{L+1}_1}{\partial u^{L+1}_1}\frac{\partial f^L_\alpha}{\partial u^L_\alpha}\frac{\partial u^L_\alpha}{\partial x_j}.
\end{aligned} \tag{4.47}$$

If the weight is a bias weight, that is $\alpha = 0$, we get

$$\gamma^L_{\alpha 1} = \gamma^L_{01} = \kappa^L_1 = \frac{\partial^2 f^{L+1}_1}{\partial \left(u^{L+1}_1\right)^2}\frac{\partial u^{L+1}_1}{\partial x_j} \tag{4.48}$$

because $\partial u^L_\alpha \big/ \partial x_j = 0$ and $y^l_0 = 1$.

For weights between the layers $L-1$ and L we can proceed similarly:

$$\begin{aligned}\gamma_{\alpha\beta}^{L-1} &= \frac{\partial}{\partial x_j}\frac{\partial}{\partial w_{\alpha\beta}^{L-1}} f_1^{L+1}\left(w_{01}^L + \sum_{r=1}^{m^{(L)}} w_{r1}^L\, f_r^L\left(w_{0r}^{L-1} + \sum_{s=1}^{m^{(L-1)}} w_{sr}^{L-1} y_s^{L-1}\right)\right) \\ &= \frac{\partial}{\partial x_j}\left[\frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial f_\beta^L}{\partial u_\beta^L} y_\alpha^{L-1}\right] \\ &= \frac{\partial^2 f_1^{L+1}}{\partial x_j \partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial f_\beta^L}{\partial u_\beta^L} y_\alpha^{L-1} + \frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial^2 f_\beta^L}{\partial x_j \partial u_\beta^L} y_\alpha^{L-1} \\ &\quad + \frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial f_\beta^L}{\partial u_\beta^L}\frac{\partial f_\alpha^{L-1}}{\partial u_\alpha^{L-1}}\frac{\partial u_\alpha^{L-1}}{\partial x_j}\end{aligned} \tag{4.49}$$

and

$$\begin{aligned}\gamma_{\alpha\beta}^{L-1} = &\frac{\partial^2 f_1^{L+1}}{\partial\left(u_1^{L+1}\right)^2}\frac{\partial u_1^{L+1}}{\partial x_j} w_{\beta 1}^L \frac{\partial f_\beta^L}{\partial u_\beta^L} y_\alpha^{L-1} + \frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial^2 f_\beta^L}{\partial\left(u_\beta^L\right)^2}\frac{\partial u_\beta^L}{\partial x_j} y_\alpha^{L-1} \\ &+ \frac{\partial f_1^{L+1}}{\partial u_1^{L+1}} w_{\beta 1}^L \frac{\partial f_\beta^L}{\partial u_\beta^L}\frac{\partial f_\alpha^{L-1}}{\partial u_\alpha^{L-1}}\frac{\partial u_\alpha^{L-1}}{\partial x_j}.\end{aligned} \tag{4.50}$$

For the purpose of deriving a general expression we formally introduce a summation over just one ($m^{(L+1)} = 1$) element

$$\begin{aligned}\gamma_{\alpha\beta}^{L-1} = \sum_{q=1}^{m^{(L+1)}} y_\alpha^{L-1} w_{\beta q}^L \Bigg[&\frac{\partial^2 f_q^{L+1}}{\partial\left(u_q^{L+1}\right)^2}\frac{\partial u_q^{L+1}}{\partial x_j}\frac{\partial f_\beta^L}{\partial u_\beta^L} + \frac{\partial f_q^{L+1}}{\partial u_q^{L+1}}\frac{\partial^2 f_\beta^L}{\partial\left(u_\beta^L\right)^2}\frac{\partial u_\beta^L}{\partial x_j} \\ &+ \frac{\partial f_q^{L+1}}{\partial u_q^{L+1}}\frac{\partial f_\beta^L}{\partial u_\beta^L}\frac{\partial f_\alpha^{L-1}}{\partial u_\alpha^{L-1}}\frac{\partial u_\alpha^{L-1}}{\partial x_j}\Bigg].\end{aligned} \tag{4.51}$$

From Eqs. 4.36, 4.48 and 4.51 we conclude for $l = L-1$:

$$\gamma_{\alpha\beta}^l = \sum_{q=1}^{m^{(l+2)}} y_\alpha^l w_{\beta q}^{l+1}\left[\kappa_q^{l+1}\frac{\partial f_\beta^{l+1}}{\partial u_\beta^{l+1}} + h_q^{l+1}\frac{\partial^2 f_\beta^{l+1}}{\partial\left(u_\beta^{l+1}\right)^2}\frac{\partial u_\beta^{l+1}}{\partial x_j}\right] + h_\beta^l \frac{\partial f_\alpha^l}{\partial u_\alpha^l}\frac{\partial u_\alpha^l}{\partial x_j}. \tag{4.52}$$

In Appendix B we prove that Eq. 4.52 is valid for $0 \le l \le L-1$. The result for $l = L$ is given in Eq. 4.47.

The only type of expression, we have not stated explicitly, but which we need for the computation of $\gamma_{\alpha\beta}^l$ for all α, β and $0 < l \le L$, is $\partial u_k^l \big/ \partial x_j$. The solution to this can also be easily obtained by the chain rule, in a similar way as in Eq. 4.39:

$$\frac{\partial u_k^l}{\partial x_j} = \sum_{r=1}^{m^{(l-1)}} w_{rk}^{l-1}\frac{\partial f_r^{l-1}}{\partial u_r^{l-1}} \sum_{s=1}^{m^{(l-2)}} w_{sr}^{l-2}\frac{\partial f_s^{l-2}}{\partial u_s^{l-2}} \cdots \sum_{u=1}^{m^{(2)}} \cdots \sum_{v=1}^{m^{(1)}} w_{vu}^1 \frac{\partial f_v^1}{\partial u_v^1}\cdot w_{jv}^0. \tag{4.53}$$

4.3. Selection of training instances

For training sets of moderate size it is usually not possible to use each energy and especially each force component several times during the optimization process. Therefore, the mechanism of selecting energies and force components can be very critical for the success of the training. This is particularly true for the case of interface configurations, where only a small part of the atoms are close to the interface. The selection method we choose for the training of the NNPs can be described as follows: The training is divided into several training segments. In each segment a certain number of energies and forces shall be chosen for the same number of weight updates. Before a new training segment starts, the RMSE of energies and forces is computed. For each scheduled weight update a random configuration or atom is chosen. Then, the actual error of the predicted energy of the chosen configuration or the value of a force component of the selected atom is computed. The predicted error, ΔE or ΔF_α, is compared with the RMSEs of the current training segment, RMSE(E) or RMSE(F_α). That means, only if

$$\Delta E \geq c \cdot \mathrm{RMSE}(E) \tag{4.54}$$

or

$$\Delta F_\alpha \geq c \cdot \mathrm{RMSE}(F_\alpha) \tag{4.55}$$

the scheduled weight update will be performed based on the chosen configuration or atom. If the condition is not met by the actual prediction error ΔE or ΔF_α, another configuration or atom is to be randomly chosen. This can be repeated n times per scheduled weight update. The nth configuration or atom will then be selected independent of the actual prediction error. For all NNPs described in this thesis, we set $c = 1.0$ and $n = 10$.

4.4. Alternative methods

There are two more ways of constructing a NNP based on the method of Behler and Parrinello. They used only one NNP. That means only one set of atomic NNs is employed to calculate the total energy and, by taking its derivative, all force components. Whereas this is very convenient since forces mathematically are the derivatives of the energy, it can be problematic to find the right balance between optimizing the weights for energies and forces (see factor β in Eq. 4.28). Thus, two different NNPs could be used, one for the approximation of energies and one for forces. This is, of course, computationally more expensive and less convenient, due to the fact the force components do not depend on the NNP used for energies.

Besides that method, there is a second way of approximating forces. Since force components are assigned to single atoms, it would be possible to predict forces just based on one single atomic NN, where the input of which still is a vector of SFs as a function of the local environment of the atom. The force vector of an atom could then be computed by taking the derivative of the one-dimensional output of its NN w.r.t. its Cartesian coordinates. However, this method does not provide the possibility to evaluate the total energy as the sum of the one-dimensional outputs of all atomic NNs. If the energy is desired, a classical NNP of Behler and Parrinello, fitted to energies only, is additionally needed.

5. Potential energy surface of water

To create a NNP, we need a function which describes the potential energy of a collection of atoms. An accurate description of the potential energy for a wide range of thermodynamic conditions can only be reliably achieved by applying ab initio quantum mechanical calculations. Basically, to this purpose, the Schrödinger equation needs to be solved. However, already for rather simple systems, approximations and numerical methods are necessary to obtain more or less accurate solutions within an acceptable computation time. When so-called ab initio molecular dynamics (AIMD) simulations shall be based on these computations, time is a very critical factor. A well-established method is DFT, which is capable of producing rather accurate results for many materials and which is also used for AIMD simulations because of its efficiency. Nevertheless, AIMD simulations for large systems and considerably long time scales are still infeasible with this method. Due to this, empirical potentials have been used to create long trajectories for large-scale MD simulations. Such empirical potentials are based on analytic functions and free parameters which need to be determined. The functional form is often based on theoretical considerations as well as intuition, whereas the fitting procedure of the parameters is often driven by results of experiments or quantum mechanical calculations. Compared to their quantum mechanical counterparts, empirical potentials can be evaluated very fast.

5.1. Density functional theory

DFT has become a very popular method for the quantum mechanical calculation of the ground state energy of a collection of atoms. It employs a number of approximations, which are necessary to keep this approach tractable. In this section, we follow the article of Car [64].

The first approximation within the framework of DFT is the well known Born-Oppenheimer approximation. It basically contains the assumption that due to the big difference in mass the electrons adiabatically follow the nuclei. Consequently, the electrons are, at each instance, considered to be in the ground state, corresponding to the particular configuration given by the position of the nuclei [64]. This enables the separation of the motion of electrons and nuclei and leads to the problem of solving

$$\hat{H}\Psi = E\Psi, \tag{5.1}$$

where the Hamiltonian $\hat{H}$ can be written as (by using atomic units) [64]

$$\hat{H} = \sum_i \frac{-\nabla_i^2}{2} + \sum_{i,j} \frac{-Z_i}{|\boldsymbol{r_i} - \boldsymbol{R_j}|} + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\boldsymbol{r_i} - \boldsymbol{r_j}|}. \tag{5.2}$$

The $\{\boldsymbol{r_i}\}$ and $\{\boldsymbol{R_j}\}$ are the position vectors of the electrons and nuclei, respectively. The first sum represents the kinetic energy operator of the electrons, whereas the second and third sum represent the interaction energy between electrons and the collection of nuclei and the interaction between different electrons, respectively. Z_j is the atomic number of nucleus j. For a given nuclear configuration $\{\boldsymbol{R}\}$, we can easily express the potential energy ϕ of the nuclei [64]:

$$\phi(\{\boldsymbol{R}\}) = E + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\boldsymbol{R_i} - \boldsymbol{R_j}|}. \tag{5.3}$$

For classically treated dynamics of the nuclei, ϕ acts just like a classical inter-atomic potential and the main problem is the calculation of E. Since we want to find the ground state, the computation of E can be seen as a variational problem [64]:

$$E = \min_{\Psi} \langle \Psi | \hat{H} | \Psi \rangle . \tag{5.4}$$

This task can be a very laborious one because already for a simple H_2O molecule the electronic wave function Ψ is a function in 30 spatial dimensions. A CO_2 molecule requires 66 spatial dimensions. In DFT the electron density ρ is the central quantity. It is defined as [64]

$$\rho(\boldsymbol{r}) = n \int \mathrm{d}\boldsymbol{r_1} \mathrm{d}\boldsymbol{r_2} \ldots \mathrm{d}\boldsymbol{r_{n-1}} \, |\Psi(\boldsymbol{r}, \boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_{n-1}})|^2, \tag{5.5}$$

where n is the number of electrons. The big advantage of the electron density is the reduction of dimensionality because it only depends on three spatial dimension as opposed to the electron wave function. Instead of minimizing E by varying Ψ, the ground state energy shall be found by the minimization [64]

$$E = \min_{\rho} E_V[\rho]. \tag{5.6}$$

Here, E_V means the energy E with respect to the external local potential V originating from the fixed array of nuclei. $E_V[\rho]$ is a functional of the electron density ρ. That Eqs. 5.4 and 5.6 lead to the same ground state energy, E, states a theorem according to Hohenberg and Kohn [65]. Furthermore, $E_V[\rho]$ can be written as [64]

$$E_V[\rho] = \int \mathrm{d}\boldsymbol{r} V(\boldsymbol{r}) \rho(\boldsymbol{r}) + \frac{1}{2} \int \mathrm{d}\boldsymbol{r} \mathrm{d}\boldsymbol{r'} \frac{\rho(\boldsymbol{r}) \rho(\boldsymbol{r'})}{|\boldsymbol{r} - \boldsymbol{r'}|} + F[\rho]. \tag{5.7}$$

In our case $V(r)$ is, of course, just the electrostatic potential originating from the nuclei, which positions are considered to be fixed. The first two terms of Eq. 5.7 are certainly not sufficient to ensure the equivalence of Eqs. 5.4 and 5.6. Therefore, the functional $F[\rho]$ compensates for all other necessary but unknown terms. $F[\rho]$ includes the many-body kinetic energy of the electrons as well as exchange and correlation effect corrections to the classical potential energy terms in Eq. 5.7. Exchange effects come from the antisymmetry of the electronic wavefunction and correlation effects are the result of electrostatic repulsion of electrons [64]. It is a hard task to find accurate approximations for $F[\rho]$. A very popular approach to overcome this problem, proposed by Kohn and Sham [66], is the replacement of the real interacting electrons by fictitious non-interacting electrons moving in a modified potential. These non-interacting electrons are described by orthonormal single-particle wavefunctions ψ_i. $E_V[\rho]$ can then be expressed as [64]

$$E_V[\rho] = \int \mathrm{d}\boldsymbol{r} V(\boldsymbol{r})\rho(\boldsymbol{r}) + \frac{1}{2}\int \mathrm{d}\boldsymbol{r}\mathrm{d}\boldsymbol{r'}\frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r'})}{|\boldsymbol{r}-\boldsymbol{r'}|} + 2\sum_i \left\langle \psi_i \left| \frac{-\nabla^2}{2} \right| \psi_i \right\rangle + E_{\mathrm{XC}}[\rho]. \tag{5.8}$$

With this approach E_V shall be minimized with respect to the ψ_i. The functional $E_{\mathrm{XC}}[\rho]$ is the so-called exchange-correlation energy and also contains corrections for obtaining the true ground state energy. It is again necessary to find an approximation for $E_{\mathrm{XC}}[\rho]$, but since Eq. 5.8 now contains a single particle kinetic energy term, this task is easier than finding an approximation for $F[\rho]$. However, the accuracy of DFT still crucially depends on the approximation of $E_{\mathrm{XC}}[\rho]$. For this task it is convenient to define the exchange and correlation energy per electron ϵ_{XC}, [64]

$$E_{\mathrm{XC}}[\rho] = \int \mathrm{d}\boldsymbol{r}\rho(\boldsymbol{r})\epsilon_{\mathrm{XC}}(\boldsymbol{r}) \tag{5.9}$$

The local density approximation (LDA) simply assumes that the exchange and correlation energy per electron $\epsilon_{\mathrm{XC}}(\boldsymbol{r})$ is that of a uniform homogeneous electron gas, $\epsilon_{\mathrm{XC}}^{\mathrm{hom}}$, of the same density as the system under consideration at point $\boldsymbol{r}$, $\rho(\boldsymbol{r})$. That means [64]

$$\epsilon_{\mathrm{XC}}^{\mathrm{LDA}}(\boldsymbol{r}) = \epsilon_{\mathrm{XC}}^{\mathrm{hom}}(\rho(\boldsymbol{r})) \tag{5.10}$$

and

$$E_{\mathrm{XC}}^{\mathrm{LDA}}[\rho] = \int \mathrm{d}\boldsymbol{r}\rho(\boldsymbol{r})\epsilon_{\mathrm{XC}}^{\mathrm{hom}}(\rho(\boldsymbol{r})). \tag{5.11}$$

This approach is especially reasonable for systems with slowly varying electron density. A more advanced approach is the generalized gradient approximation (GGA). If the electron density undergoes rapid changes, the GGA can be a substantial improve-

ment over the LDA because here the exchange and correlation energy per electron at point $\boldsymbol{r}$ is a function of the electron density and its gradient and thus we have [64]:

$$E_{\mathrm{XC}}^{\mathrm{GGA}}[\rho] = \int \mathrm{d}\boldsymbol{r}\rho(\boldsymbol{r})\epsilon_{\mathrm{XC}}^{\mathrm{GGA}}(\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r})). \tag{5.12}$$

Unlike for the case of the LDA, there is not a unique way of defining $\epsilon_{\mathrm{XC}}^{\mathrm{GGA}}$. Two widely used approaches are the BLYP [16, 17] and RPBE [18] density functionals. In this work we use the RPBE density functional for the necessary ground state calculations. Since GGA-based density functionals are not able to correctly describe vdW forces, we additionally apply a vdW correction scheme, namely the D3 method [19].

5.2. Empirical potential energy functions

Even though DFT provides a good and efficient tool to accomplish approximate ab initio calculations, we are still forced to employ empirical potentials in order to perform simulations on large systems and long time scales. To test how well NNPs work for the liquid-vapor interface of water, we also employ empirical potentials.

5.2.1. SPC model

The simple point charge (SPC) model [67] is one of many empirical water models. It is a three-site model, which means that one water molecule is represented by three interaction sites. These interaction sites can be identified with the centers of the two hydrogen atoms and the oxygen atom. On the hydrogen atoms and on the oxygen atom there is a partial charge q_{H} and q_{O}, respectively. The potential energy is characterized by two types of interactions. Besides the Coulomb interaction between all sites of different molecules, also a Lennard-Jones (LJ) potential with parameters ϵ_{O} and σ_{O} is applied to oxygen atoms. Thus, the potential energy $V_{\alpha\beta}$ between two distinct water molecules α and β can be expressed as

$$V_{\alpha\beta} = 4\epsilon_{\mathrm{O}}\left(\frac{\sigma_{\mathrm{O}}^{12}}{r_{\mathrm{OO}}^{12}} - \frac{\sigma_{\mathrm{O}}^{6}}{r_{\mathrm{OO}}^{6}}\right) + \sum_{i}^{\text{on }\alpha}\sum_{j}^{\text{on }\beta}\frac{q_i q_j}{r_{ij}}, \tag{5.13}$$

where r_{OO} is the distance between the oxygen atoms of the molecules α and β, q_i is the partial charge of atom i of molecule α and q_j is the partial charge of atom j of molecule β. Furthermore, r_{ij} is the distance between these two atoms i and j and

the summations are over all atoms of the respective molecules. The total energy of a collection of N molecules is then

$$V_{\text{tot}} = \sum_{\alpha<\beta}^{N} V_{\alpha\beta}. \tag{5.14}$$

The SPC model imposes fixed O-H bond lengths r_{OH} and a fixed H-O-H bond angle θ_{HOH} on the water molecules. The exact geometry of a SPC water molecule can be seen in Fig. 5.1. All parameters, namely ϵ_{O}, σ_{O}, q_{H}, q_{O} as well as r_{OH} and θ_{HOH} are listed in Table 5.1. However, due to the rigid geometry of water molecules in the SPC model, it is not a good choice for testing the NNP because the NNPs shall be used to fit the PES of ab initio methods, which do not know any restrictions on the geometry of a molecule. Simulations based on ab initio calculations even allow for bond breaking and reformation. Flexible SPC models are not capable of simulating bond breaking, but they remove the restriction of rigid molecules.

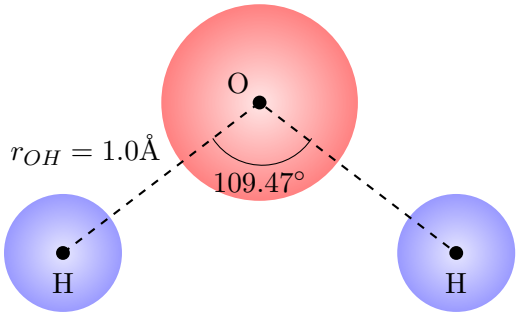


Figure 5.1.: Rigid water molecule with O-H bonds of length 1.0Å and H-O-H angle of 109.47°.

5.2.2. SPC/F model

The SPC/F model [68] is a modified and flexible version of the SPC model. That means the geometry of the water molecule is not rigid but flexible. There is, however, an equilibrium geometry defined by the parameters r_{OH} and θ_{HOH}, the values of which are the same as for the rigid SPC model. The flexible angle and the flexible bonds are characterized by harmonic potentials. The total energy can then be modeled as a sum of intermolecular and intramolecular forces

$$V_{\text{tot}} = V_{\text{intra}} + V_{\text{inter}}, \tag{5.15}$$

where V_{inter} can be expressed as

$$V_{\text{inter}} = \sum_{\alpha<\beta}^{N} V_{\alpha\beta} \tag{5.16}$$

	SPC	**SPC/F**
r_{OH} (Å)	1.000	1.000
θ_{HOH} (deg)	109.470	109.470
q_{O} (e)	-0.820	-0.780
q_{H} (e)	0.410	0.390
ϵ_{O} (meV)	6.735	7.003
σ_{O} (Å)	3.166	3.145
k_r (keV Å^2)	–	480.593
k_θ (eV rad^2)	–	3.969

Table 5.1.: Parameters for the SPC and SPC/F water models [69, 70]. For the non-rigid SPC/F model r_{OH} and θ_{HOH} are equilibrium values.

with $V_{\alpha\beta}$ from Eq. 8.1. The intramolecular potential energy is defined as

$$V_{\mathrm{intra}} = \sum_{\alpha=1}^{N} \left[\frac{k_r}{2} \left(\Delta_\alpha r_{\mathrm{OH}_1}^2 + \Delta_\alpha r_{\mathrm{OH}_2}^2 \right) + \frac{k_\theta}{2} \Delta_\alpha \theta_{\mathrm{HOH}}^2 \right] \tag{5.17}$$

where $\Delta_\alpha r_{\mathrm{OH}_1}$ is the difference between the equilibrium bond length and the actual bond length between the oxygen atom and the first hydrogen atom of molecule α. The quantities $\Delta_\alpha r_{\mathrm{OH}_2}$ and $\Delta_\alpha \theta_{\mathrm{HOH}}$ are defined similarly and k_r and k_θ are constants of the harmonic potentials. The values for all needed parameters of the SPC/F model can be found in Table 5.1.

6. Fitting low-dimensional data using neural networks

Our main interest in this work is, of course, to fit empirical potential energy functions and especially energies from DFT calculations. The algorithm we use for this purpose is more than just a simple MLP. First of all, this algorithm does not only fit a NN to function values but also to derivatives of the function. Secondly, the steps for preprocessing the input variables (input coordinates of atoms) are by far more elaborate than just linearly transforming the input and output values for scaling to the certain intervals. A third peculiarity is the usage of the EKF as optimization algorithm for NNs together with parallel training. In order to check and visualize the functionality of the core part of the algorithm, we modified the code in such a way that we can fit a NN to simple functions of one or just a few variables without taking function derivatives or the aforementioned preprocessing steps into account.

6.1. One-dimensional functions

To start with a simple example, we consider the polynomial reference function

$$f_{\mathrm{ref}}(x) = x - 2.0x^2 + 0.5x^3 - 0.032x^4. \tag{6.1}$$

If we try to fit a polynomial of degree 4,

$$f_{\mathrm{model}}(x) = a + bx - cx^2 + dx^3 - ex^4 \tag{6.2}$$

to Eq. 6.1, we see that a function with 5 parameters can be complex enough to obtain an adequate fit. We demonstrate this by utilizing the function **optimize.fmin** which is part of the open source Python library SciPy [71, 72]. This function employs a Nelder-Mead simplex algorithm [73] to find the minimum of

$$\sum_{i=1}^{N} (f_{\mathrm{model}}(x_i) - f_{\mathrm{ref}}(x_i))^2, \tag{6.3}$$

where $\{x_i\}$ is a certain set of input values. For this algorithm one needs to define an initial guess for the parameters to be optimized. We chose

$$\{a, b, c, d, e\}_{\mathrm{init}} = \{5.0, 5.0, 5.0, 5.0, 5.0\}$$

and found the final parameters

$$\{a, b, c, d, e\}_{\text{final}} = \{-6.3 \cdot 10^{-6}, 1.000, 2.000, 0.500, 0.032\}. \tag{6.4}$$

Fig. 6.1 shows the reference function, Eq. 6.1, and the model function with the final parameters. During the fitting procedure we only used input values from the interval $[0, 6]$, but we can clearly see that the model function precisely coincides with the reference function also outside the interval. We emphasize this point because we will see that, in general, a NN fit is bad at estimating values beyond the original range of observation.

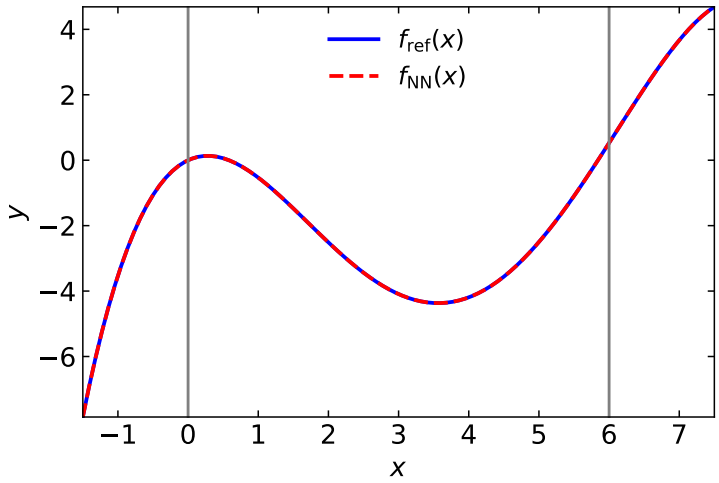


Figure 6.1.: Comparison between the reference function (Eq. 6.1) and the fitted model function (Eq. 6.3) using the optimized parameter set of Eq. 6.4.

Next, we used the modified code to fit a NN to the polynomial function. We employed a NN with one hidden layer consisting of 10 neurons. As activation functions we chose the hyperbolic tangent function for the hidden layer and the identity function for the input as well as the output layer. To optimize the weight parameters we used the EKF with forgetting factor with parameters λ=0.98, μ=0.9989, q_0=0.0 and p_0=1000. We randomly selected 500 data points for the training procedure within the interval $x \in [0, 6]$ by employing a uniform distribution. After 10 epochs we obtained a RMSE of $3.41 \cdot 10^{-4}$. To visualize the accuracy of the NN fit, we let the trained NN predict 500 outputs by randomly choosing new 500 input values on the interval $[-1.5, 7]$. Even though the trained NN gave accurate predictions on the interval $x \in [0, 6]$, the deviations from the reference functions were especially large for $x < 0$. We can clearly see this issue in Fig. 6.2.

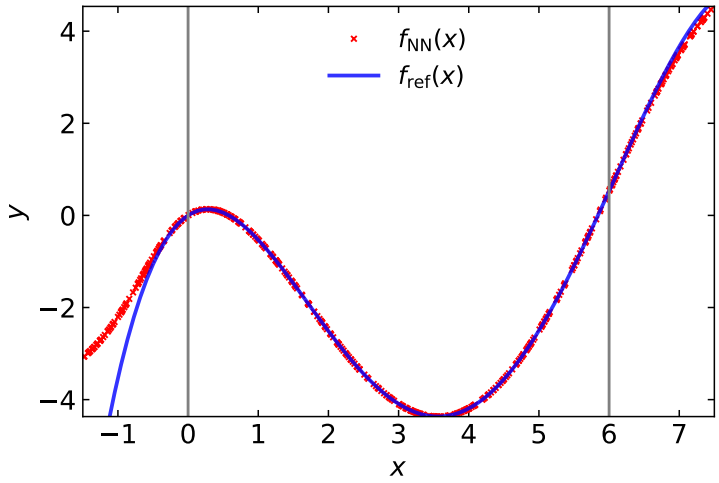


Figure 6.2.: NN fit of the polynomial function, Eq. 6.1. Each red cross represents the predicted output of the NN for an input value, which the NN has not seen during training. The range of observation during training was the interval $[0, 6]$ visualized by two vertical lines in the graph.

6.2. Two-dimensional functions

To show that a NN is capable of learning functions of more than one input variables, we consider the following reference function

$$f_{\text{ref}}(x, y) = \cos^2(x)\sin(y^2) + \sin^3(y). \tag{6.5}$$

We randomly selected 10,000 points (x, y), whereas both x and y are restricted to the interval $[0, \pi]$. In order to show once again that NNs are not the method of choice when it comes to extrapolating functions beyond the limits of the domain used for training, we look at the predicted output for another randomly selected 2,500 points, where we extended the interval for y-values to $[0, 4.1]$. In Fig. 6.3 it can be easily seen that the NN fails to approximately reproduce the reference function for $y > \pi$. Nonetheless, within the domain of observation we find a RMSE of $6.76 \cdot 10^{-3}$ for the training data set and $6.95 \cdot 10^{-3}$ for unseen data points after 10 training epochs. The NN used here contained two hidden layers, each consisting of 10 neurons, and the hyperbolic tangent function was employed as activation function in these hidden layers. We used the same parameters for the EKF as for the one-dimensional case.

6.3. Potential energy of water molecules

As last example, we consider the case of water molecules. We employed the SPC/F water model (see Section 5.2.2) to calculate the energy of a given configuration. We only used one or two water molecules to keep things simple. This example demonstrates that it is very important how the data is presented to the NN.

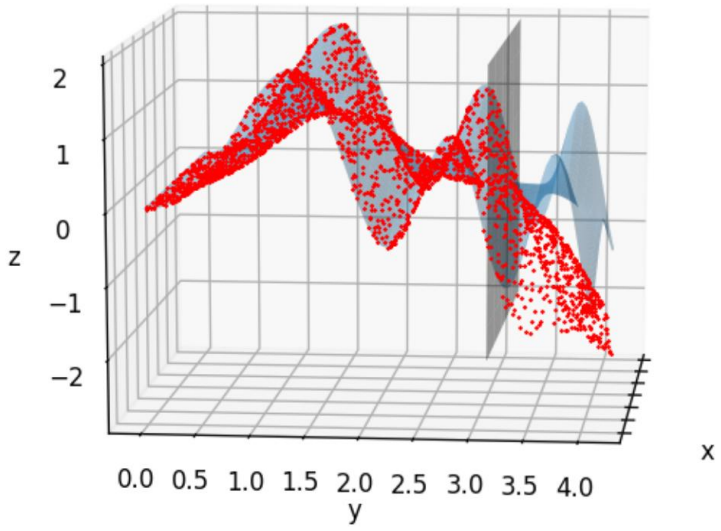


Figure 6.3.: NN fit of the reference function $z = f_{\text{ref}}(x, y) = \cos^2(x)\sin(y^2) + \sin^3(y)$. The NN output is represented by red crosses and the reference function is indicated by a blue surface. The original range of observation was $0 < x < \pi,\ 0 < y < \pi$. The NN function does not longer match the reference function for y-values greater than π, in the graph separated by a gray plane.

6.3.1. One water molecule

We start with the case of one water molecule. We generated a training set consisting of 2,000 single water molecules in a box of $20 \times 20 \times 20$ Å^3. The molecules were randomly placed and oriented in the box, where each bond length was randomly drawn from a uniform distribution over the interval [0.8 Å, 1.2 Å]. The bond angle was uniformly distributed over the interval [90 deg, 130 deg]. To simplify the NN fit, we did not accept molecules with atoms separated from each other by one of the periodic boundaries of the box. Each so-created instance of the training set contained 9 Cartesian coordinates as input and the corresponding potential energy as desired output. Based on these instances, we derived a second training set by transforming the 9 Cartesian coordinates to three interatomic distances. For the training set with Cartesian coordinates, we naturally used an input layer composed of 9 neurons, whereas the training set utilizing interatomic distances requires 3 input neurons. The structures of the used NNs with an emphasis on the input layer for the training sets with Cartesian coordinates and interatomic distances are depicted in Figs. 3.1 and 6.4, respectively. One may expect that both variants work well, but surprisingly we could not find any settings which could provide reasonably accurate training results for the case of the Cartesian coordinates. We tried many different ways to improve this, such as rescaling the input and the output values to the range $[-1, 1]$ or $[0, 1]$, using many different NN architectures and a wide variety of parameter combinations for the EKF. However, the best NN fit we could achieve was still very bad. The NN of this fit was composed of 2 hidden layers with 12 neurons

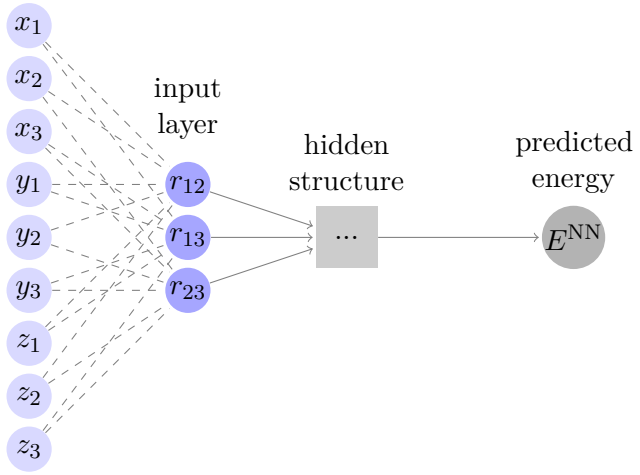


Figure 6.4.: Structure of a NN used to predict the energy of one water molecule based on interatomic distances. The 9 Cartesian coordinates of the water molecule are transformed to the complete set of interatomic distances, r_{12}, r_{13}, r_{23}, which are subsequently used for the input layer.

each. The input and output values were rescaled to the range $[0, 1]$. All hidden nodes utilized the hyperbolic tangent function and the EKF parameters were set to λ=0.985, μ=0.9985, q_0=0.0 and p_0=1000.
In contrast, it was easy to train a NN in order to predict the energy of a single water molecule when the three interatomic distances were used as inputs. Again, rescaling the input and output to the interval $[0, 1]$ improved the fitting accuracy. For this NN fit, we used the same parameters and architecture as for the case of Cartesian coordinates, but we changed the number of hidden neurons per hidden layer to 10. In Fig. 6.5 we plotted some of the predicted energies against the correct ones for the best fit we could achieve for both Cartesian coordinates and interatomic distances. The RMSEs per molecule (for the training set) of the fits with Cartesian coordinates and interatomic distances are approximately 362 meV and 1 meV, respectively. The reason for these difficulties in the case of Cartesian coordinates as input values seems to lie in the additional complexity due to the transformation from Cartesian coordinates to interatomic distances. Interestingly, we needed a NN consisting of three hidden layers to accurately approximate the following function:

$$d_1(x_1, x_2, y_1, y_2, z_1, z_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}. \tag{6.6}$$

Furthermore, independent of the choice of the NN architecture, we could not find a good NN approximation of the function d_2,

$$\begin{aligned} d_2(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4) \\ = d_1(x_1, x_2, y_1, y_2, z_1, z_2) + d_1(x_3, x_4, y_3, y_4, z_3, z_4), \end{aligned} \tag{6.7}$$

which can be seen as the sum of the two Euclidean distances of two pairs of atoms in the three-dimensional space.

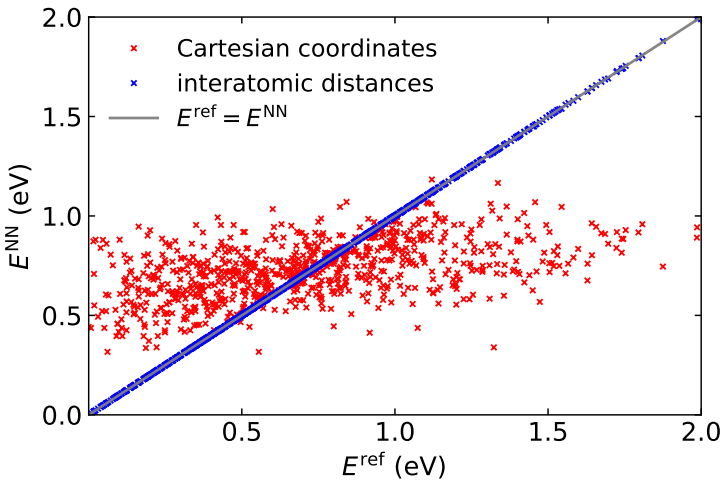


Figure 6.5.: NN fit of potential energy function of one water molecule for two different types of inputs. Red crosses represent the output of a NN using Cartesian coordinates (see Fig. 3.1) and blue crosses show the predicted energy values of the distance-dependent NN (see Fig. 6.4). This graph shows predicted energies of 950 configurations that the NN has not seen during training.

6.3.2. Two water molecules

The main problem the NN has to deal with is not the higher number of inputs (9 Cartesian coordinates as opposed to 3 interatomic distances). We can easily see this by considering the case of two water molecules. We used the full set of 15

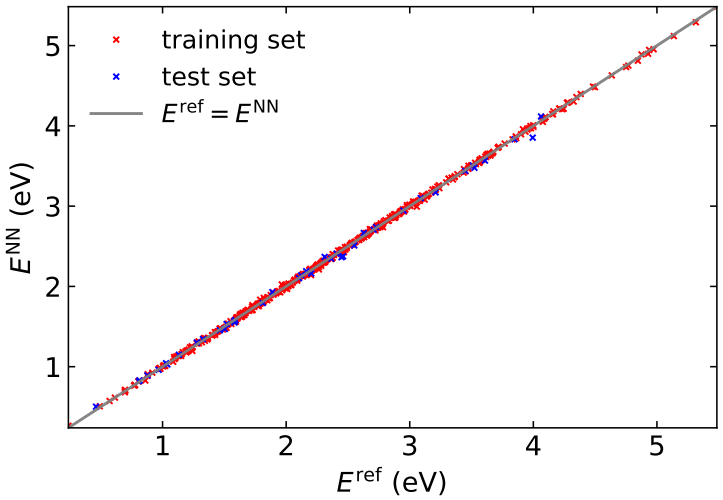


Figure 6.6.: Predictions of a NN against the reference energies of 2 water molecules. The input layer contains 15 nodes, each representing one out of 15 interatomic distances.

interatomic distances between these 6 atoms as input variables of the NN, which gives us a complete, although overdetermined, description of the system of 2 water molecules. The resulting accuracy of the fit is still satisfying as shown in Fig. 6.6. For generating the training and test set, we placed two randomly oriented molecules in a box of $40 \times 40 \times 40$ Å^3, exactly in the same way as in Section 6.3.1. One of these molecules was then shifted, whereas the new distance was drawn from a uniform distribution on the interval [6.5 Å, 11.5 Å]. The training and test set contained 400 and 40 configurations, respectively. The NN was composed of two hidden layers, 20 neurons in the first and 10 neurons in the second hidden layer. All other settings were identical to the NN training of the energy of one water molecule. The RMSE of the training set was around 16 meV and 38 meV for the test set.
We want to stress that, under certain conditions, Cartesian input coordinates might work as well, but our example showed that a set or subset of interatomic distances can be the better choice for the training of a NN to accurately represent the PES of atomic systems.

7. Neural network potential simulations of liquid-vapor interfaces

In the paper of Morawietz *et al.* [15], Behler-Parrinello NNPs were used to perform MD simulations of bulk water. With the aid of these NNPs it has been shown how important vdW interactions are for the unique properties of water. For this purpose, the energy and forces of about 7,200 periodic condensed water configurations have been calculated employing the BLYP and RPBE density-functionals. For each density-functional these calculations have been done with and without vdW corrections using Grimme's D3 method [19] with the zero-damping scheme and neglecting three-body contributions. We were provided with the training data from the RPBE density-functional with vdW correction and we employed NNPs trained on this data set to perform MD simulations of the liquid-vapor interface of water. However, this data set does not include interface configurations of the liquid and its vapor. Since NNs are bad at extrapolating, as we have shown in Chapter 6, a NNP trained only on the original data set of Morawietz *et al.* is expected to fail at predicting accurate energies and forces of atoms near the liquid-vapor phase boundary. Therefore, we aim to extend the data set by adding configurations with liquid-vapor interfaces. Unfortunately, NNPs are limited in two ways, what can state a problem, especially when it comes to interfaces.

7.1. Limitations of neural network potentials

Over the last few years it has been shown that Behler-Parrinello NNPs are a very useful tool to overcome the high computational complexity of first principle methods and the lack of predictive power of empirical potentials. Nonetheless, it should not be forgotten that such NNPs are just a tool to approximate PESs from ab initio calculations. Besides the fact that inappropriate training can lead on to very bad results, there are two more limitations which can decrease the accuracy of the NNP.

7.1.1. Symmetry function cutoff

Within the approach of Behler and Parrinello the energy of each atom is represented by an own atomic NN (see Fig. 4.1). For each atom i, a transformation

$$\boldsymbol{R} \rightarrow \boldsymbol{G_i}(\boldsymbol{X}) \tag{7.1}$$

is necessary to obtain the input vector for atomic NN i. Here, $\boldsymbol{X}$ is the vector of all Cartesian atomic coordinates and $\boldsymbol{G_i}$ represents the vector of SF values assigned to atom i. If for a system of N atoms each transformation required the Cartesian coordinate vectors of each of the other $N-1$ atoms, leaving aside periodic boundary conditions, $\mathcal{O}(N^2)$ values would be processed by the totality of all SFs. This can be very time consuming and therefore, in order to reduce the computational costs of the NNP algorithm, an individual cutoff for each SF function is used. This leads to an unavoidable loss of information and especially when we deal with interfaces, this can be very critical for the accuracy of the NNP. A possible problematic situation is depicted in Fig. 7.1. It must be noted that during the training the NNP is provided with energies and forces calculated without such a loss of information, but already during the training the atomic NN assigned to the atom in the middle of the cutoff sphere in Fig. 7.1 is not able to distinguish between the two depicted atomic environments. This may give rise to some kind of averaging process over all training configurations regarding effects coming from atoms outside the cutoff sphere.

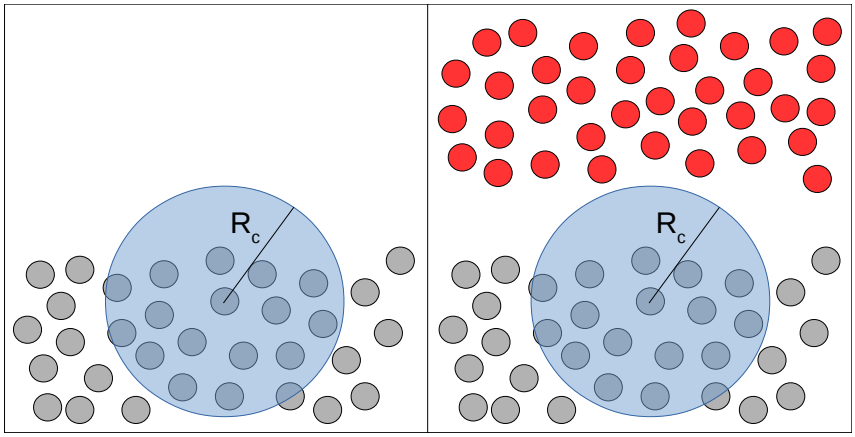


Figure 7.1.: Two equally sized simulation boxes with different configurations. Additional atoms in the right box are red colored. An atomic NN assigned to the atom in the center of the blue cutoff sphere yields the same atomic energy for both configurations.

7.1.2. Uniqueness of symmetry function vectors

As we have seen in the last section, for a system of N atoms the vector of all Cartesian atomic coordinates $\boldsymbol{X}$ is transformed to N vectors of SF values $\{\boldsymbol{G_i}\}$. For the sake of simplicity, let the number of components of each $\boldsymbol{G_i}$ be $c =$ constant and keep in mind that c normally is much smaller than N. Then, by the transformation of Eq. 7.1, we obtain cN symmetry function values from $3N$ atomic Cartesian coordinates. Usually c is larger than 3 and thus one could conclude that enough information is preserved to uniquely describe the given configuration by the totality of all SFs. However, the SF input vectors are not shared between the atomic NNs and effectively

only c values are used to characterize the atomic environment of an individual atom. Although only a part of all $N-1$ other atoms is taken into account due to the cutoff, SFs are still just a means to approximately describe atomic environments within a certain cutoff radius. For example, consider identical atoms interacting only via a LJ potential. Define r_{ki} as the distance between atom k and another atom i and let n_c be the number of neighbors of atom k within the cutoff sphere. If we wanted to uniquely find a relation between the SF vector and the set of distances $\{r_{ki}\}$ for $i = \{1, 2, ..., n_c\}$, the SF vector would need at least n_c components. Assume now a system in which an atom k is typically surrounded by 30 other atoms within the NNP cutoff sphere. In such a case we would usually employ between 5 and 12 radial SFs to characterize the radial atomic environment. It is clear, that it is impossible to uniquely describe the set of 30 interatomic distances by only 5 to 12 numbers. One reason why this approach can still provide good results may be that the distribution of atoms is not completely arbitrary. Especially in condensed phase systems some structural order is present and, therefore, only a few numbers may be sufficient to accurately distinguish between different occurring atomic environments. If the same NNP is utilized to predict energies and forces for systems under various thermodynamic conditions accompanied by substantial changes of the structural order, the limited number of SFs may not be sufficient to reliably characterize each of the possible atomic environments. Even though we have already learned from the work of Morawietz *et al.* [15] that the Behler-Parrinello NNP approach works for bulk water in the liquid and ice phase at various densities and also for the interface between them, it is not clear if water molecules at the liquid-vapor interface can be successfully described in the same way. The main problem could be the highly anisotropic distribution of atoms at the interface, where the density rapidly changes from that of the liquid to almost zero.

7.2. Testing accuracy and stability

The potential issues described in section 7.1 may negatively affect the accuracy or even the stability of a MD simulation of interfaces employing a NNP. In order to test the accuracy and stability, we utilize the empirical SPC/F water model (see section 5.2.2) to provide the NNP with training data before we proceed with DFT calculations. By using an empirical potential, we can easily compare results from simulations with the reference model as well as the NNP. This may give us valuable insights whether or not we can expect reliable results from simulations of the liquid-vapor interface with NNPs trained on DFT data. For the purpose of comparison, we evaluate different observables, namely the mass density profile perpendicular to the flat interface, the surface tension and also microscopic properties, for instance the distribution of OH bond distances in general and as a function of the position

on the axis perpendicular to the interface. The observables being used are described in the following sections.

7.3. Observables

7.3.1. Density profile

The density profile is the density as a function of some spatial variable. In our case, since we are interested in the characterization of interfaces, this variable is the position on the axis which is perpendicular to the flat interface we simulate. In our simulation, this is always the z-axis. For the definition of the density profile along the z-axis, the simulation box is divided into identical slabs of thickness Δz. For each slab s, an indicator function $\chi_s(z)$ is defined as

$$\chi_s(z) = \begin{cases} 1, & \text{if } s < z/\Delta z < s+1 \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

The value of the density profile in the interval $[\Delta z{\cdot}s, \Delta z{\cdot}(s+1)]$, denoted as ρ_s is then given by

$$\rho_s = \left(\frac{1}{L_x L_y \Delta z}\right) \sum_{i \in \text{atoms}} \chi_s(z_i) p_i, \tag{7.3}$$

where z_i is the position of atom i and p_i is a specific atomic property of atom i like its mass for the mass density profile or just 1 when the number density profile is desired. L_x and L_y are the lengths of the x and y side of the simulation box. An example of a density profile is depicted in Fig. 7.2. It is important to note that each frame is modified before it is used for the calculation of the density profile. That is because during the simulation of a slab geometry the center of mass can considerably move in a direction perpendicular to the interfaces. This can give rise to undesired effects on the density profile, what can be prevented by shifting the slab so that the z-coordinate of the center of mass is the same in every frame. If the two interfaces of the slab are separated by the periodic boundary perpendicular to the z-axis, this needs to be changed by another shift of the slab even before the z-coordinate of the center of mass is calculated.
From the density profile the density of the coexisting liquid and vapor phases, ρ_L and ρ_V, respectively, can be found. This can be done by a Levenberg-Marquardt least-square fit of the function [74]

$$\rho(z) = \frac{1}{2}\left(\rho_\mathrm{L} + \rho_\mathrm{V}\right) - \frac{1}{2}\left(\rho_\mathrm{L} - \rho_\mathrm{V}\right) \tanh[(z - z_0)/d] \tag{7.4}$$

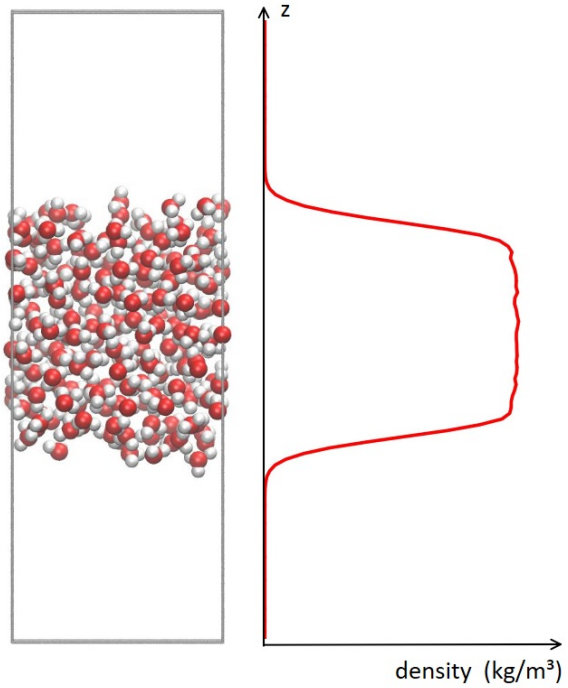


Figure 7.2.: Mass density profile for slab configurations of water. The plotted curve results from 1000 different configurations, obtained from a MD simulation.

to the density profiles, where z_0 represents the middle of the interface and d is a parameter describing its thickness. We accomplish these fits by utilizing the function **optimize.curve_fit**, which is part of the open source Python library Scipy [71, 72].

7.3.2. Surface tension

The surface tension γ is the energy required to increase the area of a surface by one unit area. Thermodynamically, in the canonical ensemble, this can be written as [75]

$$\gamma = \left(\frac{\partial F}{\partial A}\right)_{N,V,T}, \tag{7.5}$$

where F is the Helmholtz free energy, A is the surface area, N is the number of particles, V is the volume and T is the temperature of the system under consideration. For the simulations of the liquid-vapor interface we utilize a slab-geometry with periodic boundary conditions and, therefore, we need an expression for determining the surface tension of a planar interface. For this purpose, we employ the formula by Kirkwood and Buff based on the pressure tensor [76, 77]

$$\gamma = \frac{L_z}{2}\left(\langle P_{zz}\rangle - \frac{\langle P_{xx} + P_{yy}\rangle}{2}\right), \tag{7.6}$$

where L_z is the length of the simulation box edge parallel to the interface normal and $P_{\alpha\beta}$ is the $\alpha\beta$th element of the pressure tensor, which can be written as

$$P_{\alpha\beta} = \frac{1}{V}\left(\sum_{k=1}^{N} m_k v_{k\alpha} v_{k\beta} + \sum_{k=1}^{N'} r_{k\alpha} f_{k\beta}\right). \tag{7.7}$$

In this equation m_k denotes the mass of atom k, $r_{k\alpha}$ and $v_{k\alpha}$ are the α-th component of the position vector and velocity vector of atom k, respectively, and $f_{k\beta}$ is the βth component of the force acting on atom k. The second summation is over N' instead of N elements. N' includes not only all atoms of the local cell but also such ghost atoms which interact with atoms of the local cell by a finite-range potential [78] (electrostatic interactions must be treated differently [77]). The force acting on these ghost atoms is considered to be the partial force, originating only from interactions with atoms of the local cell.

7.3.3. Molecular orientation of molecules at the interface

To learn more about the microscopic properties of the interface, we analyze the orientation of the water molecules. Therefore, we study the angle θ of the OH bonds with the surface normal, as depicted in Fig. 7.3. To be more precise, the ensemble average $\langle \cos\theta \rangle$ as a function of z is calculated to visualize the influence of the interface on this quantity. This has also been done, for instance, by Nagata *et al.* [79].

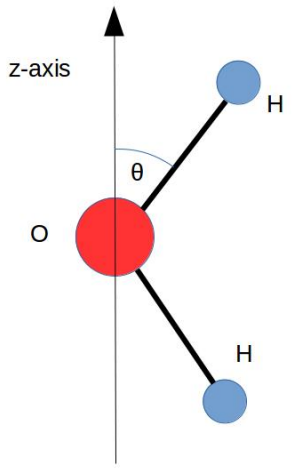


Figure 7.3.: Orientation of the OH bonds relative to the surface normal (z-axis).

7.3.4. OH bond length and HOH bond angle

The flexible geometry of water molecules during a MD simulation is continuously changing. Probability distributions of the OH bond length and HOH angles can

be easily created. We also analyze the average OH bond length as a function of z. If we perform simulations with the SPC/F model and with its NNP counterpart, differences of the respective distributions shall be minimal. Otherwise, this is a good indicator that the NNP does not accurately reproduce the reference SPC/F model. Moreover, we regularly check if one of the bond lengths or the bond angle of a single molecule takes on suspiciously high or low values. Whereas this should not happen under any circumstances during simulations with the NNP trained on the SPC/F model, this may happen in the case of the DFT-based NNPs due to the possibility of bond breaking. However, also then it is not desired because we do not specifically train the NNP on such configurations.

8. Lennard-Jones liquid-vapor interface

Before we tried to create a NNP for the empirical SPC/F water model suited for liquid-vapor interfaces, we were concerned with a simple LJ fluid. The LJ (12,6) potential of a collection of N atoms is

$$V = \sum_{i}^{N-1} \sum_{j>i}^{N} 4\epsilon \left(\frac{\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^{6}}{r_{ij}^{6}} \right), \tag{8.1}$$

where r_{ij} is the distance between atom i and atom j and ϵ, σ are parameters. In the following section we will show that it was an easy task to fit a NN to the PES of a system of atoms interacting via a simple LJ potential. The question we are investigating here is: How accurately can we reproduce the density profile and surface tension of the liquid-vapor interface by replacing the LJ potential by a NNP? A more general formulation is: Does the replacement of a simple short range and non-bonded potential by a precisely trained NNP substantially change macroscopic and surface-specific properties obtained by NNP simulations?

8.1. Creating the training set

We generated five MD trajectories at different temperatures of a liquid-vapor interface of argon. We chose temperatures of 85, 95, 105, 115 and 125 K. The simulations were performed in the NVT ensemble with the aid of a Nosé-Hoover thermostat with a relaxation time of 0.5 ps. For each trajectory the atoms were initially enclosed in a box of dimensions $L_x \times L_y \times L_z = 33.54$ Å $\times$ 33.54 Å $\times$ 100.62 Å with periodic boundary conditions. We started with a cube of 216 fcc unit cells (864 argon atoms) taking up 33.54 Å $\times$ 33.54 Å $\times$ 33.54 Å on the bottom of our simulation box. This corresponds to a density of 1518.97 kg/m^3 or 0.9 in reduced units for Argon within the fcc crystal, which lies beyond the liquid coexistence density for the lowest temperature we imposed on the system. We chose a short cutoff radius of 8.5 Å and a timestep of 5 fs. The equations of motions were integrated with the velocity Verlet algorithm. After an equilibration time of 2 ns we found a stable liquid-vapor interface. During a subsequent production period of 10 ns one configuration was saved every ps. The instantaneous value of the surface tension was calculated every 10th time step and an averaged value of the last 200 calculations was written to disk every 2,000 time steps. Since we are only interested in a comparison between the MD simulation based on the exact LJ potential, on the one hand, and a NNP

simulation, on the other hand, we were not applying any corrections to the surface tension. Finally, the reference data set was created from these five saved trajectories by randomly choosing 40 configurations from each of them. We split this data set into two parts, a training and a test set. The training set contained 185 randomly chosen configurations and the remaining 15 configurations were used as test set.

8.2. Training of the neural network potential

Since the system contains only argon atoms, we needed just one set of SFs. We chose the first 11 Legendrian SFs ($0 \leq m \leq 10$) of Section 4.1.4 and the cutoff function $f_{c,2}$ with a cutoff radius $R_c = 8.5$ Å, which is the same cutoff radius used in the MD simulations for producing the training data. The two cutoff radii were chosen identically in order to show that potential discrepancies between the results of the MD and NNP simulations do not arise from problems due to the necessary NNP cutoff radius discussed in Section 7.1.1 but rather from other possible limitations of NNPs related to interfaces. We used a NN structure of two hidden layers containing

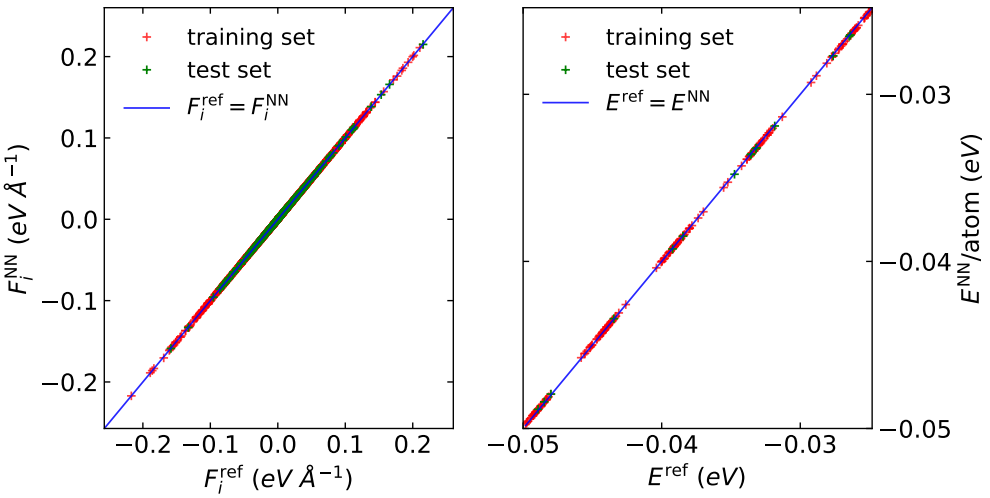


Figure 8.1.: (Left) Prediction of force vector components against their reference values. In this plot the three components of forces from 2000 randomly chosen atoms from the training set and 175 atoms from the test set are shown. (Right) Predicted energies of 200 and 12 configurations from the training and test set, respectively, against their reference values.

15 and 10 neurons, respectively. Since we employed 11 SFs, our identical atomic NNs consist of 11 input neurons, 15 neurons in the first, 10 neurons in the second hidden layer and one output neuron. Whereas the output neuron and those of the input layer just apply the identity function as activation function, the two hidden layers used a sigmoid function. Additionally, as usual, each node in the two hidden layers as well as the output node are connected to a bias node.
As optimization method we employed the MSEKF (Eqs. 2.82-2.85) with 8 streams. All elements of all $\boldsymbol{Q}_k$ were set to 0, for $\boldsymbol{P_0}$ we chose a the unity matrix multiplied

by 100, $\lambda_0 = 0.995$ and $\mu = 0.9987$. During the training procedure we used about 15,000 forces and only 200 energies. The initial weights were randomly drawn from a uniform distribution on the interval [-1,1] and we used $\beta = 50$ (Eq. 4.28).
The RMSE for energies and single force components was ≈ 0.05 meV per Argon atom and ≈ 0.27 meV/Å, respectively. In contrast, the mean potential energy per atom was ≈ -38.34 meV and the mean absolute value of single force components was ≈ 21.88 meVÅ. The precision of the NNP for the configurations in the training and test set is depicted in Fig. 8.1. Moreover, we examined the mean error of forces

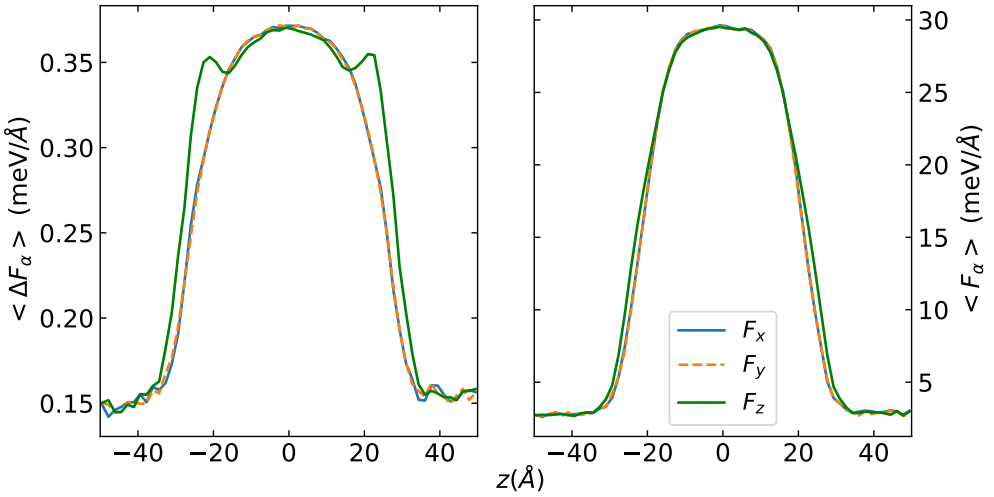
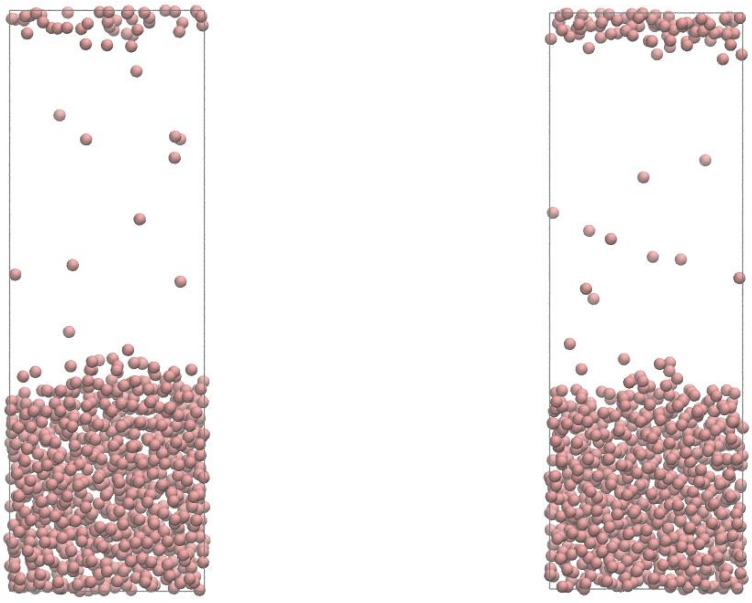


Figure 8.2.: Mean error (left) and the mean absolute reference values (right) of the x, y and z components of forces as a function of z, during the NNP simulation at 115 K. The origin is set to the center of mass of the system for each configuration.

for x, y and z components as a function of the position on the z-axis . Only near the interface there is a marginal difference between the x and y components and the z component, as can be seen in Fig. 8.2.

8.3. Simulation results

With the readily trained NNP, described in the last section, we produced five MD trajectories at exactly the same temperature values which we used for creating the reference data set (85K, 95K, 105K, 115K, 125K). For these trajectories we chose the same settings as for the MD simulations that generated the training data regarding the timestep, the number of atoms, the box dimensions, the initial configuration, the equilibration times as well as the thermostat to simulate the NVT ensemble (see Section 8.1). Furthermore, the same intervals for saving configurations and instantaneous values of the surface tension were chosen. In Fig. 8.3 two snapshots of the simulations using the LJ potential and the NNP can be seen. From these two snapshots a qualitative difference between the original LJ potential and its NNP approximation can not be noticed. We computed the density profiles from



(a) Original LJ potential (b) NNP

Figure 8.3.: Two snapshots of the simulation for T=85K with the LJ potential and its NNP.

each saved trajectory and determined the saturated liquid and vapor densities as described in Section 7.3.1. The resulting mass density profiles for the LJ potential and NNP can be found in Fig. 8.4. The mass density profiles of the NNP simulations are in very good agreement with those of the original LJ potential, but for increasing temperature the used NNP tends to give rise to slightly higher liquid densities and lower vapor densities. By performing a fit to the mass density curves based on the method given in Section 7.3.1, we obtained the saturated liquid and vapor densities, which can be found in Fig. 8.5. Additionally, we computed the ensemble average of

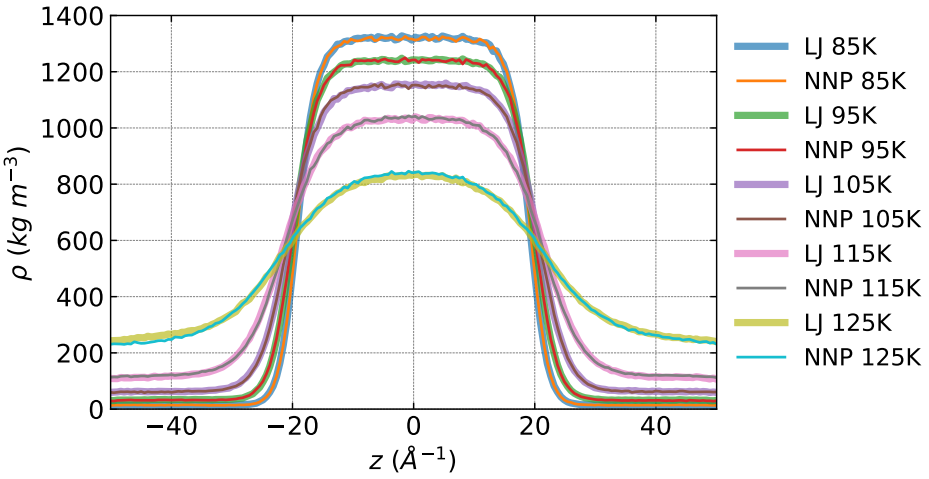


Figure 8.4.: Mass density profile of simulations based on the NNP and its reference LJ potential for different temperatures.

the surface tension. The deviations for the surface tension between the LJ potential and the NNP are rather small as summarized in Table 8.1.

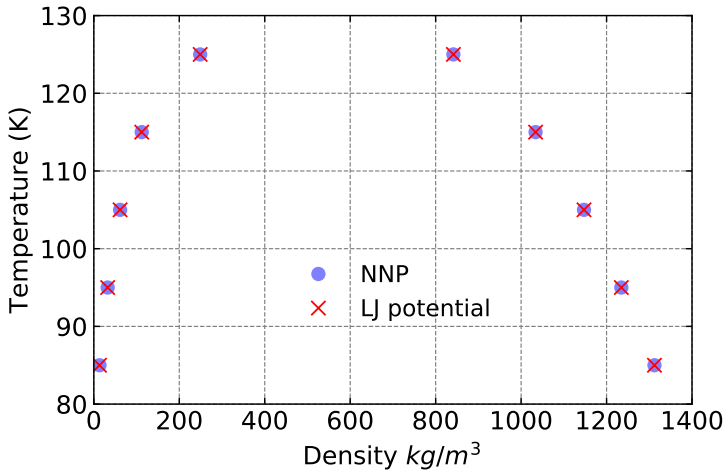


Figure 8.5.: Saturated densities ρ_L and ρ_V for the coexisting liquid and vapor phases of Argon at different temperatures for the exact LJ potential and its NNP. The error bars are much smaller than the symbols.

Temp. (K)	LJ γ (mN/m)	NNP γ (mN/m)
85	8.29(11)	8.16(13)
95	6.00(9)	6.05(9)
105	3.88(8)	4.02(10)
115	2.18(10)	2.03(9)
125	0.54(9)	0.75(10)

Table 8.1.: Surface tension γ for the LJ potential and its NNP for different temperatures.

By looking at these results, we can clearly conclude that the NNP is capable of approximating the LJ potential in such a way that the ensemble averages of the interface-specific observables we computed can be reproduced with a high degree of accuracy. However, it must be noted that we used a very short cutoff distance (8.5 Å) for the classical MD simulations with the LJ potential and, furthermore, we used exactly the same cutoff radius for the SFs of the NNPs.
In many cases we have to deal with long-range interactions, which are naturally harder to handle for a NNP with its cutoff radius. Moreover, the equality of the LJ and the NNP cutoff distance may facilitate an accurate approximation of the NNP. If the cutoff radius of the LJ potential was larger than that of the NNP, this would clearly give rise to some inaccuracies.

9. SPC/F water-vapor interface

From the last chapter we may conclude that for simple potentials like the LJ potential a NNP can successfully reproduce the density profile and the surface tension of a liquid-vapor interface. The SPC/F water model (see Section 5.2.2) also includes LJ interaction between oxygen atoms but, additionally, Coulomb interaction between all atoms of different water molecules is present. This is especially critical because Coulomb interactions are long-ranged and NNPs generally use a cutoff radius (see Section 7.1.1). Besides that, intramolecular forces are necessary in this flexible simple point charge model. Even though these intramolecular forces originate from simple harmonic potentials, it must be checked that the NNP basically reveals the same distributions of bond lengths and bond angles during simulations compared to the reference model. Furthermore, it is important to observe if bond breaking occurs because the SPC/F model does not allow for this.

9.1. Creating the training set

We created two different training sets. The second training set was derived from the first one.

9.1.1. Training data set 1

To create the first training set, we performed simulations of interfaces at different temperatures in the canonical ensemble and bulk simulations for different combinations of temperature and pressure in the isothermal-isobaric ensemble. In the first case a Nosé-Hoover thermostat with relaxation time of 50 fs was used and for the bulk simulation temperature and pressure were controlled by a Nosé-Hoover thermostat and barostat with relaxation times of 50 fs and 0.5 ps, respectively. To this purpose, the equations of Shinoda *et al.* [80] were used by the LAMMPS program. The Velocity Verlet algorithm was employed to integrate the equations of motions with a timestep of 0.5 fs. Long-range forces were computed by the particle-particle particle-mesh (PPPM) method [81] with a relative tolerance for forces of 10^{-5}. The real-space cutoff was chosen to be 9.2 Å, the same as the cutoff for LJ interactions. Bulk simulations were started from a cube of 216 regularly spaced and identically oriented water molecules, whereby O-atoms formed a simple cubic crystal structure. Both O-H distances in these identical molecules were 1.0 Å and the H-O-H angel was 109.47°. The density of this cube with an edge length of 18.6 Å was 1003.31kg/m^3.

The dimensions of the periodic box were set to $L_x \times L_y \times L_z$=18.6×18.6×18.6 Å^3. We created trajectories for each of the following 16 combinations of temperature and pressure values $300, 350, 400, 450$ K and $1, 10, 100, 1000$ atm, respectively. After 0.5 ns of equilibration we saved one configuration every 0.5 ps to disk over a simulation time of 1.5 ns.
For the simulations of the interface the aforementioned cube of regularly spaced identical water molecules was placed at the bottom of a periodic simulation box with the same x and y dimensions but with a three times bigger z dimension. This amounts to a box of size 18.6×18.6×55.8 Å^3 . After 1 ns of equilibration time at 300K we saved the last configuration as initial configuration for all subsequent simulations of interfaces for both cases, simulations with the SPC/F model and the NNP. For each of the aforementioned temperature values imposed on the system, one trajectory was created. Therefore, after an additional equilibration time of 0.5 ns, one configuration was saved every 0.5 ps over a time span of 5 ns. An averaged value of the surface tension was saved every 1000 time steps, whereas this average was based on the last 100 evaluations from each 10th timestep.
For the first training set we randomly chose 5 configurations from each bulk trajectory and 60 configurations from each interfacial trajectory. Thus, the first training set contains 80 and 300 configurations from the trajectories of the bulk and interfacial system, respectively. Several simulations using NNPs which where trained based on these configurations showed that it regularly occurred that water molecules were split or permanently formed an unnatural H-O-H angle. So we decided to distort the configurations of the first training set to obtain a second training set which could possibly improve the stability of subsequent NNP simulations.

9.1.2. Training data set 2 (distorted)

In order to create the second training set, we used all 380 configurations of the first training set. Each atom of each configuration was displaced in a random direction. The distance how far an atom was displaced should be drawn from a Gaussian-like distribution, at least in the proximity of the highest density region. This enabled us to control the mean and the variance of the distribution of displacement distances. However, an exact Gaussian distribution was not desired due to negative distances which could have been drawn. So we settled on the following distribution:

$$f(r) = \frac{3r^2\lambda}{a^3} \exp\left[-\lambda\left(\frac{r}{a}\right)^3\right]. \tag{9.1}$$

For each atom we first randomly chose a direction from the uniform distribution on the surface of the unit sphere by drawing three variables, U, V, W, from the standard normal distribution

$$U, V, W \sim \mathcal{N}(\mu{=}0,\, \sigma^2{=}1). \tag{9.2}$$

We grouped these random variables together to a random vector $\vec{X} = (U, V, W)$. Then we normalized this vector

$$\vec{X}_{\text{unit}} = \vec{X}/\sqrt{U^2 + V^2 + W^2}. \tag{9.3}$$

The point the resulting random vector $\vec{X}_{\text{unit}}$ is pointing to is uniformly distributed on the surface of the unit sphere [82]. In order to obtain a random variable R, distributed as in Eq. 9.1, we first drew

$$X \sim Exp(\lambda) = \lambda e^{-x\lambda}, \tag{9.4}$$

where $Exp(\lambda)$ is the exponential distribution, to subsequently find R by

$$R = aX^{1/3}. \tag{9.5}$$

The final random displacement vector, $\vec{R}_{\text{disp}}$, can then be expressed as

$$\vec{R}_{\text{disp}} = R \cdot \vec{X}_{\text{unit}}. \tag{9.6}$$

The proof that the length of the displacement vector R is distributed as in Eq. 9.1 can be easily shown and is given in Appendix C. We determined different parameters for the radial displacement distributions for oxygen and hydrogen atoms. We chose $a_O = 0.047$Å and $a_H = 0.095$ for oxygen and hydrogen atoms, respectively, and $\lambda = 5.5$ for both atom types. In Fig. 9.1 the distribution of radial displacements is shown. It can be seen that the shape of Eq. 9.1 closely resembles a Gaussian with appropriate parameters. Only for small values of r there is a clear difference, what is desired to avoid negative displacement lengths. The mean of the Gaussian which fits Eq. 9.1 the best can be approximately found by calculating the maximum of $f(r)$,

$$\mu \approx f_{\max} = \sqrt[3]{\frac{2a^3}{3\lambda}}. \tag{9.7}$$

The modification of the training set in this manner gave rise to stronger forces, as depicted in Fig. 9.2. Similarly, the total energy values of the training configurations were shifted to higher values by an average quantity of 20 eV. This procedure could have been done in many other ways. For instance, a different distribution or only a

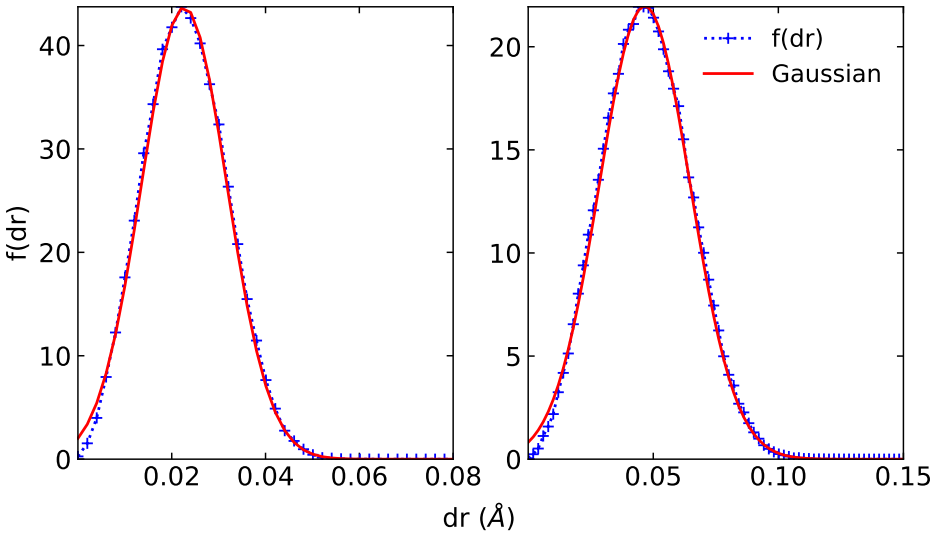


Figure 9.1.: Distribution of radial displacement of O-atoms (Left) and H-atoms (Right), based on the probability density function of Eq. 9.1 with parameters $a_O = 0.047$, $a_H = 0.095$, $\lambda = 5.5$. Parameters of the Gaussians are $\mu_O = 0.023$, $\sigma_O = 0.09$ and $\mu_H = 0.047$, $\sigma_H = 0.018$ for O- and H-atoms, respectively.

certain percentage of distorted configurations or molecules are reasonable. A limiting case could be a resulting training set, which is modified so strongly that it has nothing in common with physically relevant configurations. A training set like this, which is not extended with other physically more relevant configurations, could substantially decrease the accuracy of the NNP.

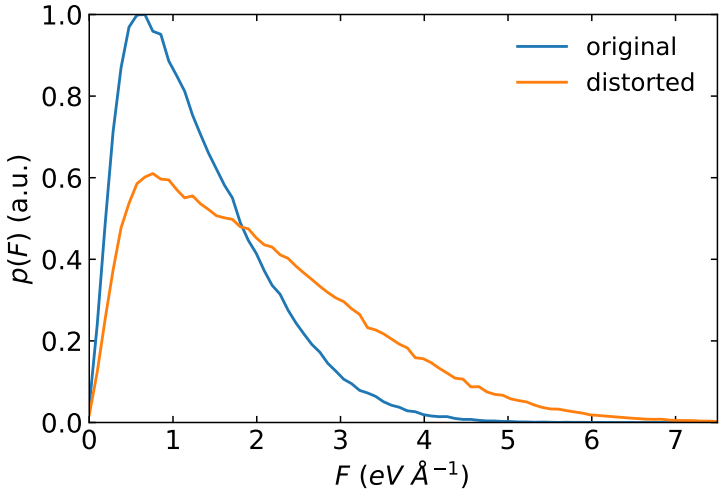


Figure 9.2.: Distribution of the magnitude of force vectors $F = |\boldsymbol{F}|$ before and after the distortion procedure was applied to configurations of the original training data set.

9.2. Training of the neural network potential

We employed three different sets of SFs. The first set contains exactly the SFs which were used in the work of Morawietz *et al.* [15] to develop NNPs for the simulation of bulk water. The exact list of SFs and parameters of this set can be found in the Appendix in Tables D.1 and D.2. In order to test the newly devised multi-radial SFs (see Section 4.1.5), we use a second set. This set was constructed by replacing the angular SFs of the first set with the same number of multi-radial SFs, given in Tables D.3 and D.4. Finally, we composed a third set by appending some multi-radial and the Legendrian SFs (see Section 4.1.4) for m=0 to the first set (Tables D.5 and D.6). For all SFs of all sets we utilized the the cutoff function of type 2, given in Eq. 4.5.

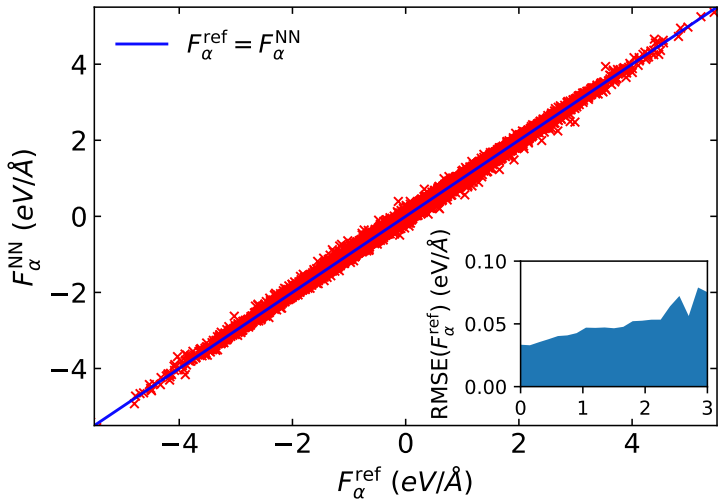


Figure 9.3.: Reference values of roughly 80,000 force components against their prediction by the NNP based on training set 1 and SF set 1 and (insert) the RMSE as a function of the absolute value of the reference force component. All force components are taken from training set 1. The corresponding RMSE of the shown data points is 39.4 meV/Å.

As activation function of each hidden layer we employed the sigmoid function. As usual, activation functions of the input layer and the single output neuron were chosen to be the identity function. SF values were scaled in such a way that the minimum and maximum value of each SF in the training set was -1.0 and 1.0, respectively. Initial weight parameters were drawn from the uniform distribution on $[-0.1, 0.1]$. We again used the MSEKF and chose the same settings and parameters as for the LJ potential (see Section 8.2). For different NNPs we tried to interchangeably apply two different NN architectures regarding the hidden layers. On the one hand, we used 2 hidden layers, each holding 25 neurons. On the other hand, 3 hidden layers consisting of, in order, 15, 8 and 4 neurons were employed. However, within the same NNP the architecture of the atomic NNs for oxygen and hydrogen atoms were always identical, except for the number of neurons in the input layer.

energies (meV)	SF set 1	SF set 2	SF set 3
ref. data set 1	0.58	0.38	0.54
ref. data set 2	0.59	0.49	0.57
forces (meV/Å)	SF set 1	SF set 2	SF set 3
ref. data set 1	39.4	35.0	39.0
ref. data set 2	46.7	44.7	44.5

Table 9.1.: RMSE of energies per H_2O and single force components of NNP trainings for different training data and SF sets. Atomic NNs using 2 hidden layers, each containing 25 neurons after 10,000 energy and 8,000 force component updates. For the test set the RMSEs were always of similar size.

In the case of reference data set 1, the undistorted reference data set, we achieved promising results. This is graphically represented in Fig. 9.3. In Table 9.1 the RMSEs of energies/H_2O and of single force components for all SFs sets and both training sets can be found. These RMSEs were evaluated by employing the test set of the training data.
However, during MD simulations, the NNPs based on reference data set 1 turned out to be error-prone. It sometimes happened that molecules became defectively deformed in a way that their HOH bond angle was between 30 and 60 degrees. We also observed OH bond lengths of more than 3 Å. The total forces acting on the atoms of such deformed molecules was often around 0 $eV/$Å. Due to this, the simulations did not always crash and often a trajectory could be produced for several hundred picoseconds or even a few nanoseconds. It is interesting to note that for a small number of NNPs trained on data set 1, oxygen atoms which lost one hydrogen atom could even form a new bond with other lost hydrogen atoms. This is remarkable because such mechanisms are not possible within the framework of the SPC/F model. Based on the extent and number of defective deformations the observables, such as the density profile and the surface tension, were strongly fluctuating and the properties of the SPC/F model could not be reliably reproduced. The main reason behind these instabilities probably are the poor extrapolation properties of the NNP regarding the intramolecular harmonic potentials. Therefore, we decided to distort the first reference data set as described in Section 9.1.2 to obtain a second reference data set. In Fig. 9.4 it is shown how training on reference data set 2 slightly improves the harmonic bond angle potential for an isolated molecule. The extent of improvement is only minor and it shall be noted that angles $\theta_{\mathrm{HOH}} < 80$ and $\theta_{\mathrm{HOH}} > 130$ occur extremely rarely. Nevertheless, in the presence of other molecules, the NNP does not explicitly evaluate the harmonic potentials for isolated molecules and the level of improvement by the distortion of the training data may be higher in that case.

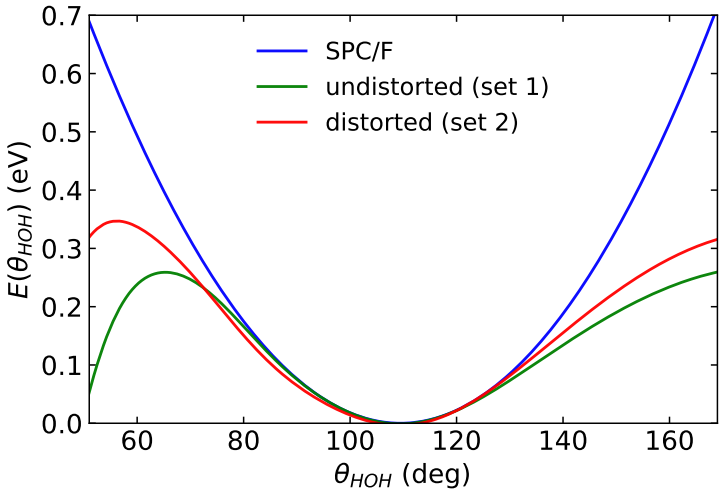


Figure 9.4.: NNP prediction of the harmonic potential regarding the HOH bond angle in SPC/F water molecules. We obtained these curves by varying the bond angle of a single H_2O molecule with equilibrium bond lengths in an empty box and predicting its energy value by means of the readily trained NNP based on SF set 1 and training set 1 or 2. Training on distorted configurations resulted in a slightly better approximation of the harmonic angle potential of the SPC/F model for large and small angles.

9.3. Simulation results

The NNPs trained on reference data set 2 were stable and we utilized exactly such a NNP to perform simulations. We employed SF set 3 and in order to achieve a faster NNP both atomic NNs for hydrogen and oxygen consisted of 3 hidden layers with, in order, 15, 8 and 4 neurons. This 3-layered architecture reduces the number of weights from 1651 to 754 for the case of oxygen compared to 2 hidden layers each consisting of 25 neurons. We performed approximately 200 energy and 2500 force component updates. Other details regarding the training of this NNP are identical to those described in the last section. For the training set the RMSE of energies per H_2O and force components was 0.86 meV and 48 meV/Å, respectively. The RMSEs for the test set were of similar size. We performed 5 simulations with slab geometry for exactly the same temperature values as for the interface trajectories which we used for creating the reference data. All simulation details regarding the initial configuration, equilibration time, the thermostat, box dimension remained unchanged and are described above (see Section 9.1). Moreover, the same timestep of 0.5 fs was chosen and over a production period of 5 ns every 0.5 ps one configuration was saved. The instantaneous value of the surface tension was computed every 10th time step and an averaged value of the last 100 evaluations was written to disk every 1000 time steps. It was of interest to us whether the RMSE for energies and forces during the simulation were comparable to the training errors since the application of the NNP introduces inaccuracies. During NNP simulations, these inaccuracies can

possibly lead to configurations substantially different from the training and test set. The NNP might not be properly trained for these configurations and an increasing error could be the consequence. Therefore, we calculated the RMSE for forces of all saved configurations of the first nanosecond. In Fig. 9.5 the running average of the RMSE of force components of the current as well as the last and next 37 saved configurations is shown. It can be seen that the RMSE is not increasing with simulation time and it is of similar magnitude to that of the training and test set. These results let us assume that our NNP was appropriately trained. The offset between different temperatures is reasonable since the mean magnitude of force components increases from 0.59 eV/Å for 300 K to 0.73 eV/Å for 460 K.

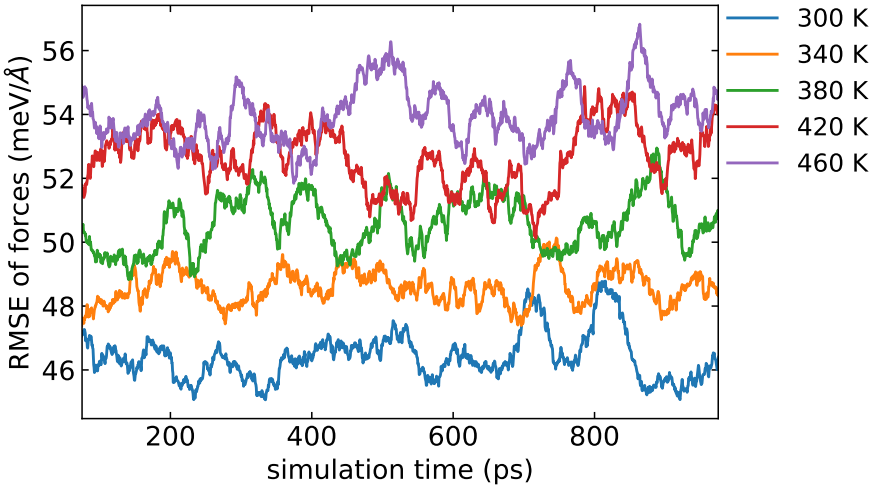


Figure 9.5.: Running average of the RMSE of force components of the current configuration as well as the last and next 37 saved configurations during the NNP simulations.

When we examine the mean error of forces for x, y and z components as a function of the position on the z-axis, we see a different functional dependence for the x and y components on the one hand and the z component on the other hand. This situation is depicted in Fig. 9.6. A difference in the functional dependence of the mean error for the z component is expected due to symmetry reasons. However, in Fig. 9.6 one can see that the mean z component within the liquid phase is approximately the same as for x and y components but the mean error is around 30% higher for the z component. Interestingly, this difference is smaller in the vicinity of the interface. In the case of the LJ potential we did not find such a separation of the mean error as a function of z for different force components, as shown in Fig. 8.2. This might possibly indicate that long-range interaction is the reason for the bigger RMSE of the z component of forces. For completeness, in the Appendix the RMSEs of a function of z for the simulations at 380 K and 460 K is shown in Fig. E.3.

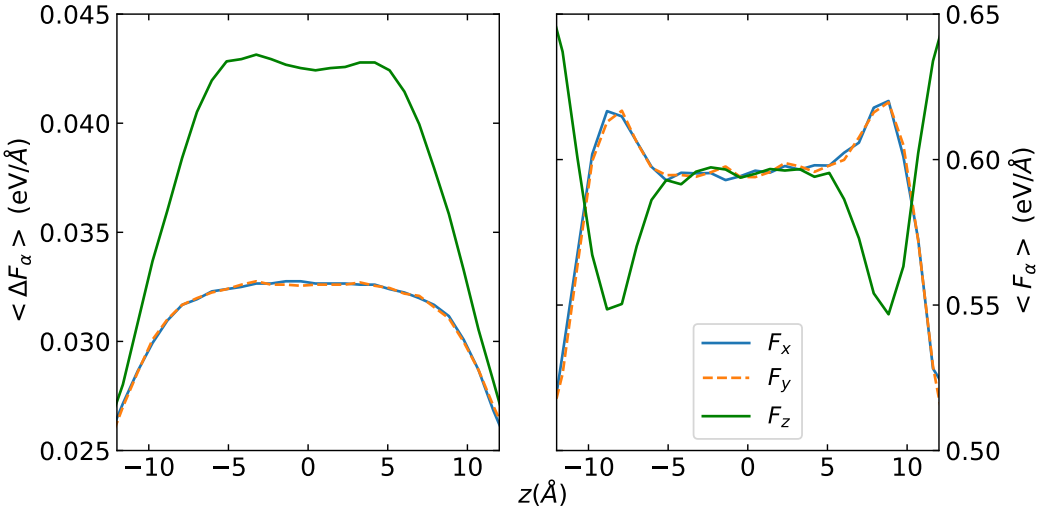


Figure 9.6.: Mean error (left) and the mean absolute reference values (right) of the x, y and z components of forces as a function of z during the NNP simulation at 300 K. The origin is set to the center of mass of the system for each configuration.

9.4. Comparison between SPC/F and NNP simulations

As mentioned before, we performed simulations with the SPC/F model for creating the training and test sets. In this section we compare the results from these simulations with those employing the NNP. First, we analyzed the geometry of water

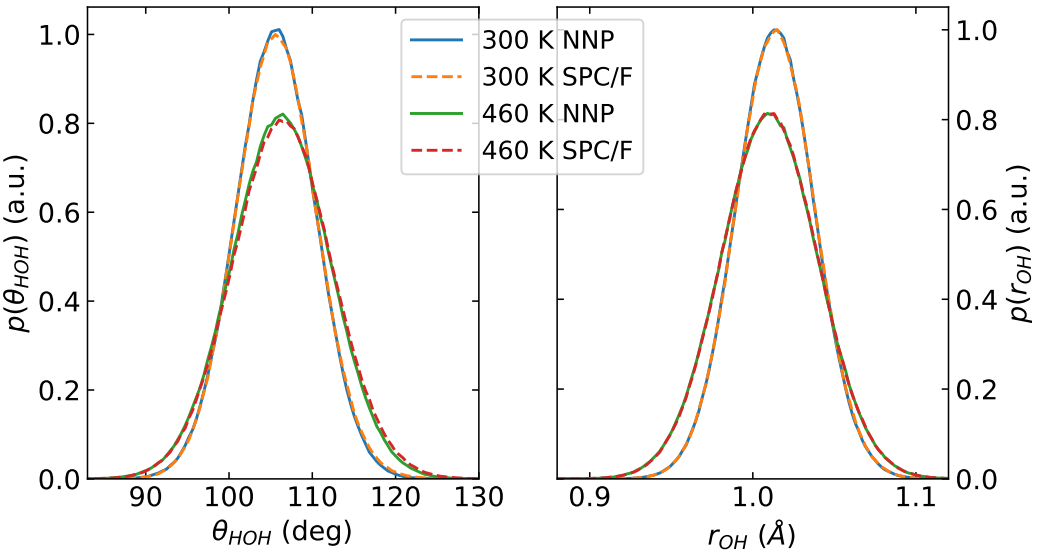


Figure 9.7.: Comparison of the distributions of the bond angle θ_{HOH} and bond length r_{OH} of water molecules between simulations with the SPC/F model and its NNP approximation.

molecules for both potentials. Therefore, we computed the distribution of OH-bonds and HOH-angles. The distributions of both quantities coincide to a very high degree of accuracy, as depicted in Fig. 9.7. We fitted Gaussians to these curves for

all temperatures. In Appendix E the fitted mean and variance of the distributions of the OH-bonds and HOH-angles are shown. However, we are more interested in interface-specific properties. Hence, we computed the mass density profiles from the saved configurations. The resulting curves can be seen in Fig. 9.8. Especially for lower temperatures we found a high level of consistency. For simulations at higher temperatures substantial deviations become visible. By performing a fit to the mass

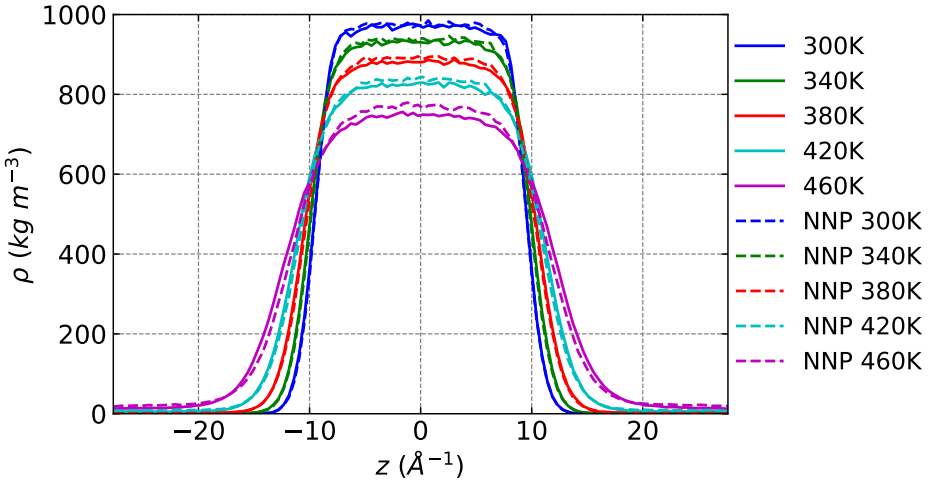


Figure 9.8.: Mass density profile for the NNP and its reference potential, the SPC/F model, for different temperatures.

density profiles by means of the method described in Section 7.3.1, we obtained the densities of the liquid and vapor phases, given in Table 9.2. Except for the highest temperature, 460 K, the deviations between the SPC/F model and its NNP approximation are not larger than 12.5 kg/m^3 (1.7%) and 2.7 kg/m^3 (50%) for the liquid and vapor densities, respectively. For 460 K the deviations are already 25.5 kg/m^3 (3.4%) for the liquid and 7.11 kg/m^3 (35%) for the vapor phase. The NNP overestimates the liquid density and underestimates the density of the vapor phase for all temperatures.

	$\rho_L(kg/m^3)$		$\rho_V(kg/m^3)$	
Temp. (K)	SPC/F	NNP	SPC/F	NNP
300	969.0(3.2)	974.6(2.5)	0.079(24)	0.146(24)
340	927.0(1.9)	934.7(2.1)	0.514(36)	0.444(37)
380	876.5(1.6)	888.9(1.7)	2.097(66)	1.790(65)
420	821.7(1.4)	835.9(1.3)	5.276(98)	7.95(11)
460	747.8(1.4)	773.3(1.1)	13.08(18)	20.19(16)

Table 9.2.: Liquid and vapor densities, ρ_L and ρ_V, for the simulations with the SPC/F model and its NNP approximation.

Another observable we calculated for both the SPC/F and NNP simulations was the surface tension. The results are plotted in Fig. 9.9. The exact values and their

standard deviations are listed in Appendix E. The biggest discrepancy can be found for $T = 380$ K which amounts to ≈ 1.91 mN/m (5.9%), whereas the standard deviation lies around 0.40 mN/m for both, the simulation with the NNP and the SPC/F model. The mass density profile and the surface tension are important measures for

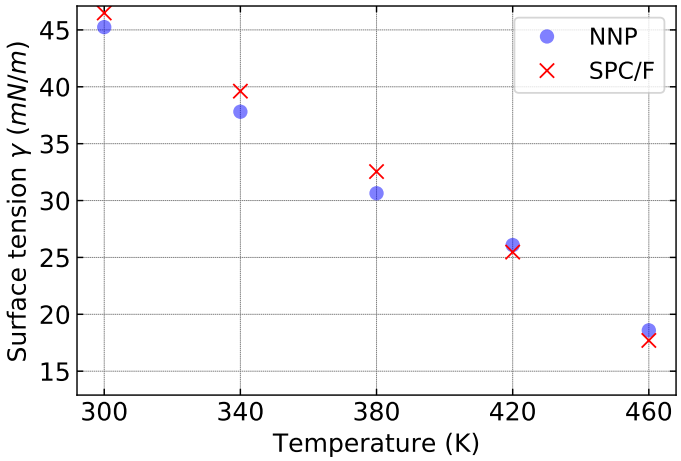


Figure 9.9.: Surface tension for the SPC/F model and the NNP for different temperatures. The standard deviation is smaller than the symbols. The exact values and their standard deviation can be found in Table E.1.

the description of interfaces, but we can also look at the structure of the interface on a microscopic level. Therefore, we computed the average OH-bond length of water molecules as a function of their position on the z-axis. For these calculation we set the the origin to the center of mass of the system. In Fig. 9.10 this quantity is shown for 300 K and 460 K. Within the liquid bulk phase the mean OH bond distance reaches its maximum and is almost constant. In the vicinity of the interface this value starts to continuously decrease with increasing magnitude of z. From the NNP simulations the same average OH-bond length within the liquid bulk phase are obtained as for the case of the reference SPC/F model, but the curves slightly deviate in the vicinity of the interface.

Next, we analyzed the orientational distribution of water molecules as a function of their position on the z-a xis (see Section 7.3.3). Again, for every considered configuration the origin is set to the center of mass of the system. For both the NNP and the SPC/F model the situation is depicted in Fig. 9.11. The NNP simulations could reproduce the most essential properties of the reference distribution. It can be seen that molecules of the approximately two outermost layers of the liquid slab exhibit a preferential orientation, while molecules within the bulk phase are, on average, randomly orientated. Molecules of the outermost layer are most likely oriented in such a way that one of their OH-bonds points in the direction of the surface normal, whereas molecules of the second outermost layer tend to avoid having OH-bonds pointing in this direction.

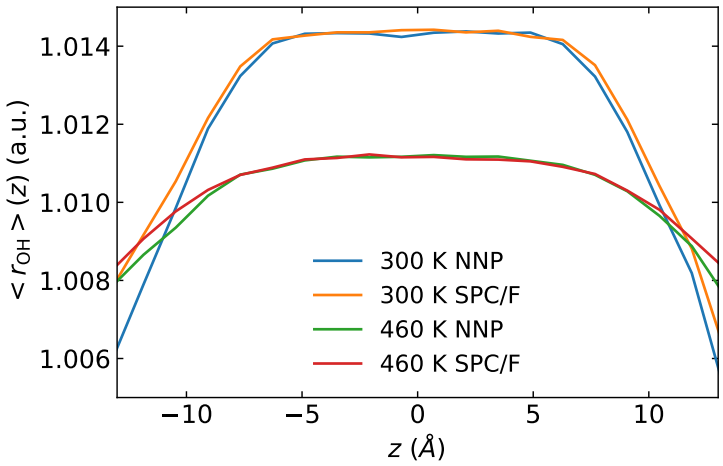


Figure 9.10.: Average OH-bond length as a function of z, where the origin is the center of mass of the system.

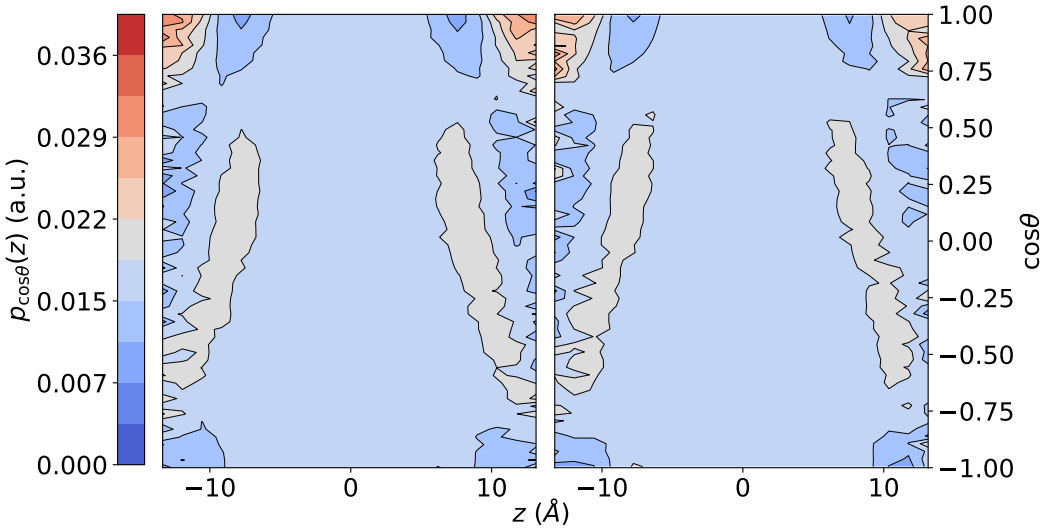


Figure 9.11.: Distribution of the angle θ between the OH bonds and the surface normal as a function of the position on the z-axis for the reference SPC/F model (left) and NNP (right) at 300 K.

10. RPBE-D3 water-vapor interface

The interesting use case of NNPs clearly is the representation of very complex functions like the PES of water using DFT. For this thesis, we employed the RPBE functional [18] with VdW corrections, namely the D3 method [19]. The disadvantage of using a NNP is the question whether the ideally small error of the training and test set can still give rise to substantial errors regarding ensemble averages of observables in subsequent simulations. To address this question, we have shown in the last chapter that a NNP is capable of dealing with the liquid-vapor interface of SPC/F water including long-range interactions since the results from NNP simulations were in good agreement with them of the reference SPC/F model. Thus, the following results can be trusted within a certain, yet unknown, degree of accuracy.

10.1. Creating the training set

Unlike for the case of SPC/F water, we can not easily generate a training set by performing simulations based on DFT calculations. Since we were provided with the bulk water training set of Morawietz *et al.* [15], which does not include interfacial configurations, we extended this training set by a two-step process. First, we added 190 interface configurations of the distorted training data set of the SPC/F simulations (see Section 9.1.2), recomputed by DFT. Since these configurations naturally differ in some points from configurations one would obtain from AIMD simulations based on DFT and the RPBE functional, we again added training data in a second step. Therefore, we trained a first NNP based on this preliminary training set and performed NNP simulations of systems with slab geometry at different temperatures in hopes of generating configurations being very similar to those we would obtain from AIMD simulations. From these trajectories we randomly selected additional 54 configurations to finally complete our training set. We thus added 244 configurations each consisting of 648 atoms to the original training set, which corresponds to 474,336 additional force components.
The original training data set of Morawietz *et al.* contains around 7,200 periodic configurations with a total of approximately 590,000 atoms. The training set was generated by an iterative process, whereas an initial training set was extended stepwise. The initial training set contained liquid and crystalline configurations. The initial liquid configurations were obtained from MD simulation based on the SPC/Fw model [83], whereas the crystalline part of the initial training set was generated by DFT relaxations. Additional configurations were obtained by distorting the fully

relaxed structures and by performing simulations based on preliminary NNPs. All configurations were, of course, recomputed by the reference method before adding them to the training set. This procedure was repeated until the NNPs were converged. More information about the original data set can be found in the Appendix of the paper of Morawietz *et al.* [15].
The additional DFT computations of energies and forces, which were necessary due the extension of our training set, were done by Marcello Sega. For this purpose, the code FHI-aims [84] was used with the same settings as for the original training set of Morawietz *et al.* in order to preserve consistency. FHI-aims utilizes numeric atom-centered basis functions. Besides the minimal basis set, the additional basis function sets "tier 1" and "tier 2" were used. The desired vdW corrections were realized by Grimme's D3 method [19] employing the zero-damping scheme while neglecting three-body contributions.

10.2. Training of the neural network potential

For the training of the NNP we employed SF set 1 (see Appendix D). As cutoff function we chose $f_{c,2}$, given in Eq. 4.5. For the SPC/F-based NNP we added some newly developed SFs, which improved the accuracy of the NNP. We could not find the same positive effect for the DFT-based training data. Hence, we preferred to stay consistent with the previous work of Morawietz *et al.* [15] and used exactly the same set of SFs. We employed the sigmoid function as activation function for each

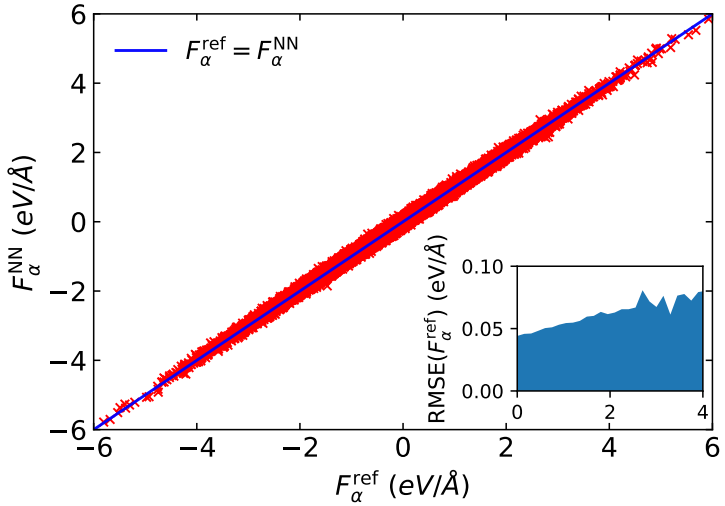


Figure 10.1.: Reference values of roughly 100,000 force components against their NNP predictions and (insert) the RMSE as a function of the absolute value of the reference force component. All force components are taken from 50 interface configuration of the extended training set.

hidden layer and the identity function for the input layer as well as the single output neuron. SFs values were scaled to the interval $[0, 1]$. The number of hidden layers

and their number of neurons was the same for atomic NNs of oxygen and hydrogen, namely 2 hidden layers, each consisting of 25 neurons. Initial weight parameters were randomly drawn from a uniform distribution on the interval [-1,1]. As before, we employed the MSEKF and chose the settings of Section 8.2. During the training procedure we used about 6,000 forces and 6,000 energies of the extended training set and we used $\beta = 1$.

The RMSE of the final NNP for configurations of the original training set was 1.99 meV/H_2O for energies and 0.51 meV/Å for forces, which is almost 0.20 meV/Å lower than the RMSE of forces in the work of Morawietz *et al.*. For the interface configurations in the extended training set, we obtained a RMSE of 0.98 meV/H_2O for energies and 0.50 meV/Å for forces. In Fig. 10.1 the predicted force components of some interface configurations of the extended training set are plotted against their reference values. Additionally, we analyzed the mean error of x, y and z force components as a function of the position on the z-axis for all interface configurations of our extended training set. The result is depicted in Fig. 10.2. Within the liquid phase and close to the interface the mean error of the z component was found to be higher than that of the x and y component. The largest difference was found in the vicinity of the interface and is approximately 5 meV/Å, which is about 11% higher than the respective mean error for the x and y component. In contrast, we have found up to 33% higher errors for the z component for SPC/F based NNP simulations at 300 K, corresponding to a difference of about 10.5 meV/Å. For higher temperatures relative and absolute differences were even higher.

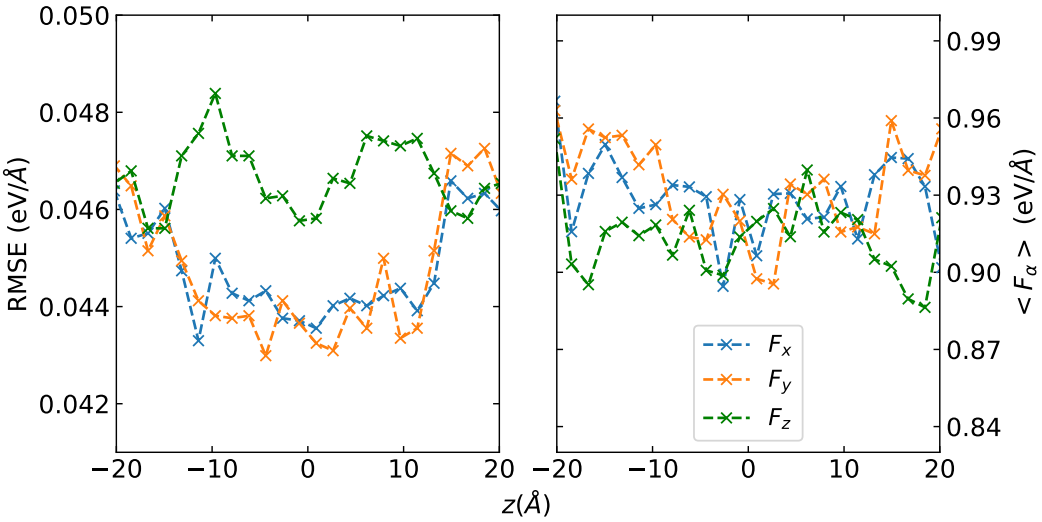


Figure 10.2.: Mean error (left) and the mean absolute reference values (right) of the x, y and z components of forces as a function of z for 244 interface configurations of the extended training set (described in Section 10.1).

10.3. Simulation results

Again, we performed simulation of interfaces by utilizing a slab geometry. We used a periodic box of size $L_x \times L_y \times L_z = 18.6 \times 18.6 \times 55.8$ Å^3 containing 216 water molecules. As initial configuration we utilized the same one as for the case of the SPC/F model (described in Section 9.1.1), which is basically a random configuration of an equilibrated system of slab geometry simulated at 300 K and based on the SPC/F model. The simulations were conducted in the canonical ensemble. To control temperature, a Nosé-Hoover thermostat with a relaxation time of 50 fs was used. We chose a timestep of 0.5 fs. After an equilibration time of 0.5 ns we saved one configuration every 0.5 ps over a simulation time of 5 ns. The instantaneous value of the surface tension was computed every 10th time step and an averaged value of the last 100 evaluations was written to disk every 1,000 time steps.
We generated 6 different trajectories by using a NNP trained on the extended training set at 300 K, 350 K, 400 K, 450 K, 500 K and 550 K. Additionally, we trained NNPs only on the original training set (without interfacial configurations). Surprisingly, we obtained NNPs from these trainings which we could use for simulations, finding stable interfaces between the liquid and the vapor. However, the number of extrapolation warnings per atom and per timestep was around 5, where most of the extrapolation warnings occurred for atoms close to the interface. However, as we have demonstrated in Chapter 6, NNs are not a good method for extrapolation and we thus expect that at least interface-specific results of these simulations substantially differ from that of simulations based on the NNP trained on the extended training set. When we compare the density profiles of NNP simulations based on these two different training sets (see Fig. 10.3), one can see quite good agreement for 300 K but widely differing curves for 550 K. Even though the mass density profiles

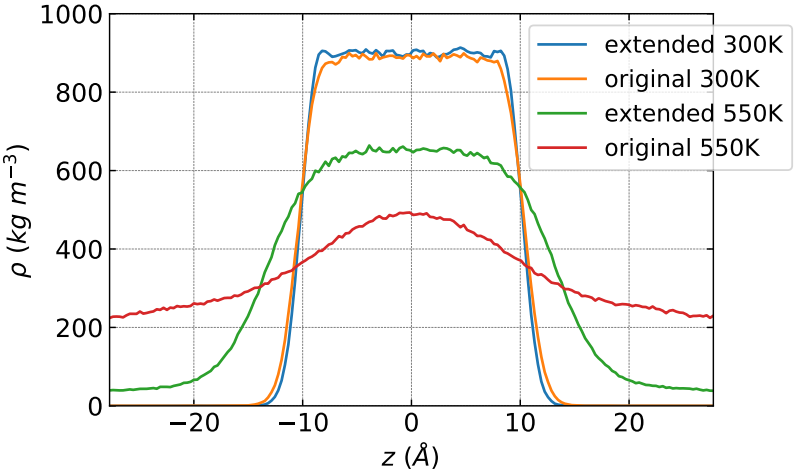


Figure 10.3.: Mass density profiles for the NNPs trained on the original and the extended training set.

at 300 K look similar, the microscopic properties close to the interface are clearly

different. This can, for instance, be seen on the orientational distribution of the OH-bonds (see Fig. 10.4). Including interfacial configurations into the training set gives rise to a noticeable effect on the orientation of the molecules close to the interface. For NNPs trained on the original training set one of the OH-bonds of molecules of the outermost layer preferentially pointed in the direction of the interface normal. For NNPs trained on the extended training set we found that this preferential orientation was less pronounced. Additionally, a second preferential orientation can be observed, that is, water molecules also tend to have both of their OH-bonds pointing away from the interface. OH-bonds of molecules of the second outermost layer are more likely to point in the opposite direction of the surface normal and avoid pointing in the direction of the surface normal. In the case of NNPs trained on the original training set, we only found a notably weaker preferential orientation in the second outermost layer. Another difference regarding the microscopic structure near the interface is the average OH-bond length as a function of the position on the z-axis, plotted in Fig. 10.6. Whereas within the liquid slab the average OH-bond length is of comparable magnitude for the NNPs based on the original and on the extended training set, the functional dependence is substantially different in the vicinity of the interface. Even though simulations based on the NNPs fitted to the original training set exhibited stable interfaces, the inclusion of interface configurations considerably changed microscopic properties close to the interface. Moreover, we want to stress the fact again that the number of extrapolation warnings for the NNPs without interface training data was around 5 extrapolation warnings per timestep and per atom. In contrast the inclusion of the interface configurations strongly reduces the frequency of extrapolation warnings. We only got between 0.0001 and 0.003 extrap-

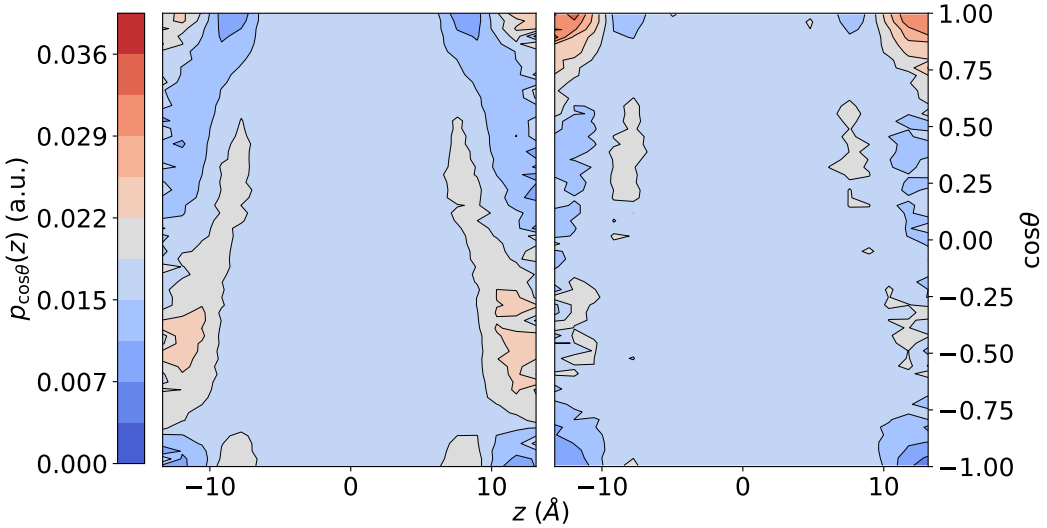


Figure 10.4.: Distribution of the angle between the OH bonds and the surface normal, θ, as a function of the position on the z-axis (origin is set to the center of mass for each configuration) for the NNP based on the extended training set (left) and the original training set (right) at 300 K.

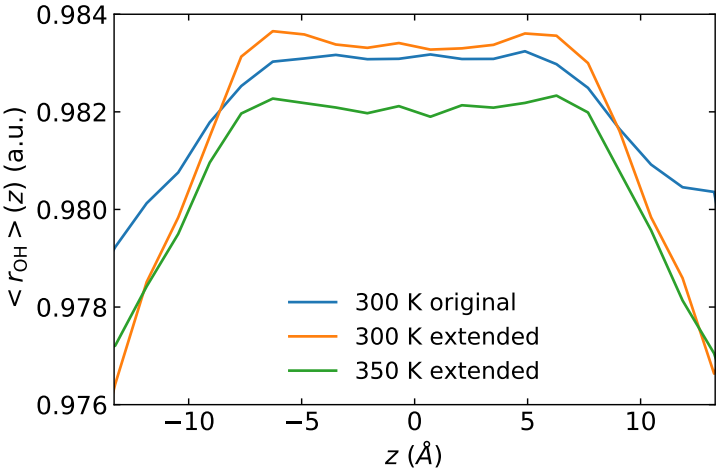


Figure 10.5.: Mean OH-bond length as a function of z, for NNPs trained on the original and extended training set. The origin is set to the center of mass for each configuration.

olation warnings per timestep and per atom in the case of the extended training set. The poor extrapolation properties of NNs thus imply that the exact history of the training of a NNP, based only on bulk configurations, determines whether or not the NNP terribly fails for water-vapor interfaces.

In the following only results from the NNP trained on the extended data set are presented. In Fig. 10.6 the density profiles are shown for all simulated temperatures. Again, we employed the method described in Section 7.3.1 to obtain values for the liquid and vapor densities. The exact values and their standard deviation can be found in Table 10.1. Based on these densities we tried to estimate the critical

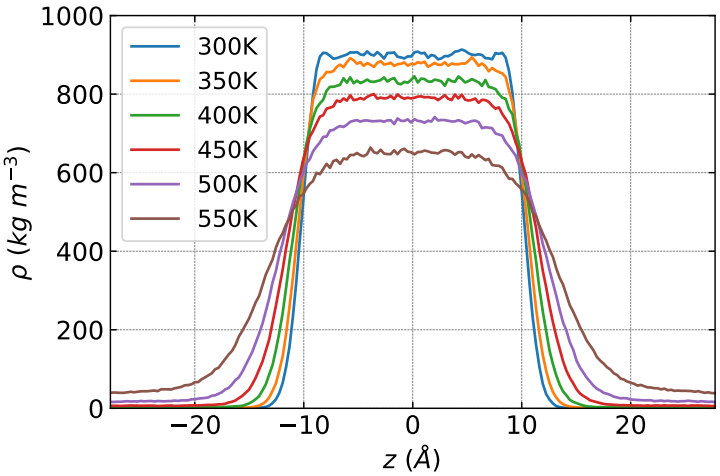


Figure 10.6.: Mass density profile for NNP based on the extended training set for different temperatures.

Temp. (K)	ρ_L (kg/m^3)	ρ_V (kg/m^3)
300	903(3)	0.13(3)
350	877(2)	0.60(4)
400	833(2)	2.09(6)
450	792(1)	6.39(9)
500	735(1)	17.4(2)
550	657(1)	39.9(3)

Table 10.1.: Liquid and vapor densities, ρ_L and ρ_V, for the NNP simulations based on the RPBE functional.

temperature T_c of DFT water based on the RPBE functional with vdW correction. Therefore, we chose the fitting function [85]

$$\rho_\pm = \rho_\mathrm{c} + a|T - T_\mathrm{c}| \pm b|T - T_\mathrm{c}|^\beta, \tag{10.1}$$

where $\rho_+ = \rho_\mathrm{L}$, $\rho_- = \rho_\mathrm{V}$. Besides the critical temperature T_c, there are also the critical density ρ_c, coefficients a, b and exponent β as fitting parameters. We tried to find values for these parameters by minimizing the following weighted error function

$$\Gamma = \sum_{T_i} \left(\frac{1}{\sigma_{\rho_\mathrm{V}(T_i)}} \left[\rho_+(T_i) - \rho_\mathrm{V}(T_i)\right] \right)^2 + \left(\frac{1}{\sigma_{\rho_\mathrm{L}(T_i)}} \left[\rho_-(T_i) - \rho_\mathrm{L}(T_i)\right] \right)^2 \tag{10.2}$$

for $T_i \in \{300\mathrm{K}, 350\mathrm{K}, 400\mathrm{K}, 450\mathrm{K}, 500\mathrm{K}, 550\mathrm{K}\}$. On the one hand, $\rho_+(T_i)$ and $\rho_-(T_i)$ are the values of the fitting function for temperature T_i. On the other hand, $\rho_\mathrm{L}(T_i)$ and $\rho_\mathrm{V}(T_i)$ are our results from fitting the mass density curves at the respective temperatures. The error function also considers the corresponding values of their standard deviation σ_{ρ_L} and σ_{ρ_V}. The minimization was done by utilizing the function **optimize.fmin**, which uses a downhill simplex algorithm and is part of the open source Python library Scipy [71, 72]. The resulting fit is shown in Fig. 10.7 and yields a critical temperature of T_c=657.3 K. The other parameters are ρ_c=307.1 kg/m^3, a=0.4280 kg/m^{3}K, b=74.28 kg/m^{3}K and β=0.3101. In Table 10.2 the evaluated

Temp. (K)	γ (mN/m)
300	64.6(8)
350	55.6(5)
400	46.0(5)
450	36.0(4)
500	24.9(4)
550	14.8(4)

Table 10.2.: Surface tension γ for the LJ potential and the NNP for different temperatures.

values of the surface tension are listed for all available temperatures. Again, we

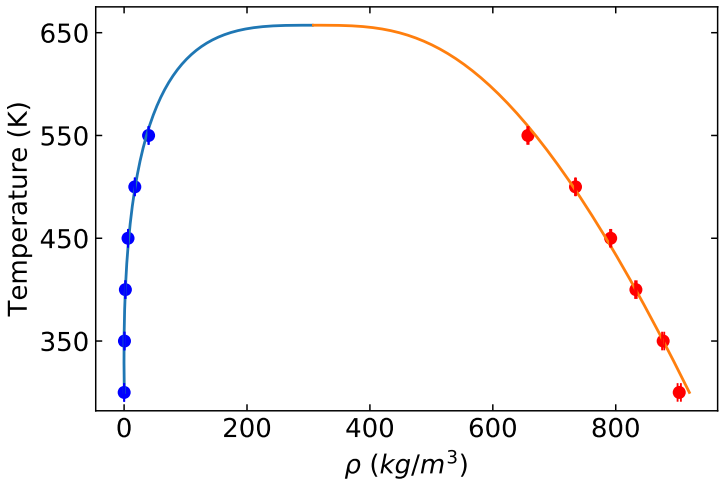


Figure 10.7.: Vapor (blue points) and liquid (orange points) densities as well as the two branches of the fitting function of Eq. 10.1 with the parameters obtained by minimization of the error function (see Eq. 10.2).

tried to estimate the critical temperature, this time based on the surface tension values. Therefore, we employed the following fitting function, as was done by Sega and Dellago [74]:

$$\gamma = B\tau^{\mu}\left(1 + b\tau\right), \tag{10.3}$$

where $\tau = 1 - T/T_c$. For this fit, we use T_c, B, b and μ as free parameters. The experimental curve of the surface tension of water can be represented by the parameters $T_c = 647.15$, $B = 235.8\text{mN/m}$, $b = -0.625$ and $\mu = 1.256$ [86]. In contrast, our fit by means of the Scipy function **optimize.curve_fit** yielded $T_c = 640(18)$, $B = 222(60)\text{mN/m}$, $b = -0.61(20)$ and $\mu = 1.33(23)$. The resulting curve we obtained from the fit as well as the experimental curve are depicted in Fig. 10.8.

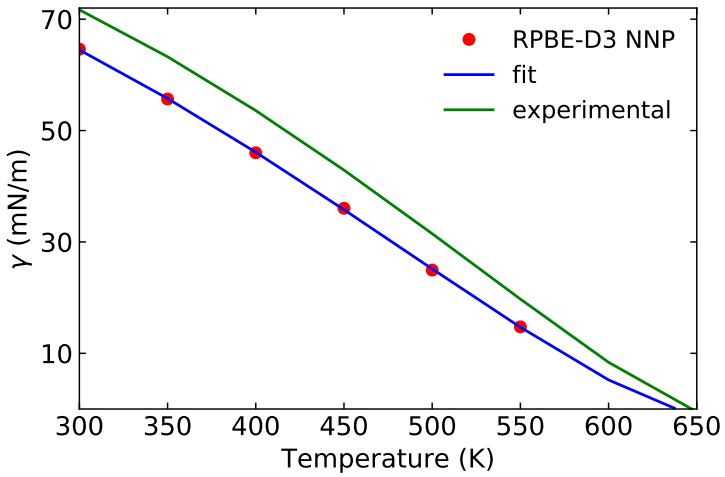
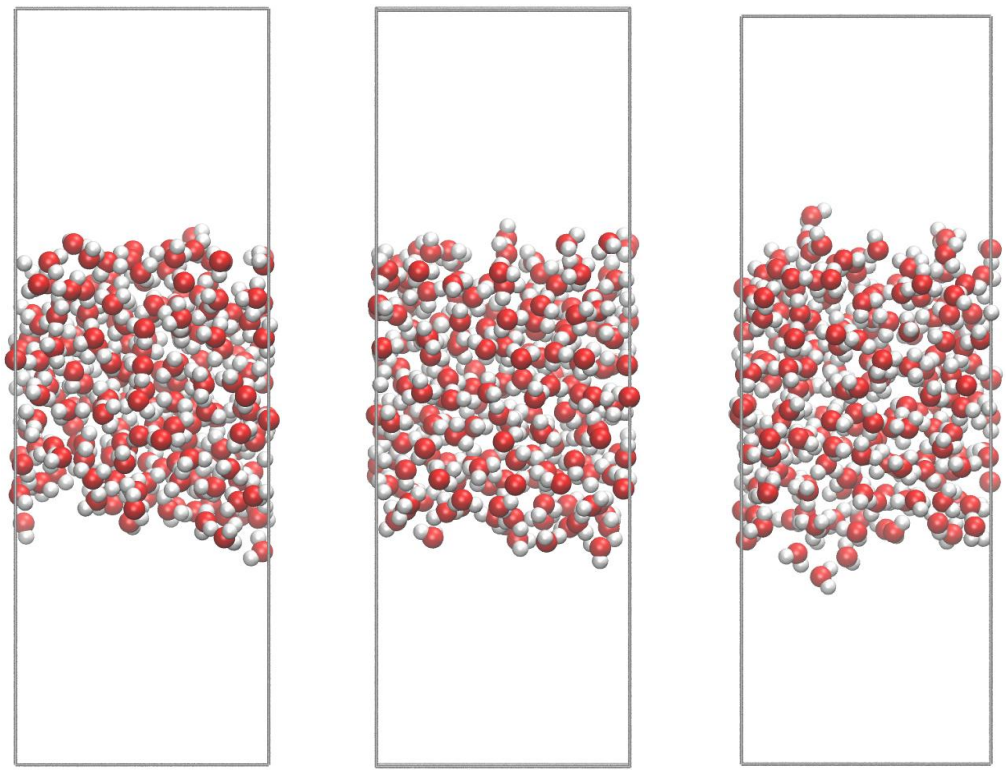


Figure 10.8.: Surface tension γ against the temperature for the NNP simulations and the resulting fit of the function in Eq. 10.3 as well as the experimental curve.

Thus, the estimated values of the critical temperature obtained by these two fits differ from each other by approximately 17 mN/m, which is still within the range of the estimated standard errors. At the end of this chapter we present three snapshots of the NNP simulations at 300 K. In Fig. 10.9 one snapshot is shown for the NNP based on the SPC/F model and two snapshots are illustrating the systems in the cases of DFT-based NNPs trained on the extended and on the original data set.

10.4. Discussion and open questions

We trained a NNP to predict energies and forces of atoms of the water liquid-vapor interface based on DFT calculations employing the RPBE functional. Even though we can easily examine the errors of the training set and also those of single configurations, we can not easily estimate ensemble averages of macroscopic observables obtained by NNP simulations. Therefore, for the case of the empirical SPC/F model, we demonstrated that NNP based simulations can indeed reproduce essential properties of the liquid-vapor interface. Only for simulations at higher temperatures, $T > 450$ K, we found significant deviations regarding the densities of the liquid and vapor phases between the NNP and the reference simulations. Of course, the



(a) SPC/F (b) DFT (extended) (c) DFT (original)

Figure 10.9.: Snapshots of the NNP simulations at 300 K, based on the SPC/F model and DFT with the extended and original data set used for the training.

SPC/F model shall not be carelessly compared with DFT-based calculations, but in both cases NNP potentially suffer from similar issues, for instance the needed cutoff radius and the very anisotropic arrangement of atoms near the interface. For both SPC/F and DFT water we observed a clear difference regarding the prediction error for the z component of forces, on the one hand, and the x and y component, on the other. However, this is less marked for DFT water. Surprisingly, we also found that it is possible to perform simulations based on NNPs which were only fit to data from bulk configurations. Nevertheless, important properties such as the OH bond length and the statistical orientation of molecules as a function of the distance to the interface exhibit a substantially different behavior compared with NNP simulations based on training sets including interface configurations. Furthermore, a high number of extrapolation warnings were issued in the former case. It is thus clear that interfaces need to be taken into account during the training.

For $T = 300$ K, our results agree well with previous work regarding the density of the liquid phase ρ_L. Morawietz *et al.* have obtained $\rho_\mathrm{L} = 895$ kg/m by NNP NpT simulations at 300 K and 1 bar [15]. Schienbein and Marx have performed ab initio Gibbs ensemble Monte Carlo simulations for the liquid-vapor phase diagram of RPBE-D3 water [87]. At 300 K, they have found $\rho_\mathrm{L} = 912$ kg/m^3, which compares well with our result, $\rho_\mathrm{L} = 903$ kg/m^3. For higher temperatures our values of ρ_L are also below the results of Schienbein and Marx but the difference is bigger (around 40 kg/m^3 at 550 K). They have also evaluated the critical temperature, $T_c = 710$ K, which is way above our estimations (640 K and 657 K). However, they have only used 128 molecules and the statistical uncertainties of the obtained densities have been estimated as 20 kg/m^3.

Nagata *et al.* performed AIMD simulations for the calculation of the surface tension of the water liquid-vapor interface at 300 K using DFT and various different functionals [88]. One of them also was the RPBE functional together with vdW correction (D3 method). For RPBE-D3 water they obtained a value of $\gamma = (83 \pm 28)$ mN/m, whereas we found $\gamma = (64.6 \pm 0.8)$ mN/m. The simulation of Nagata *et al.* was done by using only 80 molecules and a simulation time of 264 ps.

The tool of NNPs is, of course, just an approximation technique and naturally implies approximation errors. When we consider again the case of our NNP for the SPC/F water-vapor interface, we can clearly see the deviations between the densities of the liquid and vapor phases obtained by the reference model and the NNP. It is yet unclear if these deviations originate from general approximation errors or from the limitations of NNPs regarding long range interactions due to the cutoff in particular. Future work could investigate the effect of explicit consideration of long range interactions, for instance, by means of Hirshfeld charges [89], as was suggested by Behler [6]. Another interesting idea which is worth investigating is the application of empirical potentials to possibly improve the treatment of electrostatic interactions. The energy of each configuration of a training set computed by ab ini-

tio methods could be subtracted by the electrostatic part of the empirical potential. This electrostatic part could then be added to predictions of the NNP.

A. Reformulation of the extended Kalman filter with forgetting factor

Here we show that the EKF with forgetting factor of Section 2.4.5 for optimizing the weights of a NN,

$$\lambda_k = \mu\lambda_{k-1} + 1 - \mu, \tag{A.1}$$

$$\boldsymbol{G}_k = \lambda_k^{-1}\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}}\left(\lambda_k^{-1}\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}} + \boldsymbol{R}_k\right)^{-1}, \tag{A.2}$$

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{G}_k\boldsymbol{\xi}_k, \tag{A.3}$$

$$\boldsymbol{P}_{k+1} = \lambda_k^{-1}(\boldsymbol{I} - \boldsymbol{G}_k\boldsymbol{H}_k)\boldsymbol{P}_k + \boldsymbol{Q}_k, \tag{A.4}$$

can also be formulated as follows:

$$\lambda_k = \mu\lambda_{k-1} + 1 - \mu, \tag{A.5}$$

$$\alpha_k = \alpha_{k-1}\lambda_k, \tag{A.6}$$

$$\boldsymbol{G}_k^* = \boldsymbol{P}_k^*\boldsymbol{H}_k^{\mathrm{T}}\left(\boldsymbol{H}_k\boldsymbol{P}_k^*\boldsymbol{H}_k^{\mathrm{T}} + \alpha_k\boldsymbol{R}_k\right)^{-1}, \tag{A.7}$$

$$\hat{\boldsymbol{w}}_{k+1}^* = \hat{\boldsymbol{w}}_k^* + \boldsymbol{G}_k^*\boldsymbol{\xi}_k, \tag{A.8}$$

$$\boldsymbol{P}_{k+1}^* = (\boldsymbol{I} - \boldsymbol{G}_k^*\boldsymbol{H}_k)\boldsymbol{P}_k^* + \alpha_k\boldsymbol{Q}_k, \tag{A.9}$$

where α_0=1. We will show that $\boldsymbol{G}_k^*$=$\boldsymbol{G}_k$ and, therefore, $\hat{\boldsymbol{w}}_k^*$=$\hat{\boldsymbol{w}}_k$ for all $k \geq 0$. That means, the resulting weights remain unchanged by this reformulation of the optimization scheme.

Proof. First, we note that Eq. A.2 can be written as

$$\boldsymbol{G}_k = \boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}}\left(\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^{\mathrm{T}} + \lambda_k\boldsymbol{R}_k\right)^{-1}. \tag{A.10}$$

We shall assume that $\boldsymbol{P}_0$ and $\boldsymbol{P}_0^*$ are identical. For $\boldsymbol{G}_0^*$ we find

$$\boldsymbol{G}_0^* = \boldsymbol{P}_0^*\boldsymbol{H}_0^{\mathrm{T}}\left(\boldsymbol{H}_0\boldsymbol{P}_0^*\boldsymbol{H}_0^{\mathrm{T}} + \alpha_0\boldsymbol{R}_0\right)^{-1}. \tag{A.11}$$

Since α_0=λ_0 and $\boldsymbol{P}_0^*$=$\boldsymbol{P}_0$, we obtain $\boldsymbol{G}_0^*$=$\boldsymbol{G}_0$. For $\boldsymbol{P}_1^*$ we get

$$\boldsymbol{P}_1^* = (\boldsymbol{I} - \boldsymbol{G}_0^*\boldsymbol{H}_0)\boldsymbol{P}_0^* + \alpha_0\boldsymbol{Q}_0 \tag{A.12}$$

and thus $\boldsymbol{P}_1^* = \alpha_0 \boldsymbol{P}_1$. From the above equations we can conclude that for $k = 0$ the following equations are valid:

$$\boldsymbol{G}_k^* = \boldsymbol{G}_k, \tag{A.13}$$

$$\boldsymbol{P}_{k+1}^* = \alpha_k \boldsymbol{P}_{k+1}. \tag{A.14}$$

Then, for l=k+1, we can express $\boldsymbol{G}_l^*$ as

$$\boldsymbol{G}_l^* = \boldsymbol{P}_l^* \boldsymbol{H}_l^{\mathrm{T}} \left(\boldsymbol{H}_l \boldsymbol{P}_l^* \boldsymbol{H}_l^{\mathrm{T}} + \alpha_l \boldsymbol{R}_l \right)^{-1} \tag{A.15}$$

and by using Eq. A.14 we obtain

$$\boldsymbol{G}_l^* = \boldsymbol{P}_l \boldsymbol{H}_l^{\mathrm{T}} \left(\boldsymbol{H}_l \boldsymbol{P}_l \boldsymbol{H}_l^{\mathrm{T}} + \frac{\alpha_l}{\alpha_{l-1}} \boldsymbol{R}_l \right)^{-1}. \tag{A.16}$$

By using Eq. A.6 the expression α_l / α_{l-1} can be rewritten as

$$\frac{\alpha_l}{\alpha_{l-1}} = \frac{\lambda_l \alpha_{l-1}}{\alpha_{l-1}} = \lambda_l \tag{A.17}$$

and therefore we arrive at $\boldsymbol{G}_l^* = \boldsymbol{G}_l$. Together with Eq. A.9 this relation gives

$$\boldsymbol{P}_{l+1}^* = (\boldsymbol{I} - \boldsymbol{G}_l \boldsymbol{H}_l) \boldsymbol{P}_l^* + \alpha_l \boldsymbol{Q}_l. \tag{A.18}$$

By utilizing Eq. A.14 we get

$$\boldsymbol{P}_{l+1}^* = (\boldsymbol{I} - \boldsymbol{G}_l \boldsymbol{H}_l) \alpha_{l-1} \boldsymbol{P}_l + \alpha_l \boldsymbol{Q}_l \tag{A.19}$$

and, finally, employing the relation of Eq. *A*.17 gives

$$\boldsymbol{P}_{l+1}^* = \alpha_l \left[\lambda_l^{-1} (\boldsymbol{I} - \boldsymbol{G}_l \boldsymbol{H}_l) \boldsymbol{P}_l + \boldsymbol{Q}_l \right] = \alpha_l \boldsymbol{P}_{l+1}. \tag{A.20}$$

□

B. Weight derivatives of forces

Here we prove that the expression

$$\gamma^l_{\alpha\beta} = \frac{\partial}{\partial x_j}\frac{\partial \varphi^L_1}{\partial w^l_{\alpha\beta}}, \tag{B.1}$$

given in Section 4.2.3, where

$$\varphi^L_1 = f^{L+1}_1\left(w^L_{01} + \sum_{r=1}^{m^{(L)}} w^L_{r1} y^L_r\right), \tag{B.2}$$

can be written as

$$\gamma^l_{\alpha\beta} = \sum_{q=1}^{m^{(l+2)}} y^l_\alpha w^{l+1}_{\beta q}\left[\kappa^{l+1}_q \frac{\partial f^{l+1}_\beta}{\partial u^{l+1}_\beta} + h^{l+1}_q \frac{\partial^2 f^{l+1}_\beta}{\partial\left(u^{l+1}_\beta\right)^2}\frac{\partial u^{l+1}_\beta}{\partial x_j}\right] + h^l_\beta \frac{\partial f^l_\alpha}{\partial u^l_\alpha}\frac{\partial u^l_\alpha}{\partial x_j} \tag{B.3}$$

for $0 \leq l \leq L-1$.

Proof:

To prove the statement above, we start with the following expression,

$$\gamma^l_{\alpha\beta} = \frac{\partial}{\partial x_j}\frac{\partial \varphi^L_1}{\partial w^l_{\alpha\beta}} \tag{B.4}$$

and utilize the developed expressions for $g^l_{\alpha\beta}$ and h^l_j from Eq. 4.36, which we have proven to be valid for $0 \leq l \leq L$:

$$\begin{aligned}\gamma^{l-1}_{\alpha\beta} &= \frac{\partial}{\partial x_j} g^{l-1}_{\alpha\beta} = \frac{\partial}{\partial x_j}\sum_{q=1}^{m^{(l+1)}} h^l_q w^l_{\beta q}\frac{\partial f^l_\beta}{\partial u^l_\beta} y^{l-1}_\alpha \\ &= \sum_{q=1}^{m^{(l+1)}}\left[\frac{\partial h^l_q}{\partial x_j} w^l_{\beta q}\frac{\mathrm{d}f^l_\beta}{\mathrm{d}u^l_\beta} y^{l-1}_\alpha + h^l_q \frac{\partial}{\partial x_j}\left(w^l_{\beta q}\frac{\partial f^l_\beta}{\partial u^l_\beta} y^{l-1}_\alpha\right)\right].\end{aligned} \tag{B.5}$$

Since

$$\gamma^l_{rq} = \frac{\partial g^l_{rq}}{\partial x_j}, \tag{B.6}$$

we also have

$$\kappa^l_q = \frac{\partial h^l_q}{\partial x_j}. \tag{B.7}$$

Using Eq. B.7 and taking the partial derivative in the second summand within the square brackets in Eq. B.5 gives us

$$\gamma_{\alpha\beta}^{l-1} = \sum_{q=1}^{m^{(l+1)}} \left[\kappa_q^l w_{\beta q}^l \frac{\mathrm{d}f_\beta^l}{\mathrm{d}u_\beta^l} y_\alpha^{l-1} + h_q^l w_{\beta q}^l \frac{\partial^2 f_\beta^l}{\partial \left(u_\beta^l\right)^2} \frac{\partial u_\beta^l}{\partial x_j} y_\alpha^{l-1} \right. \\ \left. + h_q^l w_{\beta q}^l \frac{\partial f_\beta^l}{\partial u_\beta^l} \frac{\partial f_\alpha^{l-1}}{\partial u_\alpha^{l-1}} \frac{\partial u_\alpha^{l-1}}{\partial x_j} \right] \tag{B.8}$$

and by utilizing the definition of h_β^{l-1},

$$h_\beta^{l-1} = \sum_{q=1}^{m^{(l+1)}} \left[h_q^l w_{\beta q}^l \frac{\partial f_\beta^l}{\partial u_\beta^l} \right], \tag{B.9}$$

we can rewrite Eq. B.8 to

$$\gamma_{\alpha\beta}^{l-1} = \sum_{q=1}^{m^{(l+1)}} y_\alpha^{l-1} w_{\beta q}^l \left[\kappa_q^l \frac{\mathrm{d}f_\beta^l}{\mathrm{d}u_\beta^l} + h_q^l \frac{\partial^2 f_\beta^l}{\partial \left(u_\beta^l\right)^2} \frac{\partial u_\beta^l}{\partial x_j} \right] + h_\beta^{l-1} \frac{\partial f_\alpha^{l-1}}{\partial u_\alpha^{l-1}} \frac{\partial u_\alpha^{l-1}}{\partial x_j}. \tag{B.10}$$

□

C. Distribution of radial displacements

Here we prove that a random variable $R = aX^{1/3}$ is distributed as

$$f_R(r) = \frac{3r^2\lambda}{a^3} \exp\left[-\lambda\left(\frac{r}{a}\right)^3\right], \tag{C.1}$$

where X has the exponential distribution $Exp_X(x) = \lambda e^{-x\lambda}$.

Proof. We start with the cumulative distribution function $P(R \leq q)$ with $q = ax^{1/3}$,

$$P(R \leq ax^{1/3}) = \int_0^x \lambda e^{-\lambda x'} dx'. \tag{C.2}$$

Substituting $ax^{1/3}$ by r gives

$$P(R \leq r) = \int_0^{(r/a)^3} \lambda e^{-\lambda x'} dx' = -e^{-\lambda r^3/a^3} \tag{C.3}$$

and differentiating with respect to r gives us the desired probability density function

$$f_R(r) = \frac{3r^2\lambda}{a^3} \exp\left[-\lambda\left(\frac{r}{a}\right)^3\right]. \tag{C.4}$$

□

D. Sets of symmetry functions

Type	EC	E1	E2	μ	R_s	λ	ζ	R_c
rad	H	H	-	0.001	0.0	-	-	12.00
rad	H	H	-	0.01	0.0	-	-	12.00
rad	H	H	-	0.03	0.0	-	-	12.00
rad	H	H	-	0.06	0.0	-	-	12.00
rad	H	H	-	0.15	1.9	-	-	12.00
rad	H	H	-	0.30	1.9	-	-	12.00
rad	H	H	-	0.60	1.9	-	-	12.00
rad	H	H	-	1.50	1.9	-	-	12.00
rad	H	O	-	0.001	0.0	-	-	12.00
rad	H	O	-	0.01	0.0	-	-	12.00
rad	H	O	-	0.03	0.0	-	-	12.00
rad	H	O	-	0.06	0.0	-	-	12.00
rad	H	O	-	0.15	0.9	-	-	12.00
rad	H	O	-	0.30	0.9	-	-	12.00
rad	H	O	-	0.60	0.9	-	-	12.00
rad	H	O	-	1.50	0.9	-	-	12.00
ang,2	H	H	O	0.01	-	1.0	4.0	12.00
ang,2	H	H	O	0.01	-	-1.0	4.0	12.00
ang,2	H	H	O	0.03	-	1.0	1.0	12.00
ang,2	H	H	O	0.03	-	-1.0	1.0	12.00
ang,2	H	H	O	0.07	-	1.0	1.0	12.00
ang,2	H	H	O	0.07	-	-1.0	1.0	12.00
ang,2	H	H	O	0.20	-	-1.0	1.0	12.00
ang,2	H	O	O	0.001	-	1.0	4.0	12.00
ang,2	H	O	O	0.001	-	-1.0	4.0	12.00
ang,2	H	O	O	0.03	-	1.0	1.0	12.00
ang,2	H	O	O	0.03	-	-1.0	1.0	12.00

Table D.1.: Symmetry function set 1 for hydrogen, R_s and R_c in Bohr, η in Bohr^{-2}

Type	EC	E1	E2	μ	R_s	λ	ζ	R_c
rad	O	H	-	0.001	0.0	-	-	12.00
rad	O	H	-	0.01	0.0	-	-	12.00
rad	O	H	-	0.03	0.0	-	-	12.00
rad	O	H	-	0.06	0.0	-	-	12.00
rad	O	H	-	0.15	0.9	-	-	12.00
rad	O	H	-	0.30	0.9	-	-	12.00
rad	O	H	-	0.60	0.9	-	-	12.00
rad	O	H	-	1.50	0.9	-	-	12.00
rad	O	O	-	0.001	0.0	-	-	12.00
rad	O	O	-	0.01	0.0	-	-	12.00
rad	O	O	-	0.03	0.0	-	-	12.00
rad	O	O	-	0.06	0.0	-	-	12.00
rad	O	O	-	0.15	4.0	-	-	12.00
rad	O	O	-	0.30	4.0	-	-	12.00
rad	O	O	-	0.60	4.0	-	-	12.00
rad	O	O	-	1.50	4.0	-	-	12.00
ang,2	O	H	H	0.01	-	1.0	4.0	12.00
ang,2	O	H	H	0.01	-	-1.0	4.0	12.00
ang,2	O	H	H	0.03	-	1.0	1.0	12.00
ang,2	O	H	H	0.03	-	-1.0	1.0	12.00
ang,2	O	H	H	0.07	-	1.0	1.0	12.00
ang,2	O	H	H	0.07	-	-1.0	1.0	12.00
ang,2	O	H	O	0.001	-	1.0	4.0	12.00
ang,2	O	H	O	0.001	-	-1.0	4.0	12.00
ang,2	O	H	O	0.03	-	1.0	1.0	12.00
ang,2	O	H	O	0.03	-	-1.0	1.0	12.00
ang,2	O	O	O	0.001	-	1.0	4.0	12.00
ang,2	O	O	O	0.001	-	-1.0	4.0	12.00
ang,2	O	O	O	0.03	-	1.0	1.0	12.00
ang,2	O	O	O	0.03	-	-1.0	1.0	12.00

Table D.2.: Symmetry function set 1 for oxygen, R_s and R_c in Bohr, η in Bohr^{-2}

Type	EC	E1	E2	μ	R_s	R_c
rad	H	H	-	0.001	0.0	12.00
rad	H	H	-	0.01	0.0	12.00
rad	H	H	-	0.03	0.0	12.00
rad	H	H	-	0.06	0.0	12.00
rad	H	H	-	0.15	1.9	12.00
rad	H	H	-	0.30	1.9	12.00
rad	H	H	-	0.60	1.9	12.00
rad	H	H	-	1.50	1.9	12.00
rad	H	O	-	0.001	0.0	12.00
rad	H	O	-	0.01	0.0	12.00
rad	H	O	-	0.03	0.0	12.00
rad	H	O	-	0.06	0.0	12.00
rad	H	O	-	0.15	0.9	12.00
rad	H	O	-	0.30	0.9	12.00
rad	H	O	-	0.60	0.9	12.00
rad	H	O	-	1.50	0.9	12.00
mr	H	H	O	0.01	0.0	12.00
mr	H	H	O	0.03	0.0	12.00
mr	H	H	O	0.06	0.0	12.00
mr	H	H	O	0.2	0.0	12.00
mr	H	H	O	0.5	0.0	12.00
mr	H	O	O	0.005	0.0	12.00
mr	H	O	O	0.01	0.0	12.00
mr	H	O	O	0.03	0.0	12.00
mr	H	O	O	0.06	0.0	12.00
mr	H	O	O	0.2	0.0	12.00

Table D.3.: Symmetry function set 2 for hydrogen, R_s and R_c in Bohr, η in Bohr^{-2}

Type	EC	E1	E2	μ	R_s	R_c
rad	O	H	-	0.001	0.0	12.00
rad	O	H	-	0.01	0.0	12.00
rad	O	H	-	0.03	0.0	12.00
rad	O	H	-	0.06	0.0	12.00
rad	O	H	-	0.15	0.9	12.00
rad	O	H	-	0.30	0.9	12.00
rad	O	H	-	0.60	0.9	12.00
rad	O	H	-	1.50	0.9	12.00
rad	O	O	-	0.001	0.0	12.00
rad	O	O	-	0.01	0.0	12.00
rad	O	O	-	0.03	0.0	12.00
rad	O	O	-	0.06	0.0	12.00
rad	O	O	-	0.15	4.0	12.00
rad	O	O	-	0.30	4.0	12.00
rad	O	O	-	0.60	4.0	12.00
rad	O	O	-	1.50	4.0	12.00
mr	O	H	H	0.01	0.0	12.00
mr	O	H	H	0.03	0.0	12.00
mr	O	H	H	0.06	0.0	12.00
mr	O	H	H	0.2	0.0	12.00
mr	O	H	H	0.5	0.0	12.00
mr	O	H	O	0.005	0.0	12.00
mr	O	H	O	0.01	0.0	12.00
mr	O	H	O	0.03	0.0	12.00
mr	O	H	O	0.06	0.0	12.00
mr	O	H	O	0.2	0.0	12.00
mr	O	O	O	0.001	0.0	12.00
mr	O	O	O	0.005	0.0	12.00
mr	O	O	O	0.01	0.0	12.00
mr	O	O	O	0.03	0.0	12.00
mr	O	O	O	0.06	0.0	12.00

Table D.4.: Symmetry function set 2 for oxygen, R_s and R_c in Bohr, η in Bohr^{-2}

Type	EC	E1	E2	μ	R_s	λ	ζ	m	R_c
leg	H	H	-	-	-	-	-	0	12.00
leg	H	O	-	-	-	-	-	0	12.00
rad	H	H	-	0.001	0.0	-	-	-	12.00
rad	H	H	-	0.01	0.0	-	-	-	12.00
rad	H	H	-	0.03	0.0	-	-	-	12.00
rad	H	H	-	0.06	0.0	-	-	-	12.00
rad	H	H	-	0.15	1.9	-	-	-	12.00
rad	H	H	-	0.30	1.9	-	-	-	12.00
rad	H	H	-	0.60	1.9	-	-	-	12.00
rad	H	H	-	1.50	1.9	-	-	-	12.00
rad	H	O	-	0.001	0.0	-	-	-	12.00
rad	H	O	-	0.01	0.0	-	-	-	12.00
rad	H	O	-	0.03	0.0	-	-	-	12.00
rad	H	O	-	0.06	0.0	-	-	-	12.00
rad	H	O	-	0.15	0.9	-	-	-	12.00
rad	H	O	-	0.30	0.9	-	-	-	12.00
rad	H	O	-	0.60	0.9	-	-	-	12.00
rad	H	O	-	1.50	0.9	-	-	-	12.00
ang,2	H	H	O	0.01	-	1.0	4.0	-	12.00
ang,2	H	H	O	0.01	-	-1.0	4.0	-	12.00
ang,2	H	H	O	0.03	-	1.0	1.0	-	12.00
ang,2	H	H	O	0.03	-	-1.0	1.0	-	12.00
ang,2	H	H	O	0.07	-	1.0	1.0	-	12.00
ang,2	H	H	O	0.07	-	-1.0	1.0	-	12.00
ang,2	H	H	O	0.20	-	-1.0	1.0	-	12.00
ang,2	H	O	O	0.001	-	1.0	4.0	-	12.00
ang,2	H	O	O	0.001	-	-1.0	4.0	-	12.00
ang,2	H	O	O	0.03	-	1.0	1.0	-	12.00
ang,2	H	O	O	0.03	-	-1.0	1.0	-	12.00
mr	H	H	O	0.03	0.0	-	-	-	9.50
mr	H	H	O	0.06	0.0	-	-	-	9.50
mr	H	O	O	0.03	0.0	-	-	-	9.50
mr	H	O	O	0.06	0.0	-	-	-	9.50

Table D.5.: Symmetry function set 3 for hydrogen, R_s and R_c in Bohr, η in Bohr^{-2}

Type	EC	E1	E2	μ	R_s	λ	ζ	m	R_c
leg	O	H	-	-	-	-	-	0	12.00
leg	O	O	-	-	-	-	-	0	12.00
rad	O	H	-	0.001	0.0	-	-	-	12.00
rad	O	H	-	0.01	0.0	-	-	-	12.00
rad	O	H	-	0.03	0.0	-	-	-	12.00
rad	O	H	-	0.06	0.0	-	-	-	12.00
rad	O	H	-	0.15	0.9	-	-	-	12.00
rad	O	H	-	0.30	0.9	-	-	-	12.00
rad	O	H	-	0.60	0.9	-	-	-	12.00
rad	O	H	-	1.50	0.9	-	-	-	12.00
rad	O	O	-	0.001	0.0	-	-	-	12.00
rad	O	O	-	0.01	0.0	-	-	-	12.00
rad	O	O	-	0.03	0.0	-	-	-	12.00
rad	O	O	-	0.06	0.0	-	-	-	12.00
rad	O	O	-	0.15	4.0	-	-	-	12.00
rad	O	O	-	0.30	4.0	-	-	-	12.00
rad	O	O	-	0.60	4.0	-	-	-	12.00
rad	O	O	-	1.50	4.0	-	-	-	12.00
ang,2	O	H	H	0.01	-	1.0	4.0	-	12.00
ang,2	O	H	H	0.01	-	-1.0	4.0	-	12.00
ang,2	O	H	H	0.03	-	1.0	1.0	-	12.00
ang,2	O	H	H	0.03	-	-1.0	1.0	-	12.00
ang,2	O	H	H	0.07	-	1.0	1.0	-	12.00
ang,2	O	H	H	0.07	-	-1.0	1.0	-	12.00
ang,2	O	H	O	0.001	-	1.0	4.0	-	12.00
ang,2	O	H	O	0.001	-	-1.0	4.0	-	12.00
ang,2	O	H	O	0.03	-	1.0	1.0	-	12.00
ang,2	O	H	O	0.03	-	-1.0	1.0	-	12.00
ang,2	O	O	O	0.001	-	1.0	4.0	-	12.00
ang,2	O	O	O	0.001	-	-1.0	4.0	-	12.00
ang,2	O	O	O	0.03	-	1.0	1.0	-	12.00
ang,2	O	O	O	0.03	-	-1.0	1.0	-	12.00
mr	O	H	H	0.03	0.0	-	-	-	9.50
mr	O	H	H	0.06	0.0	-	-	-	9.50
mr	O	H	O	0.03	0.0	-	-	-	9.50
mr	O	H	O	0.06	0.0	-	-	-	9.50
mr	O	O	O	0.03	0.0	-	-	-	9.50
mr	O	O	O	0.06	0.0	-	-	-	9.50

Table D.6.: Symmetry function set 3 for oxygen, R_s and R_c in Bohr, η in Bohr^{-2}

E. Additional results of the SPC/F-based neural network potential

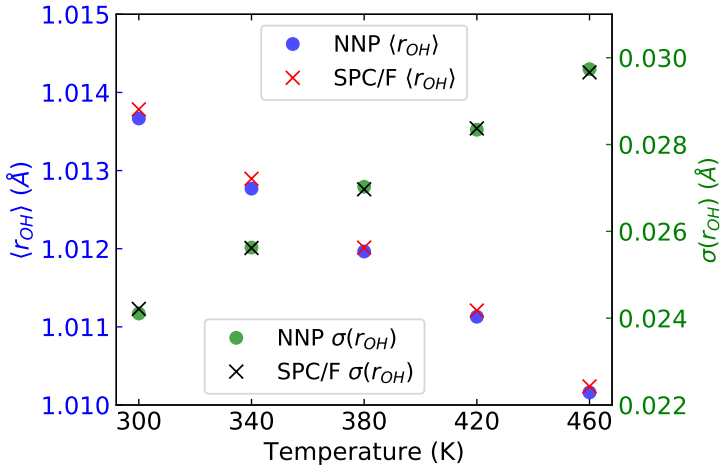


Figure E.1.: Fitted mean and standard deviation of the distributions of the OH-bond lengths for the SPC/F based simulations and the respective NNP simulations.

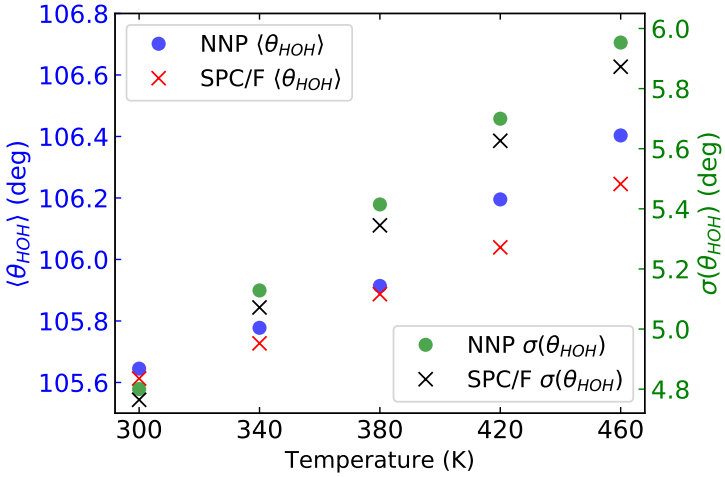


Figure E.2.: Fitted mean and standard deviation of the distributions of the HOH-bond angles for the SPC/F based simulations and the respective NNP simulations.

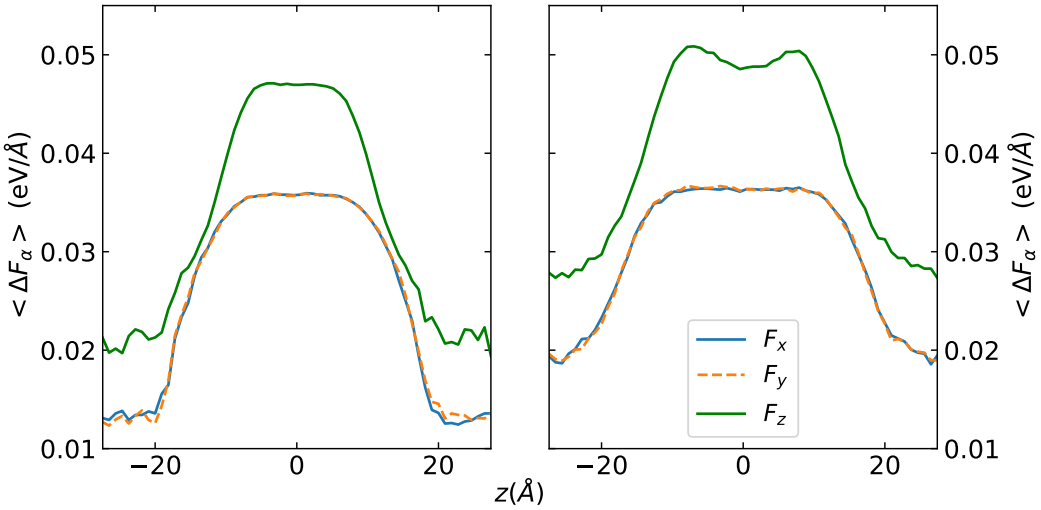


Figure E.3.: Mean error of the x, y and z components of forces as a function of z, during the NNP simulation at 380 K (left) and 460 K (right). The origin is set to the center of mass of the system for each configuration.

Temp. (K)	SPC/F γ (mN/m)	NNP γ (mN/m)
300	46.49(45)	45.24(48)
340	39.62(45)	37.81(38)
380	32.55(45)	30.64(37)
420	25.47(36)	26.09(36)
460	17.70(38)	18.59(36)

Table E.1.: Surface tension, γ, for the SPC/F potential and the NNP for different temperatures.

List of Figures

List of Tables

Bibliography

[1] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., Fei-Fei, L., "Imagenet large scale visual recognition challenge", *Int. J. Comput. Vis.* **2015**, *115*, 211–252.

[2] Lewars, E. G., "The Concept of the potential energy surface" in *Computational chemistry: Introduction to the theory and applications of molecular and quantum mechanics*, Springer International Publishing, **2016**, 9–49.

[3] Allen, M. P., "Introduction to molecular dynamics simulation", *NIC Series* **2004**, *23*, 1–28.

[4] Landau, D. P., Binder, K., *A guide to Monte Carlo simulations in statistical physics*, Cambridge University Press, **2014**.

[5] Parr, R. G., Yang, W., *Density-functional theory of atoms and molecules*, Oxford University Press, **1989**.

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

[7] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., "Gradient-based learning applied to document recognition", *Proc. IEEE* **1998**, *86*, 2278–2324.

[8] Krizhevsky, A., Sutskever, I., Hinton, G. E., "Imagenet classification with deep convolutional neural networks" in Adv. Neural. Inf. Process. Syst. **2012**, 1097–1105.

[9] Graves, A., Mohamed, A. R., Hinton, G., "Speech recognition with deep recurrent neural networks" in Proc. IEEE Int. Conf. Acoust. Speech Signal Process. **2013**, 6645–6649.

[10] Cutts, D., Hoftun, J. S., Sornborger, A., Johnson, C. R., Zeller, R. T., "Neural networks for event filtering at D0", *Comput. Phys. Commun.* **1989**, *57*, 478–482.

[11] Nguyen-Thien, T, Tran-Cong, T, "Approximation of functions and their derivatives: a neural network implementation with applications", *Appl. Math. Model.* **1999**, *23*, 687–704.

[12] Krenn, M., Malik, M., Fickler, R., Lapkiewicz, R., Zeilinger, A., "Automated search for new quantum experiments", *Phys. Rev. Lett.* **2016**, *116*, 090405.

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

[14] Behler, J., "First principles neural network potentials for reactive simulations of large molecular and condensed systems", *Angew. Chem. Int. Ed.* **2017**, *56*, 12828–12840.

[15] Morawietz, T., Singraber, A., Dellago, C., Behler, J., "How van der Waals interactions determine the unique properties of water", *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 8368–8373.

[16] Becke, A. D., "Density-functional exchange-energy approximation with correct asymptotic behavior", *Phys. Rev. A* **1988**, *38*, 3098.

[17] Lee, C., Yang, W., Parr, R. G., "Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density", *Phys. Rev. B* **1988**, *37*, 785.

[18] Hammer, B., Hansen, L. B., Nørskov, J. K., "Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals", *Phys. Rev. B* **1999**, *59*, 7413.

[19] Grimme, S., Antony, J., Ehrlich, S., Krieg, H., "A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu", *J. Chem. Phys.* **2010**, *132*, 154104.

[20] Hunter, J. D., "Matplotlib: A 2D graphics environment", *Comput Sci Eng.* **2007**, *9*, 90–95.

[21] Tantau, T., The TikZ and PGF Packages, Manual for version 3.0.0, **2013**.

[22] Kiernan, J., Rajakumar, R., *Barr's "The human nervous system": An anatomical viewpoint*, Lippincott Williams & Wilkins, **2013**.

[23] McCulloch, W. S., Pitts, W., "A logical calculus of the ideas immanent in nervous activity", *Bull. Math. Biol.* **1943**, *5*, 115–133.

[24] Piccinini, G., "The first computational theory of mind and brain: A close look at McCulloch and Pitts's "Logical calculus of ideas immanent in nervous activity"", *Synthese* **2004**, *141*, 175–215.

[25] Hebb, D. O., *The organization of behavior: A neuropsychological theory*, Wiley, **1949**.

[26] Rosenblatt, F., *Principles of neurodynamics*, Spartan Book, **1962**.

[27] Minsky, M., Papert, S. A., *Perceptrons: An introduction to computational geometry*, MIT Press, **1969**.

[28] Werbos, P. J., "Beyond regression: New tools for prediction and analysis in the behavioral sciences", PhD thesis, Harvard University, **1974**

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

[30] Hornik, K., "Approximation capabilities of multilayer feedforward networks", *Neural Networks* **1991**, *4*, 251–257.

[31] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., *Deep learning*, *Vol. 1*, MIT Press, **2016**.

[32] Broomhead, D., Lowe, D., "Radial basis functions, multi-variable functional interpolation and adaptive networks", **1988**, *RSRE-MEMO-4148*.

[33] O'Shea, K., Nash, R., "An introduction to convolutional neural networks", *Clin. Orthop. Relat. Res.* **2015**.

[34] Lipton, Z. C., Berkowitz, J., Elkan, C., "A critical review of recurrent neural networks for sequence learning", *Clin. Orthop. Relat. Res.* **2015**.

[35] Glorot, X., Bordes, A., Bengio, Y., "Deep sparse rectifier neural networks" in AISTATS, **2011**, 315–323.

[36] Larsen, J., Hansen, L. K., Svarer, C., Ohlsson, M, "Design and regularization of neural networks: The optimal use of a validation set" in NNSP, **1996**, 62–71.

[37] Spall, J. C., *Introduction to stochastic search and optimization: estimation, simulation, and control*, *Vol. 65*, John Wiley & Sons, **2005**.

[38] Bottou, L., Curtis, F. E., Nocedal, J., "Optimization methods for large-scale machine learning", *SIAM Rev.*, *60*, 223–311.

[39] Chong, E. K. P., Zak, S. H., *An introduction to optimization*, *Vol. 76*, John Wiley & Sons, **2013**.

[40] Chen, P., "Hessian matrix vs. Gauss-Newton Hessian matrix", *SIAM J. Num. Anal.* **2011**, *49*, 1417–1435.

[41] Yu, H., Wilamowski, B., "Levenberg-Marquardt training" in *The industrial electronics handbook*, *Vol. 5*, CRC Press, **2011**, Chapter 12.

[42] Kalman, R. E., "A new approach to linear filtering and prediction problems", *J. Basic Eng.* **1960**, *82*, 35–45.

[43] Pei, Y., Biswas, S., Fussell, D. S., Pingali, K., "An elementary introduction to Kalman filtering", *arXiv preprint arXiv:1710.04055* **2017**.

[44] Grewal, M. S., Andrews, A. P., Bass, R. W., "Kalman filtering: Theory and practice", *IEEE Trans. Automat. Contr.* **1995**, *40*, 1983.

[45] Maybeck, P. S., *Stochastic models, estimation, and control*, *Vol. 3*, Academic Press, **1982**.

[46] Anderson, B. D. O., Moore, J. B., *Optimal filtering*, *Vol. 21*, Englewood Cliffs, **1979**.

[47] Ribeiro, M. I., "Kalman and extended kalman filters: Concept, derivation and properties", *Institute for Systems and Robotics* **2004**.

[48] Horvath, S., Neuner, H.-B., "Comparison of Levenberg-Marquardt and extended Kalman filter based parameter estimation of artificial neural networks in modelling deformation processes" in JISDM, **2016**.

[49] Puskorius, G., Feldkamp, L., "Multi-stream extended Kalman filter training for static and dynamic neural networks" in Conf. Proc. IEEE Int. Conf. Syst. Man Cybern. *Vol. 3*, IEEE, **1997**, 2006–2011.

[50] Haykin, S., *Kalman filtering and neural networks*, *Vol. 47*, John Wiley & Sons, **2004**.

[51] Blank, T. B., Brown, S. D., "Adaptive, global, extended Kalman filters for training feedforward neural networks", *J. Chemom.* **1994**, *8*, 391–407.

[52] Witkoskie, J. B., Doren, D. J., "Neural network models of potential energy surfaces: Prototypical examples", *J. Chem. Theory Comput.* **2004**, *1*, 14–23.

[53] Handley, C. M., Popelier, P. L. A., "Potential energy surfaces fitted by artificial neural networks", *J. Phys. Chem. A* **2010**, *114*, 3371–3383.

[54] Agrawal, P. M., Raff, L. M., Hagan, M. T., Komanduri, R., "Molecular dynamics investigations of the dissociation of Si O 2 on an ab initio potential energy surface obtained using neural network methods", *J. Chem. Phys.* **2006**, *124*, 134306.

[55] Behler, J., "Constructing high-dimensional neural network potentials: A tutorial review", *Int. J. Quantum Chem.* **2015**, *115*, 1032–1050.

[56] Gassner, H., Probst, M., Lauenstein, A., Hermansson, K., "Representation of intermolecular potential functions by neural networks", *J. Phys. Chem. A* **1998**, *102*, 4596–4605.

[57] Raff, L. M., Malshe, M, Hagan, M, Doughan, D. I., Rockley, M. G., Komanduri, R, "Ab initio potential-energy surfaces for complex, multichannel systems using modified novelty sampling and feedforward neural networks", *J. Chem. Phys.* **2005**, *122*, 084104.

[58] Hobday, S., Smith, R., Belbruno, J., "Applications of neural networks to fitting interatomic potential functions", *Modell. Simul. Mater. Sci. Eng.* **1999**, *7*, 397.

[59] Bholoa, A., Kenny, S. D., Smith, R., "A new approach to potential fitting using neural networks", *Nucl. Instr. Meth. Phys. Res. Sect. B* **2007**, *255*, 1–7.

[60] Behler, J., "Atom-centered symmetry functions for constructing high-dimensional neural network potentials", *J. Chem. Phys.* **2011**, *134*, 074106–13.

[61] Toxvaerd, S., Dyre, J. C., "Communication: Shifted forces in molecular dynamics", *J. Chem. Phys.* **2011**, *134*, 081102.

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

[63] Kaplan, W., *Advanced calculus*, 4th ed., Addison-Wesley, **1993**.

[64] Car, R., "Introduction to density-functional theory and ab-initio molecular dynamics", *Quantum. Struct. Act. Rel.* **2002**, *21*, 97–104.

[65] Hohenberg, P., Kohn, W., "Inhomogeneous electron gas", *Phys. Rev.* **1964**, *136*, B864.

[66] Kohn, W., Sham, L. J., "Quantum density oscillations in an inhomogeneous electron gas", *Phys. Rev.* **1965**, *137*, A1697–A1705.

[67] Berendsen, H. J. C., Postma, J. P. M., Gunsteren, W. F. van, Hermans, J., "Interaction Models for Water in Relation to Protein Hydration" in *Intermolecular forces: Proceedings of the fourteenth Jerusalem symposium on quantum chemistry and biochemistry*, (Ed.: B. Pullman), Springer Netherlands, **1981**, 331–342.

[68] Toukan, K., Rahman, A., "Molecular-dynamics study of atomic motions in water", *Phys. Rev. B* **1985**, *31*, 2643–2648.

[69] Robinson, G. W., Singh, S, Zhu, S.-B., Evans, M. W., *Water in biology, chemistry and physics : Experimental overviews and computational methodologies*, *Vol. 9*, World Scientific, **1996**.

[70] Yuet, P. K., Blankschtein, D., "Molecular dynamics simulation study of water surfaces: comparison of flexible water models", *J. Phys. Chem. B* **2010**, *114*, 13786–13795.

[71] Jones, E., Oliphant, T., Peterson, P., "SciPy: Open Source Scientific Tools for Python", **2001**.

[72] van Rossum, G., Python tutorial, Report CS-R9526, **1995**.

[73] Nelder, J. A., Mead, R., "A simplex method for function minimization", *Comput. J.* **1965**, *7*, 308–313.

[74] Sega, M., Dellago, C., "Long-range dispersion effects on the water/vapor interface simulated using the most common models", *J. Phys. Chem. B* **2017**, *121*, 3798–3803.

[75] Rowlinson, J. S., Widom, B., *Molecular theory of capillarity*, Clarendon Press, **1984**.

[76] Kirkwood, J. G., Buff, F. P., "The statistical mechanical theory of surface tension", *J. Chem. Phys.* **1949**, *17*, 338–343.

[77] Alejandre, J., Tildesley, D. J., Chapela, G. A., "Molecular dynamics simulation of the orthobaric densities and surface tension of water", *J. Chem. Phys.* **1995**, *102*, 4574–4583.

[78] Thompson, A. P., Plimpton, S. J., Mattson, W., "General formulation of pressure and stress tensor for arbitrary many-body interaction potentials under periodic boundary conditions", *J. Chem. Phys.* **2009**, *131*, 154107.

[79] Nagata, Y., Pool, R. E., Backus, E. H., Bonn, M., "Nuclear quantum effects affect bond orientation of water at the water-vapor interface", *Phys. Rev. Lett.* **2012**, *109*, 226101.

[80] Shinoda, W., Shiga, M., Mikami, M., "Rapid estimation of elastic constants by molecular dynamics simulation under constant stress", *Phys. Rev. B* **2004**, *69*, 134103.

[81] Hockney, R. W., Eastwood, J. W., *Computer simulation using particles*, Taylor & Francis, Inc., **1988**.

[82] Knuth, D. E., *The art of computer programming, volume 2 (3rd ed.): Seminumerical algorithms*, Addison-Wesley Longman Publishing Co., Inc., **1997**.

[83] Wu, Y., Tepper, H. L., Voth, G. A., "Flexible simple point-charge water model with improved liquid-state properties", *J. Chem. Phys.* **2006**, *124*, 024503.

[84] Blum, V., Gehrke, R., Hanke, F., Havu, P., Havu, V., Ren, X., Reuter, K., Scheffler, M., "Ab initio molecular simulations with numeric atom-centered orbitals", *Comput. Phys. Commun.* **2009**, *180*, 2175–2196.

[85] Wilding, N. B., "Critical-point and coexistence-curve properties of the Lennard-Jones fluid: A finite-size scaling study", *Phys. Rev. E* **1995**, *52*, 602–611.

[86] Vargaftik, N., Volkov, B., Voljak, L., "International tables of the surface tension of water", *J. Phys. Chem. Ref. Data* **1983**, *12*, 817–820.

[87] Schienbein, P., Marx, D., "Liquid-vapor phase diagram of RPBE-D3 water: Electronic properties along the coexistence curve and in the supercritical phase", *J. Phys. Chem. B* **2017**, *122*, 3318–3329.

[88] Nagata, Y., Ohto, T., Bonn, M., Kühne, T. D., "Surface tension of ab initio liquid water at the water-air interface", *J. Chem. Phys.* **2016**, *144*, 204705.

[89] Hirshfeld, F. L., "Bonded-atom fragments for describing molecular charge densities", *Theor. Chem. Acc.* **1977**, *44*, 129–138.

Danksagung

Zuallererst möchte ich mich bei Prof. Christoph Dellago bedanken, der er es mir überhaupt erst ermöglicht hat meine Masterarbeit in der Forschungsgruppe für computergestützte Physik zu verfassen und der mir all die nötige Zeit und die notwendigen Ressourcen zur Verfügung gestellt hat, die ich gebraucht habe. Ich möchte mich auch bei den anderen Mitgliedern der Gruppe für computergesteuertützte Physik bedanken, von denen ich besonders Dr. Andreas Singraber und Dr. Marcello Sega hervorheben möchte, die mich immer mit Rat und Tat unterstützt haben.
Mein größter Dank gilt aber meinen Eltern, die meine Ideen und Entscheidungen nicht nur anerkannt, sondern stets gefördert haben. Ohne sie wäre ein so intensives Physikstudium, das ich sehr genossen habe, nicht möglich gewesen!

Zusammenfassung

Künstliche neuronale Netze können anhand der Ergebnisse komplexer und sehr aufwendiger numerischer Verfahren zur quantenmechanischen Berechnung der Energie und der daraus resultierenden Kräfte von Atomanordnungen trainiert werden, um diese Energien und Kräfte approximativ zu bestimmen. Mit den trainierten künstlichen neuronalen Netzen ist es im Weiteren möglich Molekulardynamiksimulationen durchzuführen. Dies wurde, unter vielen anderen Beispielen, in den vergangenen Jahren auch mit Systemen von Wassermolekülen im flüssigen und festen Zustand durchgeführt.

In dieser Arbeit wurde die gleiche Methode auf die Oberfläche zwischen flüssigem Wasser und Wasserdampf angewendet, wobei zur Berechnung der für das Training benötigten Energien und Kräfte die sogenannte Dichtefunktionaltheorie (RPBE-D3-Funktional) zum Einsatz kam. Außerdem wurde untersucht, inwieweit für solche Systeme zuverlässige Schlüsse aus Simulationen basierend auf neuronalen Netzen zu erwarten sind. Dazu bedienten wir uns empirischer Modelle für Wassermoleküle, die um viele Größenordnungen effizienter auswertbar sind als die erwähnten quantenmechanischen Verfahren. Diese Effizienz ermöglicht es die Ergebnisse von Simulationen zu vergleichen, die einerseits auf Basis eines empirischen Wassermodells und andererseits mit Hilfe von künstlichen neuronalen Netzen, trainiert auf eben dieses Wassermodell, durchgeführt werden. Die Resultate weisen gute Übereinstimmung miteinander auf und lassen vermuten, dass vor allem für Simulationen von Oberflächen mit Hilfe von künstlichen neuronalen Netzwerken, welche bei niedrigeren Temperaturen durchgeführt werden, Ergebnisse von viel aufwendigeren quantenmechanischen Simulationen mit guter Genauigkeit reproduziert werden können.