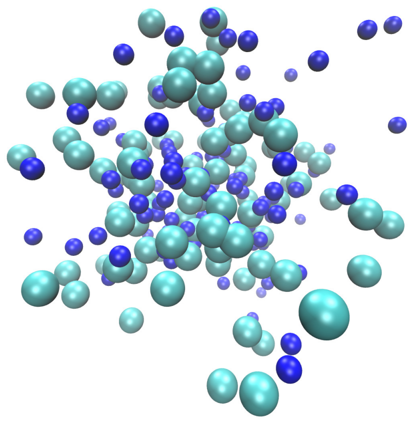
Identifying Failures in Machine Learning Potentials

Master Thesis

Supervisor: Prof. Dr. Christian Holm
Supervisor: Prof. Dr. Eric Lutz



Marco Brückner (Mat Nr.: 3120152)
Email: marco.brueckner97@web.de

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt und indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Ich weiß, dass die Arbeit in digitalisierter Form daraufhin überprüft werden kann, ob unerlaubte Hilfsmittel verwendet wurden und ob es sich – insgesamt oder in Teilen – um ein Plagiat handelt. Zum Vergleich meiner Arbeit mit existierenden Quellen darf siein eine Datenbank eingestellt werden und nach der Überprüfung zum Vergleich mit künftig eingehenden Arbeiten dort verbleiben. Weitere Vervielfältigungs - und Verwertungsrechte werden dadurch nicht eingeräumt. Die Arbeit wurde weder einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht.

Ort, Datum	Marco Brueckner

Contents

1. Abstract

We look into different potential causes, that can be responsible for faulty Machine Learned Potential Energy Surfaces. First we explore the impact of long range interactions by comparing two systems with different potential cutoffs in the context of NPT simulations. Then we investigate the influence of the quantities that the potential is being trained on. Next we vary the particle density of the training data and analyze the consequences for NPT simulations. Then we try to identify which configurations can cause a Machine Learning simulation to fail. This is done by trying to predict when the simulation will fail by observing the physical quantities. We compare the configuration spaces of classical and DFT simulations for Argon, NaCl and $BMIMBF_4$ by using the REMatch kernel for configuration comparisons. We try to solve the problem of having configurations that cause simulations to fail by making the Machine Learned potential robuster using active learning and implement an active learning algorithm into MLSuite.

2. Introduction

In order to create better and faster Molecular Dynamics (MD) simulations, either the computer hardware has to improve or physicists have to come up with new clever approaches and algorithms. One direction that looks promising for these kind of advances is the field of Machine Learning. Machine Learning (ML) is a subfield of artificial intelligence. It deals with computer algorithms, that improve with the amount of learning data. Nowadays ML can be found in many places like pattern recognition, medical diagnosis, stock market trading, self driving cars, and many more [22]. ML has also found a way to help the scientific process. There are parts of the scientific method like observation and the building of hypothesis, where the amount of data can be too much to handle for human perception alone. There ML is used for sensing in satellites, microscope imagery in cell research and in several high data-rate instruments [21].

In physics ML is mainly used to classify data and predict properties. In experimental particle physics the application of pattern recognition has been part of event detection for a long time [21].

In recent years it has become popular to use ML for predicting potential energies from Molecular Dynamics (MD) data points [3, 4, 8]. ML potentials try to fit the very high dimensional potential energy function, that depends on all the particle coordinates. This is done by interpolating between the data points provided in a training data set. This process is often called training the potential. Training a potential is an initial cost, but in return there is no need for on-the-fly energy calculations during the simulation. There are many applications for Potential energy surfaces. In quantum chemistry they are used for the modeling of molecules or for bigger coarse grained systems [31].

ML PES are always going to be less accurate than the function they are trying too interpolate. The goal is to sacrifice only a little bit of accuracy while improving the simulation speed by a lot. If the data points are sampled from ab initio MD, then Machine Learning potentials have the ability to simulate physical systems significantly more accurately than classical Molecular Dynamics simulations and be orders of magnitude faster than simply running a pure ab initio MD simulation.

Despite big progress over the last 20 years ML potentials still have problems [32]. There are issues considering the long range interactions, because the ML algorithm only considers particles, that are within a certain cutoff. This can lead to incorrect predictions of the force and of the pressure. There are also still questions on what the best way to create training data is and how to make sure that the ML simulation avoids encountering a configuration, that it can not predict correctly. There are approaches to address this like Active Learning. Here the algorithm interacts with the user to label new data points. By choosing the right configurations the potential can be made more robust and the resulting simulation might therefore be successful.

Part I.

Theory

3. Molecular Dynamics Simulations

3.1. Classical Molecular Dynamics Simulations

Molecular Dynamics (MD) is a method for computing properties of many body systems. It is a method to describe the movement of atoms and molecules. The word 'Classical' in this context means that the trajectories of the particles are calculated by solving Newton's equations of motion. There are many systems, where this is a good approximation. Most systems, that are studied using MD consist of hundreds or thousands of particles and are so complex that no analytical solutions to Newton's equations can be found. MD simulations can be viewed as a tool that can give answers to question that might be too hard to find with an exact approach.
Before starting the simulation the system needs to be prepared, meaning the particles need to be placed and given properties like mass and charge. Then the system is 'integrated', meaning that the forces acting on every particle are calculated and then the particles are moved obeying Newtons equations of motion. After enough integration steps the system reaches an equilibrium, where the macroscopic properties fluctuate about an average. In this state measurements can be taken. There are many reasons for why a particle simulation might not produce the desired outcome. The system can be prepared incorrectly, model parameters can be chosen poorly, the system can be too small or the measurement time too short.

The most time consuming part of most classical MD simulations is the force calculation. If we consider a system with pair-potentials and there are N particles, then there are $N \cdot (N-1)/2$ particle interactions. The force calculation therefore scales with $\mathcal{O}(N^2)$. There are methods like verlet-lists [13] that improve the scaling for short range interactions by making use of the fact that interactions of particles far away from each other can be neglected. Choosing what the particle interactions look like is also one of the most challenging parts of a classical MD simulation. Often empirical models, so called force fields are used, in which simple mathematical models describe Van der Waals and electrostatic interactions as well as bond and angle potentials. The force field parameters are fit to experimental data

Once all the forces acting on the particles are calculated, the integration step can be performed. A popular choice is to do this is the Velocity-Verlet algorithm [13]

1. Locate the particles and give them velocities

2. Move the particles to their new position

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{2}\vec{a}_i(t)(\Delta t)^2 \tag{3.1}$$

3. Calculate the intermediate velocity

$$\vec{v}(t+\frac{1}{2}\Delta t) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\Delta t \tag{3.2}$$

4. Compute the new acceleration $\vec{a}(t+\Delta t)$ from the forces resulting of the new postitions $\vec{F}(\vec{r}(t+\Delta t))$

5. Calculate the new velocities

$$\vec{v}(t+\Delta t) = \vec{v}(t+\frac{1}{2}\Delta t) + \frac{1}{2}\vec{a}(t+\Delta t)\Delta t \tag{3.3}$$

Verlet-style algorithms are therefore not the most accurate when it comes to calculating exact trajectories. However they have little long-term drift of the energy, which makes them valuable for equilibrium measurements. This stands in contrast to higher order algorithms, which are more accurate in the short-term but often have energy drifts in the long-term [13].

3.2. Ab Initio Molecular Dynamics Simulations

Despite the success of force fields, there are still many drawbacks. In standard classical MD simulations charges are static and therefore electronic polarization effects are not naturally included. If polarization effects want to be included, then an additional model needs to be added and the parameters for the model need to be chosen by the user. While some processes can be simulated this way, there are problems like lack of transferability and standardization. Also using Classical MD there is no easy way to simulate chemistry i.e. forming and breaking of molecules; again in order to do this new models and assumptions have to be added [30].

Choosing parameters for models is somewhat arbitrary and not directly physically motivated. These problems can be avoided with ab initio molecular Dynamics (AIMD). AIMD calculates forces from electronic structure calculations and therefore includes many-body forces, electronic polarization and breaking and formation of bonds. While classical MD tries to solve Newton's equation of motion, AIMD is based on the Schrödinger equation and uses real physical potentials [23].

The advantages and high accuracy of AIMD comes at a cost. While it is feasible for classical MD to simulate up to millions of atoms on the timescales of nanoseconds, AIMD is generally limited to thousands of atoms in the picoseconds range [30]. The accuracy of AIMD depends on the electronic structure method that is used. The most popular electronic structure method is density functional theory (DFT) using the Kohn-Sham equations.

Ab initio methods consider the N nuclei described by the coordinates $\vec{R}_1, ..., \vec{R}_N = \vec{R}$ with the momenta $\vec{P}_1, ..., \vec{P}_N$ and the masses $M_1, ..., M_N$. The N_e electrons are also considered and are described by the coordinates $\vec{r}_1, ..., \vec{r}_{Ne} = \vec{r}$ and momenta $\vec{p}_1, ..., \vec{p}_N$. The Hamilton operator for such a general system reads [30]

$$H = \sum_{I=1}^{N} \frac{\vec{P}_I^2}{2M_I} + \sum_{i=1}^{N_e} \frac{\vec{p}_I^2}{2m} + \sum_{i>j} \frac{e^2}{|\vec{r}_i - \vec{r}_j|} + \sum_{I>J} \frac{Z_I Z_J e^2}{|\vec{R}_i - \vec{R}_j|} - \sum_{i,I} \frac{Z_I e^2}{|\vec{R}_I - \vec{r}_i|} + V_{\text{ext}} \tag{3.4}$$

where m is the mass of the electron and Z is the charge of the respective nucleus. In order to solve the Schrödinger equation the eigenvectors and eigenvalues need to be found. In general it is not possible to find an analytic solution because the equation is very complicated. That is why approximations are made. First the Born-Oppenheimer approximation is applied, which assumes that the nucleus and electron motions are separable [30]. Several more approximations are applied like the adiabatic approximation, which assumes that the electronic wave function adepts to a configuration of the nuclei instantaneously. In the end, equations for the motion of the nuclei can be derived [30]

$$\dot{\vec{R}}_I = \frac{\vec{P}_I}{M_I} \tag{3.5}$$

$$\dot{\vec{P}}_I = -\nabla_I(\epsilon_0(\vec{R}) + V_{NN}(\vec{R})) \tag{3.6}$$

that can be used to integrate the system, $\epsilon_0(\vec{R})$ is the electronic ground state energy and $V_{NN}(\vec{R})$ is the nuclei repulsion term.

3.2.1. Density Functional Theory

Density functional theory (DFT) is based on the Hohenberg-Kohn theorem, which says that every ground state density matches exactly one potential. From this follows that it is possible to minimize a functional $\epsilon[n]$ with respect to the electron density $n(\vec{r})$ [6]. DFT defines a set of self consistent equations, the so called Kohn-sham equations, that must be solved [6]. Kohn-Sham DFT uses the idea of an artificial reference system of non-interacting fictitious particles in a potential $V_{KS}(\vec{r}, \vec{R})$. The resulting ground state energy and density is equal to the real interacting system. A set of orthonormal single particle orbitals $\Psi_i(\vec{r})$ are introduced, the so called Kohn-Sham orbitals [30]

$$n(\vec{r}) = \sum_i f_i |\Psi_i(\vec{r})|^2 \tag{3.7}$$

resulting in a particle density $n(\vec{r})$. f_i indicate the occupation numbers of the orbitals $\sum_i f_i = N_e$. The functional has the form [30]

$$\epsilon[\psi_i] = -\frac{\hbar^2}{2m} \sum_i f_i \langle \psi_i | \nabla^2 | \psi_i \rangle + \frac{e^2}{2} \int d\vec{r} d\vec{r}' \frac{n(\vec{r}) n(\vec{r}')}{|\vec{r} - \vec{r}'|} + \epsilon_{xc}[n] + \int d\vec{r} n(\vec{r}) V_{eN}(\vec{r}, \vec{R}) \tag{3.8}$$

where the first term describes the quantum kinetic energy, the second term describes the Coulomb interaction, the third term is the exchange correlation energy and the fourth term describes the interaction of the electrons with the nuclei. The Kohn-Sham potential is described by [30]

$$V_{KS}(\vec{r}, \vec{R}) = \frac{e^2}{2} \int \frac{n(\vec{r'})}{|\vec{r} - \vec{r'}|} d\vec{r'} + \frac{\delta \epsilon_{xc}}{\delta n(\vec{r})} + V_{eN}(\vec{r}, \vec{R}). \tag{3.9}$$

The solution to the Kohn-sham equations [30]

$$H_{KS}(\Psi_i(\vec{r})) = -\frac{\hbar^2}{2m}\nabla^2 + V_{KS}(\vec{r}, \vec{R}) = \epsilon_i \Psi_i(\vec{r}) \tag{3.10}$$

result in the Kohn-Sham orbitals $\Psi_i(\vec{r})$ with the energies ϵ_i. The Kohn-Sham equations are a self-consistent cycle, because the Kohn-Sham equations can be used to calculate a new density with equation 3.7, that can then again be used to calculate new Kohn-Sham orbitals with equation 3.10. This formalism is in principle exact, but the form of the exchange correlation energy ϵ_{xc} is unknown and therefore approximations have to be made. Local density approximation (LDA) calculates the exchange correlation energy for a homogenous electron gas. This works well for systems like metals where there are only small inhomogenities in the electron densitiy. But it fails for inhomogenous systems for example molecules with hydrogen bonds. Another approach is the generalized gradient approximation (GGA), which also takes into account the gradient of the density, making the functional more adaptive [30].

3.3. Machine Learning Molecular Dynamics

The underlying physical laws that describe a large part of physics and chemistry have been known for a long time. The equations resulting from these laws are often so complicated that no analytic solutions can be found. It is therefore desirable to develop methods that find approximative solutions for quantum mechanical problems, that do not need a lot of computation. In the past years the field of Machine Learning (ML) has been applied to find new efficient solutions. Machine Learning is part of artificial intelligence and it studies computer algorithms that improve with experience. The algorithm becomes more successful with the use of data. ML tries to extract patterns from data and uses them to make predictions and carry out tasks [22]. A popular example to teach the concept of ML is image recognition. In this case input data is given in the form of pixel values and the task is to assign a label that describes the image. The ML model is trained on pictures whose labels the algorithm knows. This is an example of Supervised Learning, the task of learning given training input and output pairs. The unique characteristic of Machine Learning is that the user ends up with an algorithm that can solve a problem without ever needing specific programming or complete understanding of the task [22].

For ML MD this technique can be applied in an analogous way. The positions of the particles, also called configurations, can be used as input data and the corresponding energies, forces and pressures constitute the labels. In this way the ML model can compare unknown configurations to the ones whose energies, forces and pressures it knows and make predictions accordingly.

The accuracy of Molecular Dynamics simulations is limited by the quality of the potential energy surface (PES). Therefore it is important to use a precise potential function $V(\vec{r})$, that describes the PES. The potential $V(\vec{r})$ is always a function of the nucleus coordinates. It is possible to either calculate the potential energy on the fly during the simulation or to calculate the PES before conducting the simulation. The first approach has the advantage that the potential energy only needs to be evaluated for points, that are relevant for the simulation. Calculating the PES beforehand can be expensive, but once it is done all subsequent simulations can be much faster. The most accurate way to calculate the potential is to use ab initio methods, but this is slow and calculating enough points to get a smooth surface is generally not feasible. Good approximations for the PES are therefore desirable. This is where ML can be used. Function interpolation is an area that ML has had a lot of success in [7].

There are currently two main approaches for constructing PESs with ML, nonlinear kernel methods and deep neural networks. A few examples of kernel methods are nonlinear support vector machines, kernel ridge regression and Gaussian processes. [17] Kernel methods often apply the input output relationship directly. For instance if there are N function values $\vec{y}$ at positions $\vec{x}$, it is possible to make predictions using linear combinations of the $\vec{y}$ values. Because of this kernel methods generally scale as $\mathcal{O}(N^2)$ to $\mathcal{O}(N^3)$ with the input data. Neural Networks generally scale better but also need more training data.

	Gaussian Process	Neural Network
Scaling with the input data	$\mathcal{O}(N^2)$ to $\mathcal{O}(N^3)$	varies but can be fast
Scaling of the evaluation	$\mathcal{O}(N^2)$	fast, independent of N
Required amount of input data N	few	many
Overfitting	Rare, only for very complex kernels	Likely
Very high accuracy	difficult	easy

[17]

Running the simulations with a PES is generally orders of magnitudes faster than ab initio methods because the expensive step of solving the Schrödinger equation becomes redundant. Kernel methods are still generally a lot slower than classical MD because the representations and the linear combinations need to be evaluated. The speed of NN simulations depend on its size but are generally faster than kernel methods but slower than classical simulations. As a summary, it can be stated that ML simulations offer a middle ground between the very accurate but slow ab initio methods and the very fast but inaccurate classical simulations. If the ML model constructs an accurate PES, then a lot of time can be saved while only a little accuracy is sacrificed.

3.4. Material Properties

Once the MD simulation has reached an equilibrium, we can extract the material properties that we are interested in. They can be used to analyze and study the system. In this work material properties are mainly used to compare to literature values or to compare two systems with each other, thus making sure that the simulations were conducted successfully. There are mainly two properties used, the radial distribution function to examine the structure of the simulation and the Diffusion coefficient for checking a dynamic physical quantity.

3.4.1. Diffusion Coefficient

Diffusion happens due to thermal motion. It smooths the concentration of particles. For example, the phenomenon can be observed when ink is dropped into water. The ink mixes with the water until the color of the liquid is uniform. The macroscopic law describing diffusion is called Fick's law [13]

$$\vec{j}(\vec{r},t) = -D\nabla c(\vec{r},t) \tag{3.11}$$

where D is the diffusion coefficient, $\vec{j}$ is the current density and c is the concentration of particles. To calculate the time dependence of c, we have to consider that the total amount of particles remains constant [13]

$$\frac{\partial c(\vec{r},t)}{\partial t} = -\nabla \cdot \vec{j}(\vec{r},t). \tag{3.12}$$

Fick's law 3.11 can be combined with the conservation equation 3.12 to result in the Diffusion equation

$$\frac{\partial c(\vec{r},t)}{\partial t} = -D\nabla^2 c(\vec{r},t). \tag{3.13}$$

This equation can be solved by using the boundary condition

$$c(\vec{r},0) = \delta(\vec{r}) \tag{3.14}$$

resulting in a time dependent particle concentration [13]

$$c(\vec{r},t) = \frac{1}{(4\pi Dt)^{\frac{3}{2}}} \exp\left(-\frac{r^2}{4Dt}\right). \tag{3.15}$$

This information can now be used to compute the mean squared displacement

$$\langle r^2 \rangle = \int r^2 c(\vec{r},t)\mathrm{d}\vec{r} \tag{3.16}$$

getting a final result of [13]

$$\langle r^2 \rangle = 6Dt. \tag{3.17}$$

3.4.2. Radial Distribution Function

The radial distribution function $g(\vec{r})$ describes how the particle density varies with the distance form a reference particle. For an ideal homogeneous gas the radial distribution function will simply be constant $g(\vec{r}) = N/V$. Real fluids always have variations in the density due to particle interactions. The radial distribution function states how likely it is to find a particle at a specific distance compared to an ideal gas.

The radial distribution function is interesting because it can be compared to literature values that are the result of X-ray scattering or neutron scattering experiments. $g(\vec{r})$ also plays an important role in liquid state theory and scattering experiments and simulations therefore can be used to assess them. Calculating $g(\vec{r})$ in a simulation is straight forward, because the exact positions of all the particles are known.

4. Classical Potentials

4.1. Born-Mayer-Huggins-Tosi-Fumi Potential

The Born-Mayer-Huggins-Tosi-Fumi (BMHTF) Potential describes the interactions of NaCl-type alkali halides [29]

$$V_{\alpha\beta}(r) = A_{\alpha\beta} \exp\left(\frac{\sigma_{\alpha\beta} - r}{\rho_{\alpha\beta}}\right) - \frac{C_{\alpha\beta}}{r^6} + \frac{D_{\alpha\beta}}{r^8} \text{ for } r < r_c \tag{4.1}$$

where σ is interaction dependent length parameter, ρ is an ionic-pair dependent length parameter and r_c is the cutoff. The first term describes the short-range Pauli repulsion, the second term represents the induced dipole-dipole van der Waals interaction and the last term describes the induced dipole-quadrupole interaction [33]. The NaCl lammps simulation use the following parameters [33]

	A (eV)	ρ ($\AA$)	σ ($\AA$)	C (eV $\cdot\AA^6$)	D (eV $\cdot\AA^8$)
Na-Na	424.097	0.317	0	1.05	0.499
Na-Cl	1256.31	0.317	0	7.0	8.676
Cl-Cl	3488.998	0.317	0	72.5	145.427

4.2. Coulomb and Wolf Potential

Coulomb's law describes the force acting between two stationary charged particles

$$\vec{F_1} = \frac{1}{4\pi\epsilon_0} \frac{Q_1 Q_2}{r_{12}^2} \vec{r_{12}} \tag{4.2}$$

with the charges Q_1, Q_2 and the vacuum permittivity ϵ_0. In a MD simulation the coulomb forces between all particles and image particles are summed up. This is often done using a pppm algorithm, which sums up the long range contributions using a Fourier sum.

The Wolf potential sums up the coulomb contributions via the Wolf sum:

$$E_i = \frac{1}{2} \sum_{i \neq j} \frac{q_i q_j \operatorname{erfc}(\alpha r_{ij})}{r_{ij}} + \frac{1}{2} \sum_{j \neq i} \frac{q_i q_j \operatorname{erf}(\alpha r_{ij})}{r_{ij}} \; r < r_{cut} \tag{4.3}$$

the interactions are only computed for a radius smaller than the cutoff radius r_{cut}. erf and erfc are the error function and the complementary error function and α is a damping parameter. The Wolf potential is essentially a range-limited, spherically-truncated, charge-neutralized, shifted and pairwise 1/r summation.

5. Gaussian Approximation Potentials

5.1. Gaussian Processes

A Gaussian process generates data, that can be described by a multivariate Gaussian distribution. A multivariate Gaussian distribution is a generalization of the one dimensional normal distribution. Multivariate Gaussian distributions $N(\mu, \sigma)$ are characterized by the k-dimensional mean vector μ and the k×k covariance Matrix Σ_k

$$f_{\vec{X}}(x_1, ..., x_k) = \frac{\exp(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \mathbf{\Sigma}^{-1}(\vec{x} - \vec{\mu}))}{\sqrt{(2\pi)^k |\Sigma|}}. \tag{5.1}$$

If there are n points in a dataset $\vec{y} = y_1, y_2, ..., y_n$, then this set can be imagined being generated from some n-variant Gaussian distribution. This explains why Gaussian Processes (GP) can be used in so many different places. The covariance function, that constructs the covariance matrix can be chosen. A popular choice is the squared exponential [11]

$$k(\vec{x}, \vec{x'}) = \sigma_f^2 \exp\left(\frac{-(\vec{x} - \vec{x'})^2}{2l^2}\right). \tag{5.2}$$

The maximum and the width are described by the hyper-parameters σ_f and l, that can also be chosen. The covariance function has the role of a similarity measure between two input points. For instance, using the the squared exponential, two vectors have a large covariance if they are placed closely together, meaning the distance between them is small. The data $\vec{y}$ is often noisy, which needs to considered in the GP. This is done by adding a Gaussian with a standard deviation σ_n to the covariance matrix [11]

$$K = \begin{pmatrix} k(\vec{x_1}, \vec{x_1}) + \sigma_n^2 & k(\vec{x_1}, \vec{x_2}) & \cdots & k(\vec{x_1}, \vec{x_k}) \\ k(\vec{x_2}, \vec{x_1}) & k(\vec{x_2}, \vec{x_2}) + \sigma_n^2 & \cdots & k(\vec{x_2}, \vec{x_k}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\vec{x_k}, \vec{x_1}) & k(\vec{x_k}, \vec{x_2}) & \cdots & k(\vec{x_k}, \vec{x_k}) + \sigma_n^2 \end{pmatrix}. \tag{5.3}$$

5.2. Gaussian Process Regression

Regression is the process for estimating the relationship between a dependent variable and one or more independent variables. Regression is used to predict the values of a function given input data. The most well known form of regression is linear regression, where a line $y = a \cdot x + c$ is found that best fits the data. Gaussian Process Regression (GPR) is a more general approach that does not assume a concrete functional form. This is one of the main strength of GPRs compared to other fitting methods, as in a lot of cases there is no way to find a functional form. We need to define two more matrices [11] for the GPR

$$K_* = [k(x_*, x_1), k(x_*, x_2), ..., k(x_*, x_k)] \tag{5.4}$$
$$K_{**} = k(x_*, x_*) \tag{5.5}$$

where x_* is the target point we want to make the prediction for. The basic assumption of GPR is that the data can be viewed as a sample from a multivariate Gaussian distribution, it holds

$$\begin{pmatrix} \vec{y} \\ y^* \end{pmatrix} \sim \; N\left(0, \begin{pmatrix} K & K_*^T \\ K_* & K_{**} \end{pmatrix}\right). \tag{5.6}$$

We want to know the conditional probability $p(y_*|\vec{y})$, which means that knowing the data how probable is a certain prediction for the point y_*. The probability is described by a Gaussian distribution

$$y_*|\vec{y} \sim \; N(K_* K^{-1}\vec{y}, K_{**} - K_* K^{-1} K_*^T) \tag{5.7}$$

where the best prediction for y_* is given by the mean of this distribution [11]

$$\overline{y_*} = K_* K^{-1}\vec{y}. \tag{5.8}$$

For many ML methods it is quite difficult to come up with estimates for the error of the predictions. An advantage of GPRs is that an error estimate is built into the method. The uncertainty for the predictions is given by the variance

$$\text{var}(y_*) = K_{**} - K_* K^{-1} K_*^T. \tag{5.9}$$

5.3. An Illustration

In Machine Learning Molecular Dynamics, Gaussian Processes are used to fit the very high dimensional potential energy function that depends on the coordinates of every particle in the system. For the purpose of better understanding this process it is helpful to look at a simpler one dimensional example. We want to use GPR to estimate the value of the point x=5.6 in figure 5.1 (a). The data points are created using a polynomial $20x - 1.4x^2 + 0.025x^3$. In this case it makes sense to use the squared exponential kernel introduced in equation 5.2. The squared exponential gives points with a similar x-value a lot of weight and the target point will therefore have a similar y-value to the points around it. This is desirable for a continuous function. There are 30 observations at the positions

$$\vec{x} = (0, 1.2, 2.4, 3.6....) \tag{5.10}$$

and $\sigma_n = 16$ is given by the error bars. In Python, data points can easily be created using numpy.

```python
import numpy as np
import matplotlib.pyplot as plt
x_array = np.linspace(0, 35, 30)

def test_polynom(x):
    return 20 * x - 1.4 * x**2 + 0.025 * x**3

y_array = test_polynom(x_array)
```

First we need to chose values for the hyper-parameters σ_f(sigma_ f) and l(l) of the kernel function. The noise σ_n(sigma_ n) should reflect the accuracy of the data, it can be related to size of the errorbars. The target point is located at $x_* = 5.6$(x_ star).

```python
sigma_f = 10
l = 5
sigma_n = 4
x_star = 5.6
```

The next step is to compute the covariance matrix as described in equation 5.3. This is done by computing the kernel function for every combination of data points. Points that are on the main diagonal all have the value 116, which matches $\sigma_n^2 + \sigma_f^2$. Points that are not on the main diagonal are smaller than the maximum of 116, since every point is most similar to itself.

```python
def kernel_function(x1, x2):
    return sigma_f**2 * np.exp(-(x1 - x2)**2/(2 * l**2))

def compute_covariance_matrix(x):
    matrix = np.ones((len(x), len(x)))
    for row in np.arange(len(x)):
        for column in np.arange(len(x)):
            matrix[row, column] = kernel_function(x[row], x[column])
    return matrix

def compute_noise_matrix(x):
    return np.diag(np.ones(len(x)) * sigma_n**2)

covariance_matrix = compute_covariance_matrix(x_array)
noise_matrix = compute_noise_matrix(x_array)
covariance_matrix = covariance_matrix + noise_matrix
inverse_matrix = np.linalg.inv(covariance_matrix)
```

$$K = \begin{pmatrix} 116 & 97 & 89 & \dots \\ 97 & 116 & 97 & \dots \\ \vdots & \ddots & \ddots & \vdots \\ \dots & 97 & 116 & 97 \\ \dots & 89 & 97 & 116 \end{pmatrix}. \tag{5.11}$$

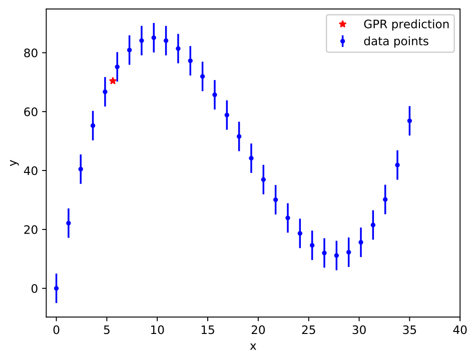
The covariance matrix has dimensions $N \cdot N$, where N is the number of data points. The next step is to compute K_* and K_{**} as described in equation 5.4 and 5.5. K_* describes the similarity of the target point with the data points and K_{**} is the kernel function of the target point with itself.

```
def compute_k_star(x_star, x):
    k_star = []
    for index in range(len(x)):
        k_star.append(kernel_function(x_star, x[index]))
    return np.array(k_star)

k_star = compute_k_star(x_star, x_array)
k_star_star = kernel_function(x_star, x_star)
```

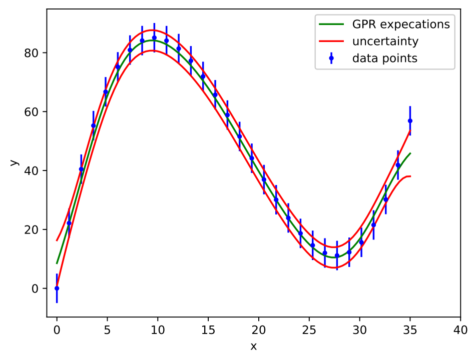
This is all the information needed to compute the expectation value $\overline{y}_* = 70.4$ with equation 5.8 and the uncertainty $\text{var}(y_*) = 3.56$ with equation 5.9.

```
y_expectation = np.dot(k_star, inverse_matrix @ y_array)
y_uncertainty = kernel_function(x_star, x_star) - np.dot(k_star,
    inverse_matrix @ k_star)
```

Function interpolation can be done by repeating the process of predicting values for target points as depicted in Figure 5.1 (b). The predictions for 100 points are plotted and the expectation values are enclosed by an uncertainty interval. The predictions made by the GPR are a lot more accurate in areas where there is a high density of data points. At the edges the predictions are off by a significant amount. This also explains why GPR is good at interpolating but not so good at extrapolating.



(a) Prediction for a single point



(b) Interpolation and Uncertainty

Figure 5.1.: Gaussian Process Regression for a one-dimensional example. Figure (a) shows the GPR prediction for a single point at x=5.6. Figure (b) shows the GPR interpolation by plotting 100 predicted values. The predictions are enclosed by the uncertainty indicated by the red lines.

6. Descriptors

Choosing a good representation of the input coordinates is essential for algorithms in computational chemistry and physics [2]. They are important to identify comparable structures for example to characterize molecules. Cartesian coordinates are a simple way of describing the positions of atoms, but they are not useful when it comes to comparing configurations. A rotation of the system will completely change the cartesian coordinates, while the physics remains the same (rotational invariance).

Descriptors can be classified by two categories. Local representations describe atomic neighborhoods, only a part of the system. Global representations describe the whole system. Some local descriptors include Smooth Overlap of Atomic Positions (SOAP) [3], Symmetry Functions (SF) [5], Moment Tensor Potentials [25] and the Bispectrum [19]. Local descriptors are fit for calculating local properties such as forces or nuclear magnetic resonance shifts [19]. Global properties can also be approximated with local descriptors by summing over atomic contributions. For local descriptors it does not matter whether the system is finite or periodic. Some global representations include the Coulomb Matrix [26], histograms of distances, angles and dihedral angles [12] and the Many Body Tensor Representation [16]. Global descriptors are fit for calculating global properties like the energy, band gap and the polarizability tensor. Many systems are periodic and therefore infinitely large. Global descriptors therefore need to be adapted for the application to these systems.

Descriptors, also called representations, are a transformation of the input coordinates. Typically they transform information in the form of cartesian coordinates into a form that is rotationally, translationally and reflection invariant. Atoms of the same kind can also be exchanged without changing the representation, this also called permutational invariance. Ideally a descriptor is also unique, meaning the descriptor describes a single configuration and a configuration has only one corresponding descriptor [15]. Additionally the descriptor should also be continuous, meaning that small changes in the configuration lead to small changes in the descriptor, and computationally efficient [15]. Discontinuities are not in line with the assumption, that we want to find the least complex function fitting the data [19].

6.1. Smooth Overlap of Atomic Positions

The Smooth Overlap of Atomic Positions (SOAP) is a descriptor that can be used to describe local environments within a configuration. It first places a sum of Gaussian distributions and then expands them using spherical harmonics. The Gaussians are centered at the locations where atoms are placed

$$\rho^Z(\vec{r}) = \sum_i^Z \exp\left(\frac{-1}{2\sigma^2} \mid \vec{r} - \vec{R} \mid\right). \tag{6.1}$$

The sum runs over every atom with the same atomic number Z to construct a density for every element. The model can be adapted by changing the hyperparameter σ. By setting the origin to $\vec{r} = \vec{0}$, the densities can be expanded using orthonormal radial basis functions g_n and spherical harmonics Y_{lm} [15]

$$\rho^Z(\vec{r}) = \sum_{nlm} c^Z_{nlm} g_n(r) Y_{lm}(\Theta, \Phi) \tag{6.2}$$

where the coefficients are given by

$$c^Z_{nlm} = \int\int\int_{R^3} g_n(\vec{r}) Y_{lm}(\Theta, \Phi) \rho^Z(\vec{r}) \mathrm{d}V. \tag{6.3}$$

The coefficients are used to compute the power spectrum, which is rotationally invariant and is the final result [15]

$$p^{ZZ'}_{nn'l} = \pi \sqrt{\frac{8}{2l+1}} \sum_m \left(c^{Z_1}_{nlm}\right)^* c^{Z_2}_{n'lm}. \tag{6.4}$$

The similarity of two configuration in the SOAP representation are calculated using the kernel [2]

$$k_{SOAP}(i,j) = \sum p^i_{nn'l} \cdot p^j_{nn'l} \tag{6.5}$$

which is simply the dot product of the power spectrum. So in the SOAP representation comparing the similarity of two local environments is simple and inexpensive.

6.2. Atom Centered Symmetry Functions

Atom centered symmetry functions (ACSFs) were introduced to read position information into NNs. ACSFs make use of the cutoff function [5]

$$f_c(R_{ij}) = \begin{cases} 0.5 \cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1 & \text{for } R_{ij} < R_c \\ 0 & \text{for } R_{ij} > R_c \end{cases} \tag{6.6}$$

that goes to zero when the distance R_{ij} between the particles i and j approaches the cutoff radius R_c. The first derivative of the cutoff function also goes to zero while the second derivative does not. By choosing the cutoff radius to be sufficiently large, the discontinuity in the second derivative becomes small. The symmetry functions used consist of two body so called radial terms and three body so called angular terms. There are several options for symmetry functions for the radial part including [5].

$$G_i^1 = \sum_j f_c(R_{ij}), \tag{6.7}$$

$$G_i^2 = \sum_j \exp(-\eta(R_{ij} - R_s)^2) \cdot f_c(R_{ij}), \tag{6.8}$$

and

$$G_i^3 = \sum_j \cos(\kappa R_{ij}) \cdot f_c(R_{ij}). \tag{6.9}$$

The first function G_i^1 is the sum of all cutoff functions of the surrounding particles. The second function G_i^2 is the sum of the cutoff functions multiplied by a gaussian with the parameter η and R_s. By changing the parameter R_s, it is possible to pick out rings of different radii around the center particle. The function therefore contains information about how many particles are at range R_s away from the center particle. To include some information about the angular distributions, the three body functions [5]

$$G_i^4 = 2^{1-\gamma} \sum_{j,k \neq i} (1+\lambda \cos \Theta_{ijk})^\gamma \cdot \exp(-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)) \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}) \tag{6.10}$$

and

$$G_i^5 = 2^{1-\gamma} \sum_{j,k \neq i} (1 + \lambda \cos \Theta_{ijk})^\gamma \cdot \exp(-\eta(R_{ij}^2 + R_{ik}^2)) \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \tag{6.11}$$

are used. The angular part has to be symmetric for an angle of $\Theta_{ijk} = 180°$. The angular resolution can be varied using the γ parameter. High γ values result in fewer nonzero symmetry functions. It is therefore possible to include information for all different angles by including many symmetry functions with different γ values.

6.3. Coulomb Matrix

Another way to describe the environment is the Coulomb Matrix [26]

$$M_{IJ} = \begin{cases} 0.5 Z_I^{2.4} & \text{for} I = J \\ \frac{Z_I Z_J}{|\vec{R}_I - \vec{R}_J|} & \text{for} I \neq J \end{cases} \tag{6.12}$$

where Z is the atomic number and $|\vec{R}_I - \vec{R}_J|$ is the euclidean distance. The diagonal elements are the result of a polynomial fit of atomic energies to the nuclear charge. The off diagonal elements take the shape of a Coulomb repulsion term. The dissimilarity measure is included by taking the euclidean distance

$$d(\mathbf{M}, \mathbf{M}') = \sqrt{\sum_I |\epsilon - \epsilon'|^2} \tag{6.13}$$

of the eigenvalues ϵ.

6.4. Software Implementation using QUIP and GAP

Gaussian Approximation Potentials (GAP) is a software package that uses GPR to fit Potential Energy Surfaces (PES). GAP provides over 20 different descriptors [3] and covariance functions. The GAP framework is embedded in the QUIP package that can be downloaded from Github `https://github.com/libAtoms/QUIP`. QUIP is a molecular simulation code, that is written in FORTRAN. It has interfaces with Python, LAMMPS, CP2K and ASE.

There are many parameters that have to be defined before GAP can fit a model. In this thesis GAP was used with the SOAP and the distance_2b descriptor. The parameters are varied depending on the desired ML Model. An example for the parameters used to create a working NaCl potential is given below.

```
"soap": {
         "n": 5,
         "l": 4,
         "atom_sigma": 1,
         "zeta": 4.0,
         "cutoff": 9.0,
         "delta": 1.0,
         "covariance_type": "dot_product",
         "n_sparse": 500,
         "sparse_method": "CUR_POINTS",
         "add_species": "True"
                             },
"distance_2b": {
         "cutoff": 9.0,
         "covariance_type": "ard_se",
         "delta": 0.1,
         "sparse_method": "CUR_POINTS",
         "add_species": "True",
         "n_sparse": 100,
                             }
```

The SOAP parameters "n" and "l" are the number of radial and angular basis functions, "atom_sigma" is the smearing width of the atom density as described in equation 6.1, "delta" is the scaling per descriptor, "sparse_method" describes the sampling of the representative points, "n_sparse" gives the amount of representative points and "cutoff" is the distance cutoff of the local environment.

7. Descriptor Metrics

The field of computer simulations is becoming more and more complex, involving more data and processing power. Therefore, it is also more and more important to develop tools which help characterize and classify data and make things less confusing. One way to do this, is to introduce a metric. A metric, also called distance function, is a function that returns a distance between two elements of a set. The elements can have arbitrary dimensions while the distance is always a scalar value. A metric therefore extracts information out of something complex and displays it in a simple way. A metric has to satisfy three conditions

$$d(x,y) = 0 \Leftrightarrow x = y \tag{7.1}$$

$$d(x,y) = d(y,x) \tag{7.2}$$

$$d(x,y) \leq d(x,z) + d(z,y) \tag{7.3}$$

the identity of indiscernibles, symmetry and the triangle inequality. In the case of ML MD a metric would be useful if it could compute the distance between configurations and in this way serve as a sort of dissimilarity measure. The most common metric used for this purpose is the root mean square displacement, which computes the Euclidean distance of cartesian coordinates. In recent years a couple of representations have been introduced that contain structural information in a form that already include physical invariances and are therefore better suited than plain cartesian coordinates. It seems logical to base these metrics on one of those new representations, that were developed to compute dissimilarities between atomic environments. The challenge therefore is to find a method that extends the comparisons of atomic environments to comparisons of configurations. Implementations of metrics, that are based on the SOAP descriptor are contained in the DScribe package [15]. The distance between two local environments i, and j can be defined as

$$d(i,j) = \sqrt{2 - 2k_{SOAP}(i,j)} \tag{7.4}$$

where $k_{SOAP}(i,j)$ is the similarity kernel from equation (6.5). This metric is a reduction of two very high dimensional elements, that describe the positions of many particles to a single dimension.

Given two configurations A and B with the same amount of N atoms, it is possible two compute a $(N \times N)$ covariance matrix, that contains the similarity measure of all possible combinations of atomic environments of the two configurations. A straight forward approach is the **Average Structural kernel** [9] that compares the average environment of configuration A with the average environment of configuration B.

$$\overline{K}(A,B) = \left(\frac{1}{N}\sum_i p(i^A)\right) \cdot \left(\frac{1}{N}\sum_j p(j^B)\right). \tag{7.5}$$

This is an inexpensive method to compare the similarity of two configurations. It is positive-definite, since it is based on a scalar product. Now a metric for configurations can be introduced that looks very similar to the SOAP metric for local environments

$$\overline{D}(A,B) = \sqrt{2 - 2\overline{K}(A,B)}. \tag{7.6}$$

By averaging all the local environments a lot of information is lost. It can be the case that two structures appear to be the same, because their differences average out. It is therefore not a very sensitive metric.

It is also possible to try and find the best matching local environments [9] with the **Best Match Kernel**

$$\hat{K}(A,B) = \frac{1}{N}\max_\pi \sum_i C_{i\pi_i}(A,B). \tag{7.7}$$

The resulting distance has the properties of a metric but is not guaranteed to be positive definite and possesses discontinuous derivatives. These problems can be avoided using the **Regularized entropy match kernel** (**REMatch kernel**). The **REMatch kernel** states the best match problem in an alternative form

$$\hat{K}(A,B) = \max_{P\in U(N,N)} \sum_{ij} C_{ij}(A,B)P_{ij} \tag{7.8}$$

where $P \in U(N,N)$ are the set of $N \times N$ matrices, where all the elements of each row and column sum up to $1/N$ meaning $\sum_i P_{ij} = \sum_j P_{ij} = \frac{1}{N}$. In practice a slightly different problem [9]

$$K^\gamma(A,B) = (Tr)\mathbf{P}^\gamma C(A,B) \tag{7.9}$$

$$\underset{P\in U(N,N)}{\mathbf{argmin}} \sum_{ij} P_{ij}(1 - C_{ij} + \gamma \mathbf{ln} P_{ij}) \tag{7.10}$$

is solved due to efficiency. K^γ can be solved using the sinkhorn algortihm [9]. It can be shown that for $\gamma \to 0$ the REMatch kernel goes to $K^\gamma(A,B) \to \hat{K}(A,B)$ and for $\gamma \to \infty$ the REMatch kernel goes to $K^\gamma(A,B) \to \overline{K}(A,B)$.

Part II.

Results

8. Introduction

8.1. Problem

Examining NPT simulations with Machine Learning Potentials is something that has not received a lot of attention yet. It is known that these simulations often times fail. A common observation is that the volume of the system keeps increasing as soon as the simulation is started. The expansion of the simulation box never stops until the simulation is entirely unphysical. It is as interesting as it would be useful to understand why this process is happening and what the cause is. The name of this thesis "Identifying Failures in Machine Learning Potentials" refers to this problem and understanding it is the first step.

8.2. Investigation Procedure

The idea is now to investigate different effects that could be responsible for the failure of ML NPT simulations. The first step consists of examining the influence of long range interactions. Then a look is taken at how accurate the ML potential needs to be. Related to that, it is examined what the influence of training the potential on different physical quantities has. Then we examine what conditions the training data needs to satisfy.

In order to investigate the different effects, we need a test system. The test system should avoid unnecessary physical difficulties. This will make it easy to understand what exactly went right or wrong. For this reason molten NaCl is chosen. It is a system that is well studied in simulations and experiments and there are accurate force fields available. This will make comparisons of the ML simulations to non ML simulations and experiments easy and clear.

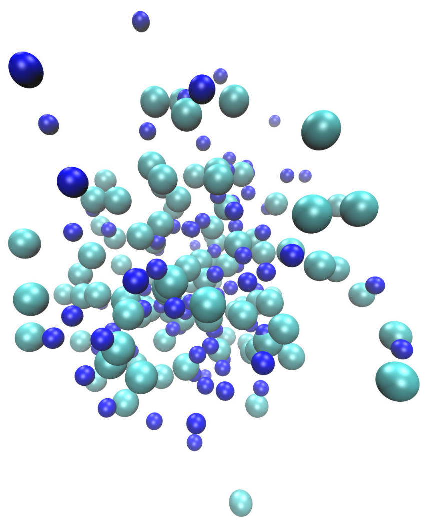


Figure 8.1.: Snapshot of a NaCl simulation using Lammps.

The goal of this thesis is to promote the understanding of ML potentials and simulations. It is therefore not necessary to use DFT data to construct very accurate potentials. For the most part fitting the ML model to classical MD data will be sufficient. This makes the generation of training data a lot faster and more adaptable. A dataset of NaCl configurations is created by running a Lammps [24] classical MD simulation using the Born-Mayer-Huggins-Tosi-Fumi potential. The training data is created by sampling the dataset uniformly in energy.

Gaussian Process Regression is chosen as the ML method and the Gaussian Approximation Potential (GAP) implementation is used [3]. GAP is a codebase that is able to approximate Potential Energy Surfaces (PES) from training configurations. The resulting potential is saved and is used as a plugin for Lammps.

9. Long Range Interactions

It is well known that long range interactions can create problems for ML potentials [32]. Every descriptor takes only particle positions into account that are within the cutoff radius the so called local environment. Particles outside of the cutoff radius therefore do not play a role for the prediction of the energy, forces and pressure by the ML algorithm. As a result, long range interactions are not explicitly considered in the ML potential. But long range interactions play an important role in DFT simulations. Different methods have been proposed to solve this problem. One way is to separate the many body interactions and the electrostatic contribution, which are always long range. Another approach is to include long range functionals in addition to the short range ML prediction [32]. In regards to running NPT simulations, these missing long range contribution could potentially cause significant errors in the prediction of the pressure, which therefore could lead to an incorrect size of the simulation box.

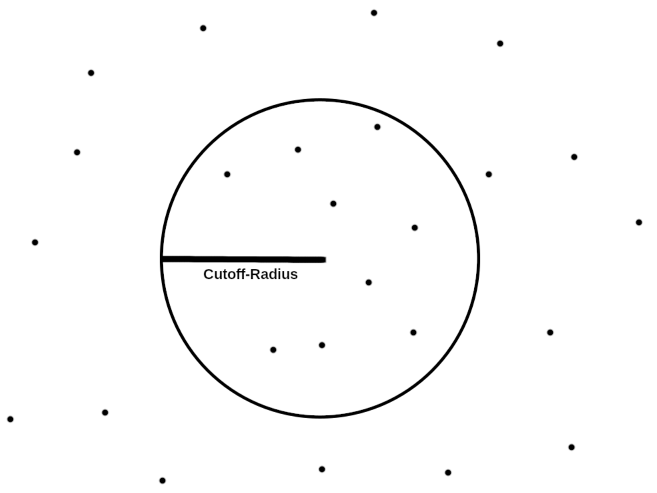


Figure 9.1.: Depiction of the cutoff radius.

The effects of long range interactions on ML potentials are investigated by setting up two systems with different particle interactions. The workflow is depicted in Figure 9.2. Both systems consist of 100 Na and 100 Cl atoms interacting via the Born-Mayer-Huggins-Tosi-Fumi (BMHTF) potential [29]. The BMHTF is an empirical potential for alkali halides (section 4.1). On top of that either a Coulomb potential with unlimited range or a Wolf potential with a cutoff range is added, as described in section 4.2. Both systems are simulated using Classical MD and the resulting trajectories are used as training data for the GAP ML method. The cutoff radius of the descriptor is set to the cutoff radius of the Wolf potential. Particles that are outside this radius therefore do not interact with the central particle and thus do not need to be considered when predicting the force on the central particle. The comparison of the predicted energy values with the MD energy values on a validation data set is depicted in figure 9.3. The ML prediction for the Wolf potential is very precise, while the prediction for the Coulomb potential has significant fluctuations. It looks like the long range interactions introduce noise.

By running an NPT simulation using the Wolf system, one can test if the noise is causing problems to the point where a NPT simulation fails. The Wolf system also fails in the same way with an always expanding box size. Getting rid of the long range interactions therefore does not reliably solve the NPT problem. This is the expected result when considering that an expansion of the box is the result of pressure that is systematically too high. Noise sometimes underpredicts and sometimes overpredicts the pressure. The mean influence of the long range interactions is learned correctly by the ML potential.

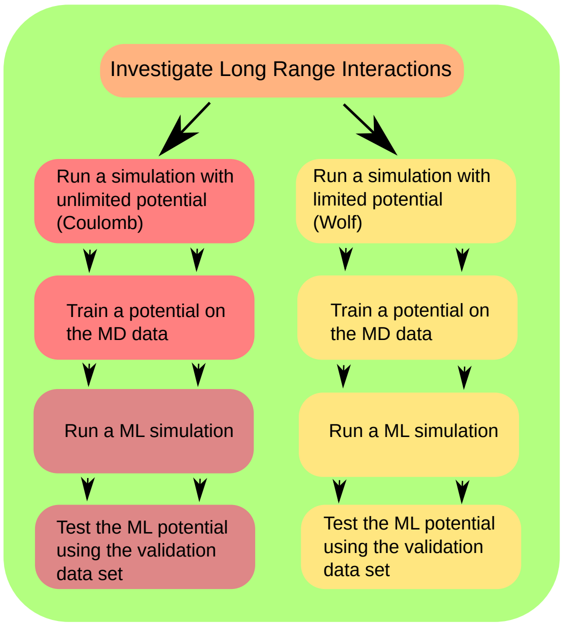
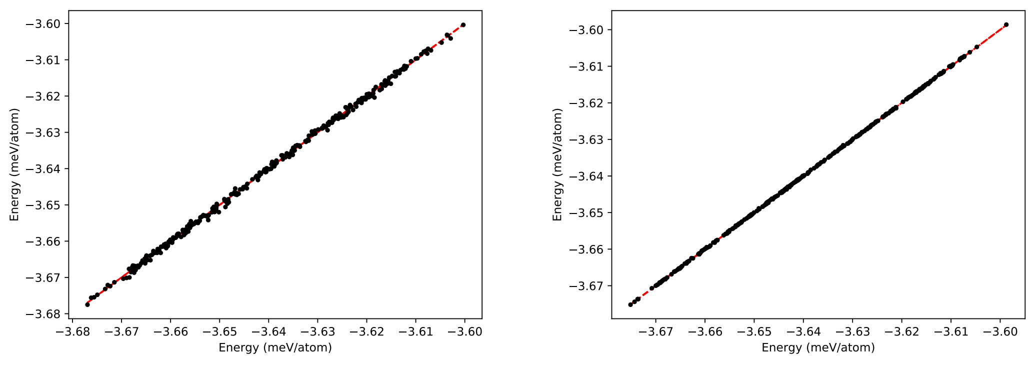

Investigate Long Range Interactions
Run a simulation with unlimited potential (Coulomb)
Run a simulation with limited potential (Wolf)
Train a potential on the MD data
Train a potential on the MD data
Run a ML simulation
Run a ML simulation
Test the ML potential using the validation data set
Test the ML potential using the validation data set

Figure 9.2.: Workflow for investigating the influence of long range interactions.



(a) Coulomb potential (b) Wolf potential

Figure 9.3.: Comparison of the simulated MD energy to the predicted ML energy for the Wolf and the Coulomb potential.

10. Accuracy of the Fitting Procedure

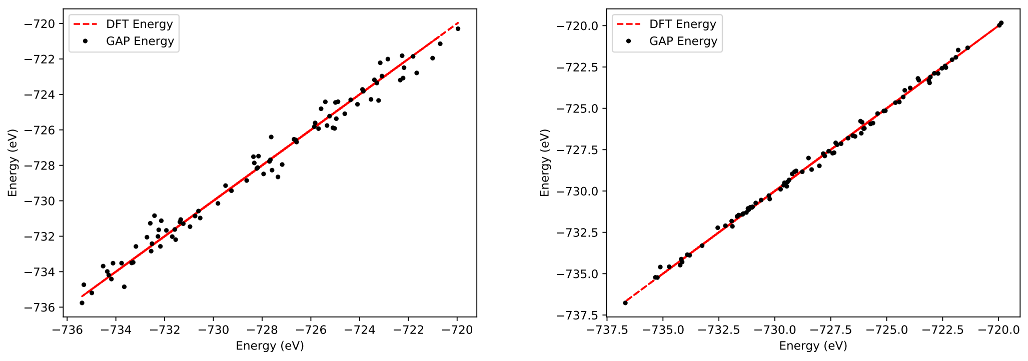
It is possible that the ML potential is just generally too inaccurate, making it very likely to fail. In order to rule out this possibility we tried taking steps to create a less noisy potential. For the sake of this discussion, a noisy potential is defined as a ML potential whose predictions significantly differ from the validation data. Because it is easier, we first examine how to create a less noisy NVT potential and then the NPT case is studied.

10.1. Investigating Descriptors

GAP allows the use of several descriptors at once. A series of tests is conducted which compare the predictions of potentials that are trained using only soap and potentials that are trained using soap and distance_2b. One of these comparisons can be seen in Figure 10.1. The parameters for both potentials are exactly the same except for the descriptor values. The potential that was trained on soap and distance_2b is significantly more accurate than the one only trained on soap. This is representative for all the comparisons made with varying cutoff radii.

10.2. Cutoff variation

In order to understand how an NPT capable ML potential can be created, first we need to investigate the simpler NVT case. For that we need to understand what information the potential has to be trained on. A test is conducted where the GAP potential is trained on energies and forces for different values of the cutoff parameter. The result is depicted in Figure 10.4 . The predicted virial and therefore also pressure for some cutoff values is systematically too high, while it is systematically too low for others. The information encoded in the energy and force values is therefore not enough to predict the virial pressure reliably for the NaCl system. Since NaCl is a particularly simple system it is to be expected that training on energies and forces only is generally not enough to predict an accurate pressure in an NVT simulation. It is therefore necessary to train on data that contains information about the pressure directly. For GAP potentials it is subsequently necessary to train on virials.



(a) Only SOAP (b) SOAP and distance_2b

Figure 10.1.: Comparing the energy predictions of a potential that was trained using SOAP and a potential that was trained using SOAP and distance_2b.

10.2.1. Structure and Dynamics

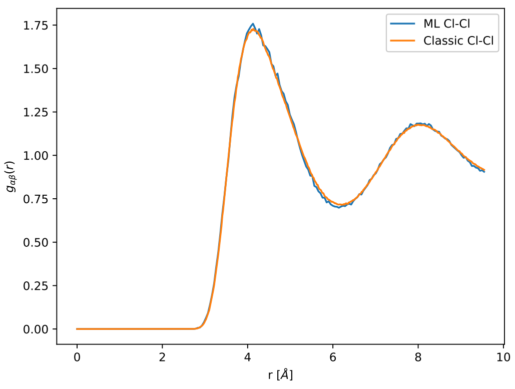
GAP Potentials that are trained on energies and forces are not sufficient to predict the pressure accurately in an NVT simulation. But they are capable of predicting the structure and the diffusion accurately. The rdfs of the training simulation run and the respective ML simulation run are depicted in figure 10.2. The rdfs look very similar. The pressure is computed by summing the kinetic term, that depends on the temperature

$$P = \frac{Nk_BT}{V} + \frac{\sum_i \vec{r}_i \cdot \vec{F}_i}{3V} \tag{10.1}$$

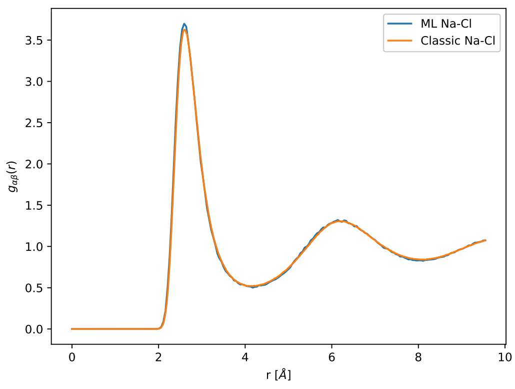
and the virial term that depends on the forces and positions. The mean of the kinetic term stays the same throughout the simulation, because the thermostat is active. The positions of the particles should be very similar in both simulation runs, because the rdfs look very similar. The forces vary a bit depending on how good the potential is but the mean should be the same. As a conclusion one can say that seemingly small differences in the rdfs and forces can lead to significantly different pressures. That also means that just looking at rdfs and diffusion coefficients is not enough to asses whether a potential gets the pressure correctly and therefore definitely not enough to assess whether it is NPT capable.

10.3. Training only on virials

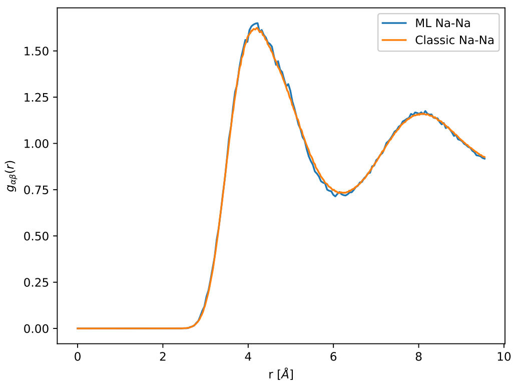
The time it takes to create a GAP potential depends on the amount of descriptors characterizing a configuration. For the calculation of the GP the covariance matrix K needs to be calculated and inverted. This process requires a lot of time and computer



(a) Cl-Cl

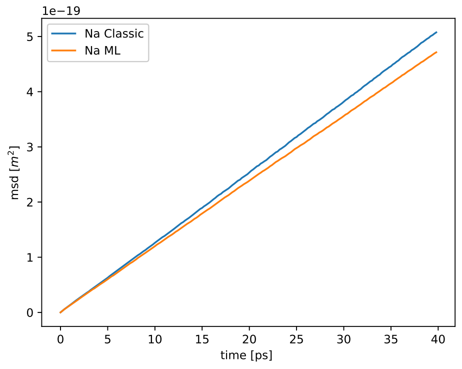


(b) Na-Cl

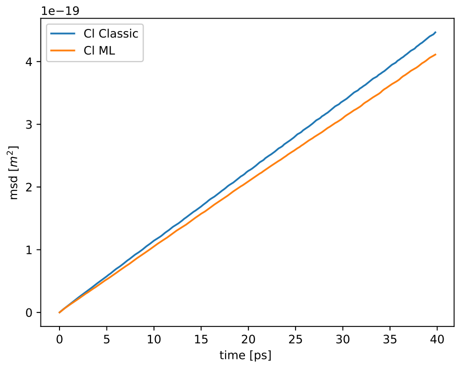


(c) Na-Na

Figure 10.2.: Comparison of the rdfs of a classical MD simulation and a ML simulation that was trained on energies and forces with a cutoff of 5.



(a) Na



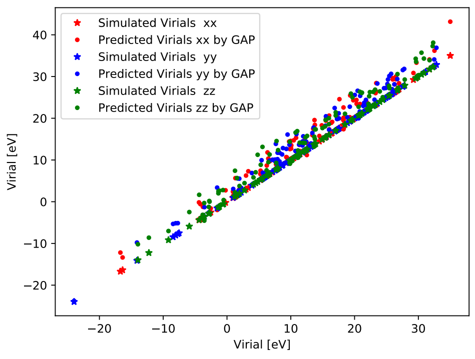
(b) Cl

Figure 10.3.: Comparing the mean squared displacement of a classical simulation and a ML simulation that was trained on energy and forces with a cutoff of 10. The resulting Diffusion coefficients are Na ML $D = 1.239 \times 10^{-8}\,\mathrm{m}^2/\mathrm{s}$, Na classic $D = 1.23 \times 10^{-8}\,\mathrm{m}^2/\mathrm{s}$, Cl ML $D = 1.115 \times 10^{-8}\,\mathrm{m}^2/\mathrm{s}$, Cl classic $D = 1.159 \times 10^{-8}\,\mathrm{m}^2/\mathrm{s}$.

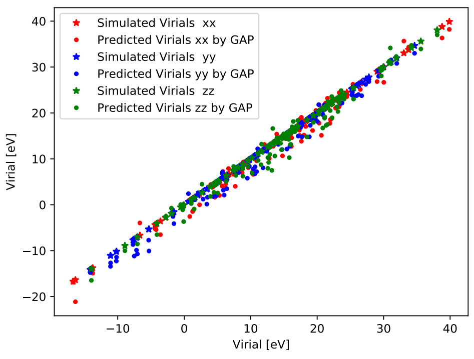
memory. It is common to run out of memory when trying to create a GAP potential on a system with 200 atoms with more than 500 training configurations on a computer with 126 GB of memory. It would therefore be valuable to know how many descriptors are needed in order to create a sufficiently good potential. For this test series the potential is only trained on virials, where

$$\text{virial} = V \cdot \underline{\underline{P}} \tag{10.2}$$

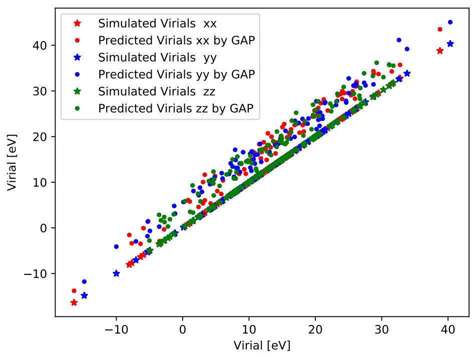
and $\underline{\underline{P}}$ is the pressure tensor. The results of the training are depicted in figure 10.5. The virials are predicted accurately by the GAP potential. The forces are noisy. To see if that has an effect on the dynamics, a four step process is conducted. First the training data is generated using a classical MD simulation. Then the potential is trained using the GAP framework. Subsequently a simulation is done using the ML potential. Finally a rerun is performed, that uses the trajectory calculated by the ML run, but the forces, energies and pressures are calculated with the classical MD potential. This way it can be tested, if the inaccuracy of the forces lead to new unknown configurations, that the ML potential can not predict. The average pressure of the different runs is depicted in table 10.1. The pressure of the training run is very close to the pressure of the ML run. When comparing that to the pressure from the rerun, it becomes obvious that the ML potential is very inaccurate and that the configurations explored in the ML run are very different from the configurations in the training run. This suggests that the inaccurate forces really do lead to new areas of the configuration space. It looks like training only on virials is not sufficient to get good pressure predictions. Predictions for the energy and forces are expectedly even worse.



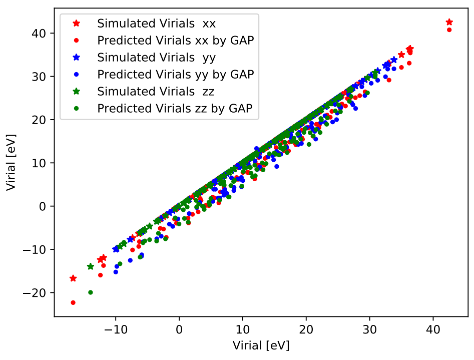
(a) Cutoff of 5



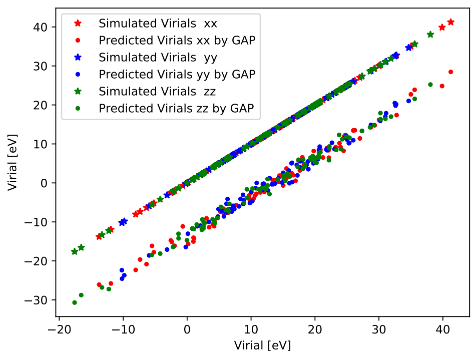
(b) Cutoff of 6



(c) Cutoff of 7

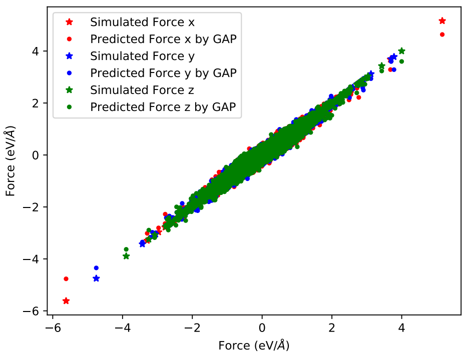


(d) Cutoff of 9

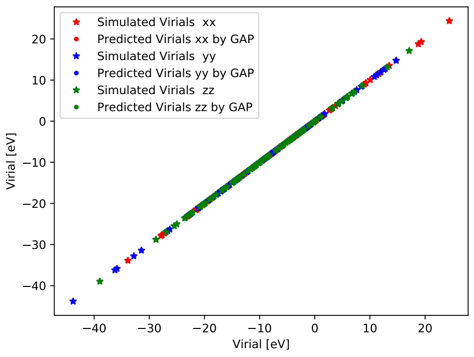


(e) Cutoff of 10

Figure 10.4.: Comparisons of the virial predictions of potentials that are trained on energy and forces with a varying cutoff parameter.



(a) Forces



(b) Virials

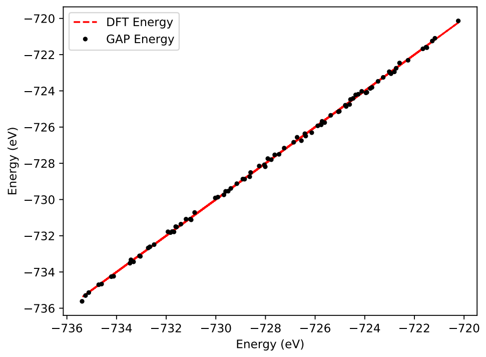
Figure 10.5.: Results of a potential that is only trained on virials. Comparison of the predicted virials and forces with the simulated virials and forces.

pressure training run	pressure ML run	pressure rerun
2610 atm.	2622 atm.	3719 atm.

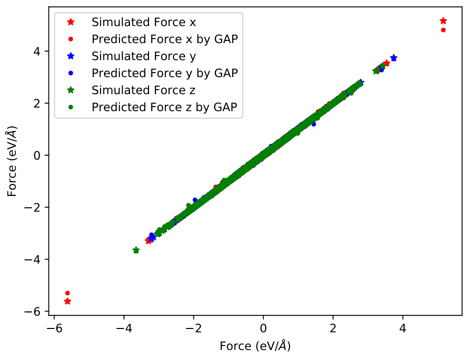
Table 10.1.: average pressure of the training run, ML run and rerun.

10.4. Creating a good NVT potential

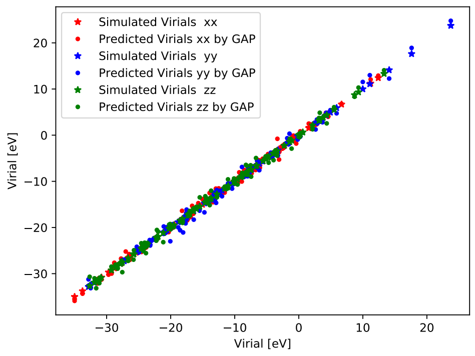
The next step is trying to create a NVT potential that can predict energies, forces and pressures accurately. Potentials that are only trained on one or two observables can not reliably reproduce them as shown in previous sections. We test how training on all three observables effects the ML potential. The comparisons between simulated properties from the training run with the predicted properties from the ML potential are shown in Figure 10.6. The energies, forces and virials are all predicted accurately on the validation set. The virials are not predicted as accurately as the potential, that was only trained on virials (Figure 10.5). But in return the forces and energies are predicted a lot more accurately. The same procedure as in the previous Section 10.3 is conducted. First, a training run is used to generate data and then the ML run and rerun are simulated. The average pressures are depicted in table 10.2. The pressure in the training run and the ML run are again similar, the average pressure of the rerun differs a bit (17 % deviation) but is significantly closer than when trained only on virials (42 % deviation). This suggests that the ML simulation still explores new configurations that were not in the training data. The ML potential predicts the pressures for these configurations inaccurately. The fact that the pressure deviation is smaller compared to the potential that was only trained on virials suggests that this effect is reduced by being able to predict the forces more accurately. The potential could probably be further improved by having a data selection



(a) Energy



(b) Forces



(c) Virials

Figure 10.6.: Comparison of the predicted observables with the simulated ones. The ML potential is trained on energies, forces and virials.

method that includes some of these unknown configurations in the training data.

pressure training run	pressure ML run	pressure rerun
2610 atm.	2556 ± 54 atm.	3060 ± 55 atm.

Table 10.2.: Average pressure of the training run, ML run and rerun. The potential is trained on energies, forces and virials.

10.5. Running NPT simulations

Now that it has been established how to create a potential that can predict the structure, dynamics, energies, forces and most importantly pressures sufficiently accurate; a closer

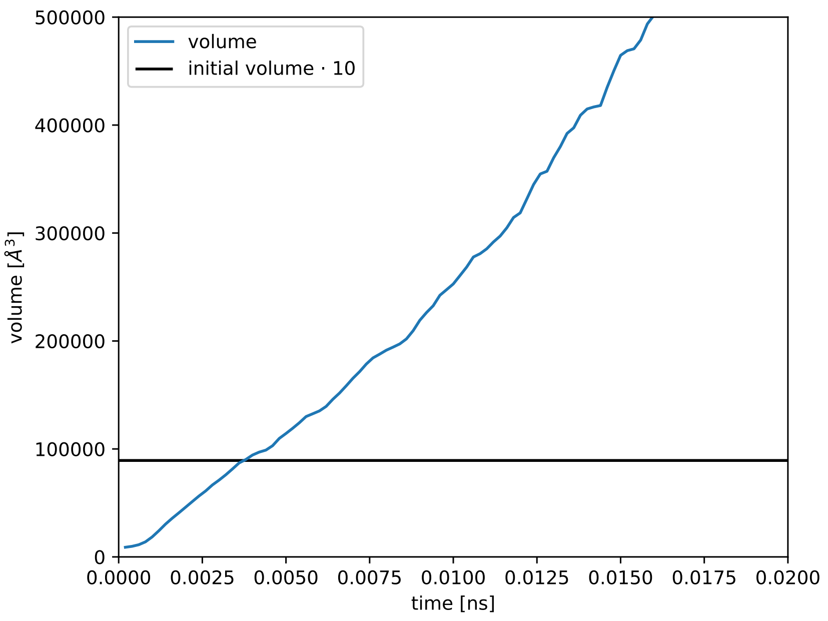


Figure 10.7.: Volume of the simulation box in dependence of time. The potential worked well for a NVT simulation, but for a NPT simulation it fails.

look can be taken at creating a NPT capable potential. The next step is to try whether the best NVT potential that was created for the previous tests is capable of running an NPT simulation. The time dependent volume of an NPT simulation is depicted in figure 10.7. From the start the volume increases and quickly surpasses ten times its initial value (0.0035 ns). At this point the simulation is surely completely unphysical and could be stopped. This suggests that there still is a systematic problem.

10.6. Density Variation

ML potentials generally work best when the configurations in the ML run are similar to the training configurations. The question we need to answer is how can the training data better match the configurations that occur during the ML run? So far all the training data has been generated by a single NVT simulation. In an NPT simulation the box size changes constantly. It seems reasonable that the training data should reflect this fact somehow. The simplest approach is to produce the training data using an NPT simulation. For small particle numbers NPT simulations have huge fluctuations, which causes problems for ML. This is why five NVT simulations with different box sizes are used to generate the training data. The box sizes are chosen in a way that the system wants to contract for the big box sizes and the system wants to expand for small box sizes. The ML algorithm should therefore be able to learn that there is an equilibrium and not continuously expand any more.

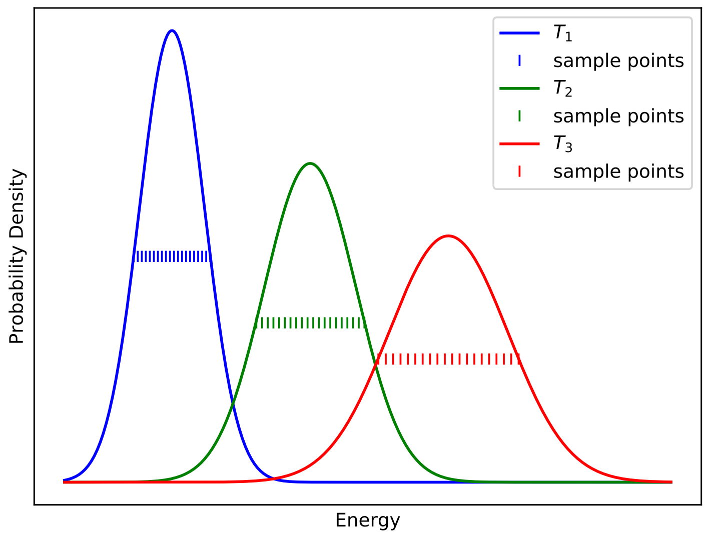


Figure 10.8.: Energy probability distribution in dependence of Temperature according to equation 10.4, where $T_1 < T_2 < T_3$.

10.6.1. Temperature Variation

Additionally to the density variation, the temperature can also be varied. The idea is to sample a bigger part of the phase space therefore making the ML potential more robust. The energy probability distribution in the canonical ensemble is given by [27]

$$p(E) = \frac{\Omega(E)\exp(-\beta E)}{Z} \tag{10.3}$$

where E is the energy, $\beta = \frac{1}{k_B T}$ and Z is the partition function. This expression can be further approximated as [27]

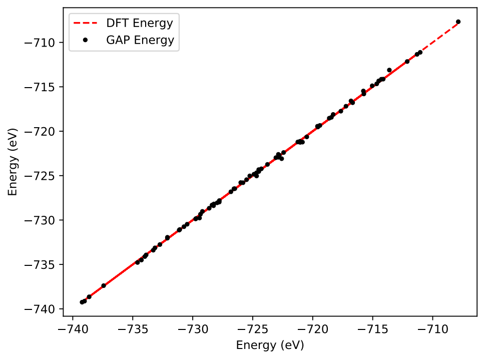
$$p(E) \approx \frac{1}{\sqrt{2\pi k_B T^2 C}} \exp\left(\frac{-(E - \langle H \rangle)^2}{2k_B T^2 C}\right) \tag{10.4}$$

where T is the temperature, $\langle H \rangle$ is the expectation value of the energy and C is the heat capacity. The distributions for different Temperatures are depicted in figure 10.8.

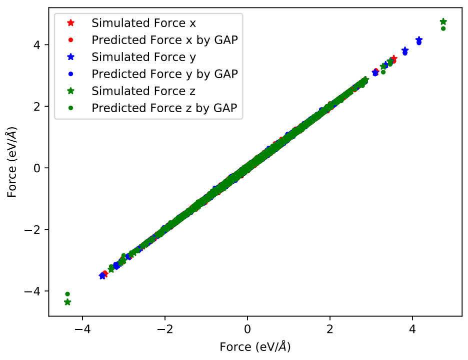
10.6.2. Different training approaches

We conduct different training approaches and compare them. For the first one the training data is generated by five NVT simulations with box volumes of from $6000\,\mathring{A}^3$ to $9000\,\mathring{A}^3$. The equilibration box volume of the classical MD NPT simulation is $7771\,\mathring{A}^3$.

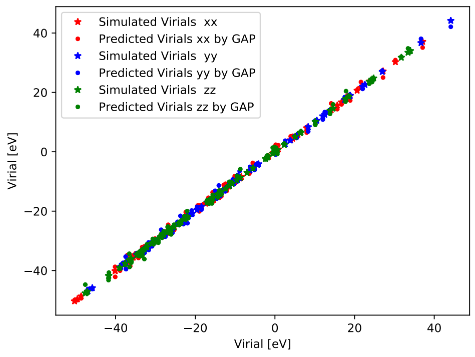
The temperature of training runs match the temperature of the ML run at 1400 K. The time dependent volume of this ML approach is depicted and compared to the classical potentials in figure 10.10. The equilibration volume of the ML simulation is matching the value of the classical potential run, the deviation of the 200 particle runs is 1.2% and of the 1000 particle simulations 2.8%. The ML simulations therefore are successful and do not continuously expand.



(a) Energy



(b) Forces



(c) Virials

Figure 10.9.: Comparison of the predicted observables with the simulated ones. The ML potential is trained on energies, forces and virials and the trainings data set is made out of configurations of 5 different nvt simulations at different box sizes. This potential was used to run the successful NPT simulation depicted in figure 10.10.

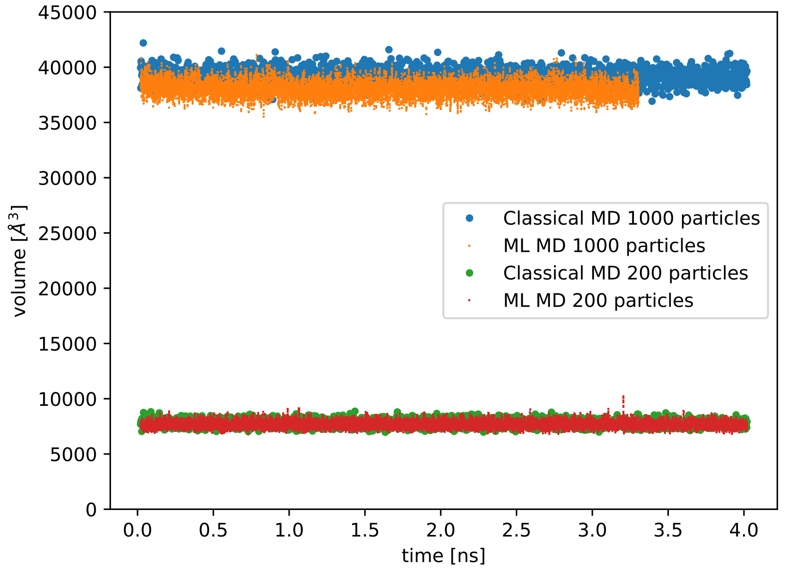


Figure 10.10.: Time dependent volume of the classical potential runs and ML potential runs. The training data for the ML potential was created using five NVT simulations at different box sizes.

11. Encountering Problematic Configurations

In the last chapter it was demonstrated, how a ML potential can be trained that runs a NPT simulation successfully. The chapter started by stating the problem, that NPT simulations expand more and more and apparently fail from the beginning. This very much looks like there is a systemic problem with the training procedure. We then went on to fix these systemic problems. But there is no guarantee that applying the new adapted training procedure will lead to a working ML potential. It simply demonstrated that it can work. There are other different problems that can occur. When training and running different ML potentials it can happen that the simulation starts out looking stable, but at some point it suddenly goes wrong. A case of this happening is depicted in Figure 11.1. The figure depicts the volume in dependence of time. In the beginning the volume fluctuates around an average value as expected. After simulating about 22000 timesteps the volume suddenly strongly increases. The potential used to run the simulation has accurate predictions on the validation data set, as it can be seen in 11.2. It would be useful to understand what happens during and shortly before the system explodes.

In contrast to the problems before, it seems that this is not a systemic problem. Rather it looks like the physics is modeled accurately until the system encounters some configurations, where the ML potential does not know what to do. Then the system runs for a little longer and expands quickly. It would be really helpful to know which the first configuration is, that causes problems. Then this configuration could be examined and a potential could be built which understands it. This could be done for example by adding the problematic configuration to the training data. This raises the question on how to find the first problematic configuration. A way to find this configuration could be to look at the physically measurable quantities and find out if any of them predict the simulation failure. The volume in dependence of time shows approximately where the simulation goes wrong. But due to the strong fluctuations it is fairly difficult to say which configuration exactly is responsible for the explosion.

Another approach is to look at the temperature during the simulation. The temperature in dependence of time is depicted in figure 11.3. The breaking step is the first configuration, that exceeds 1.5 times the equilibrium volume. At that point the simulation does not reproduce real physics any more and is way out of outside the range of the natural fluctuations which are about 10%. The breaking step is indicated by a black vertical line. A running average of the temperature is indicated by the orange line. The behavior of the temperature does not show any interesting signs before the system explodes. This

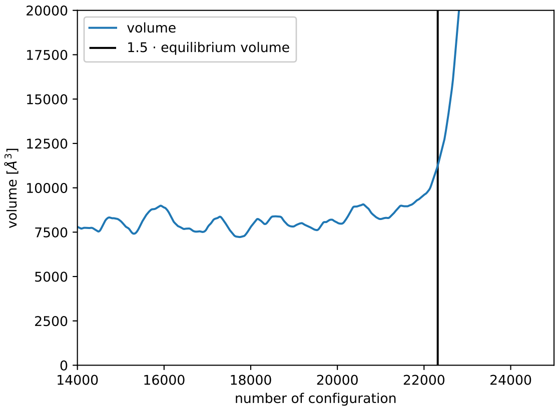


Figure 11.1.: Time dependent volume of a ML potential. The volume fluctuates around the expected value until around 22000 configurations. Then the volume rapidly increases making the simulation unphysical.

may be due to the thermostat which tries to keep the temperature roughly constant. There is a spike after the system explodes, which might be due to the sudden increase in box volume. This spike is not of interest, since it happens after the simulation failure and therefore can not be a predictor. Closely connected to the temperature are the velocities of the particles. Lammps calculates the temperatures using

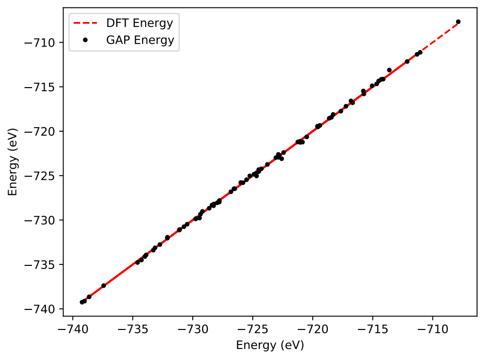
$$KE = \frac{3}{2} N k_B T \tag{11.1}$$

where N is the number of particles, k_B is the Boltzmann constant, T is the temperature and

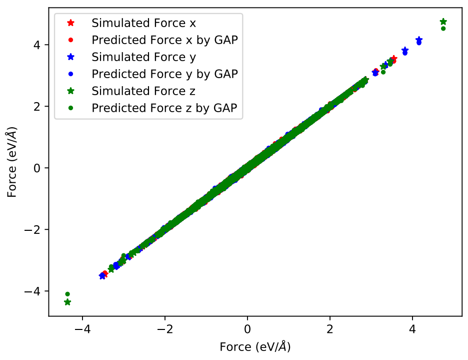
$$KE = \sum_{i=1}^{N} \frac{1}{2} m v^2 \tag{11.2}$$

is the kinetic energy of the particles. The information about the average velocities is therefore contained in the temperature. It could still be that there are a few particles that show interesting behavior. For example it could be that case that some particles reach an unusually high velocities, because the potential predicted some very high forces incorrectly. To examine this, a series of velocity histograms was created. They compare the velocity distributions before and after the breaking step at different times. One of those comparisons is shown in figure 11.3 b). The figure is representative of all the comparisons made. The distributions have a Gaussian shape, which would be expected for a correctly running NPT ensemble, that reproduces a Maxwell-Boltzmann distribution. The distributions look very similar and have no unexpected statistical outliers, that could indicate a simulation failure.

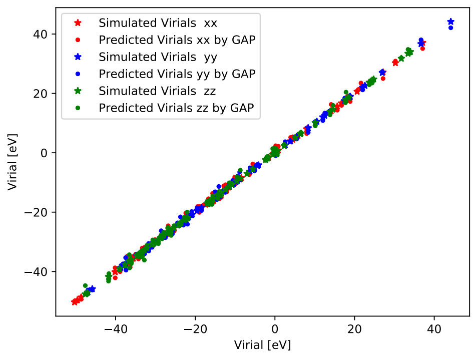
The next observable that is examined are the particle forces. A graph of the average



(a) Energy



(b) Forces



(c) Virials

Figure 11.2.: Comparison of the predicted observables with the simulated ones. The ML potential is trained on energies, forces and virials and the trainings data set is made out of configurations of 5 different nvt simulations at different box sizes. This potential resulted in an unsuccessful NPT simulation as depicted in figure 11.1.

force magnitude

$$AFM = \frac{1}{N_{part}} \sum_{i=1}^{N_{part}} \sqrt{\vec{F_i} \cdot \vec{F_i}} \tag{11.3}$$

in dependence of time is depicted in figure 11.4 a). The average force magnitude fluctuates around a constant average until the simulation breaks. After the breaking step the average force magnitude increases significantly. This is again not very useful when looking for a predictor of simulation failure. It would be reasonable to guess, that the simulation fails because the ML potential predicts some forces incorrectly. There could be one or a few particles, that experience a very high force. This would not necessarily change the average a lot and therefore not be visible in figure 11.4 a). That is why a series of histograms is created that compare the force distributions before and after the breaking step. One of them is depicted in figure 11.4 b). The same Gaussian function is plotted in both histograms for comparison. The histograms look very similar and there are no outliers, that can be detected.

It is also possible that a sudden increase in pressure, causes the simulation to explode. For this reason the time series for the pressure is examined as depicted in figure 11.5 a). The pressure

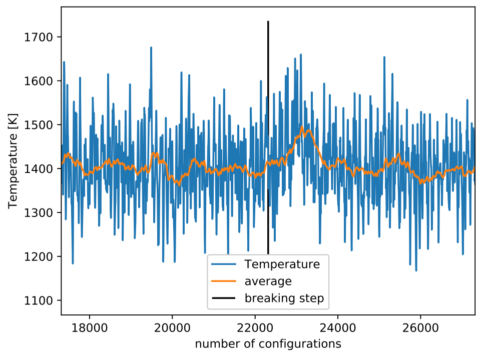
$$P = \frac{Nk_BT}{V} + \sum_{i=1}^{N_{part}} \frac{\vec{r_i} \cdot \vec{F_i}}{3V} \tag{11.4}$$

fluctuates around zero until the simulation breaks. After the breaking step the pressure spikes and reaches a maximum. Then the pressure drops again. A similar pattern can be observed for the virial pressure depicted in figure 11.5 b). The virial pressure is the pressure minus the kinetic part

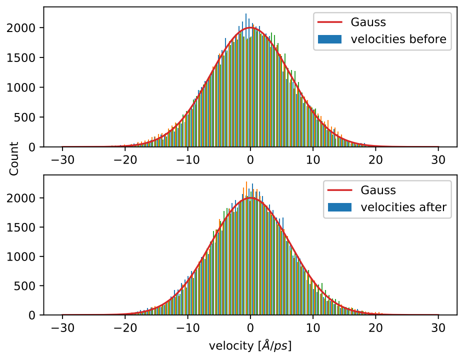
$$P_{virial} = \sum_{i=1}^{N_{part}} \frac{\vec{r_i} \cdot \vec{F_i}}{3V}. \tag{11.5}$$

The virial pressure is what is used, if it states that a ML model is trained on pressure. This is because the kinetic energy is not interesting and only creates noise when creating a potential.

As a summary there are no physical quantities found that can reliably predict simulation failure. Therefore physical quantities can not be used to identify the first configuration that the ML potential has a problem with. That is also why this problem needs to be approached differently.

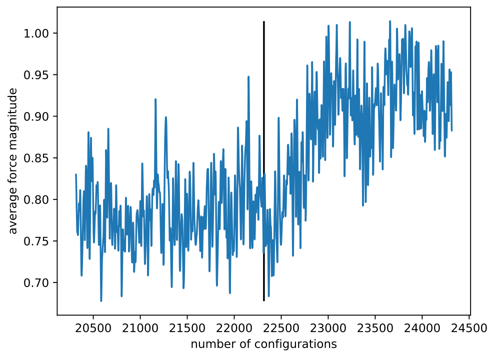


(a) Temperature

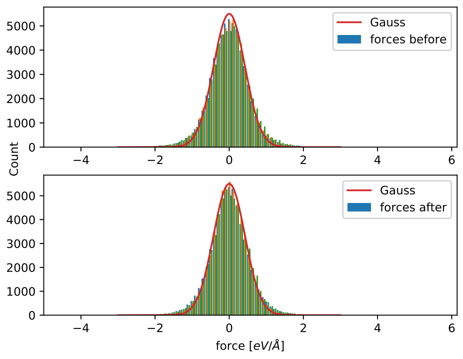


(b) Velocities

Figure 11.3.: (a) Temperature in dependence of time in a simulation that explodes. The point where the volume becomes bigger than 1.5· the equilibrium value is indicated by a vertical black line. (b) The distribution of velocities before and after the simulation explodes.

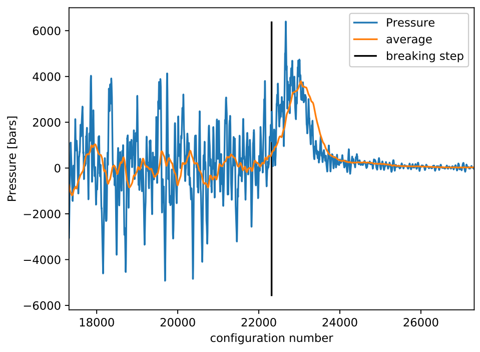


(a) Average force Magnitude

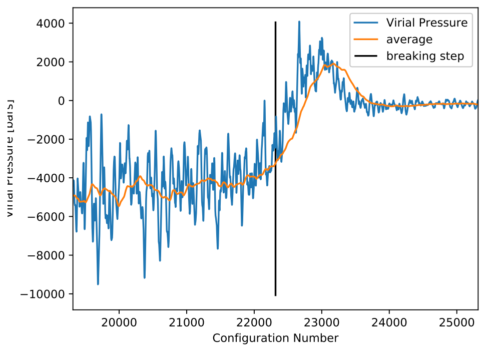


(b) Forces

Figure 11.4.: (a) Average force magnitude in dependence of time in a simulation that explodes. The point where the volume becomes bigger than 1.5· the equilibrium value is indicated by a vertical black line. (b) The distribution of forces before and after the simulation explodes.



(a) Pressure



(b) Virial Pressure

Figure 11.5.: Pressure (a) and virial pressure (b) in dependence of time. The point where the volume becomes bigger than 1.5· the equilibrium value is indicated by a vertical black line.

12. Investigation of Descriptor Space

Since simulating physical systems by solving Newton's equation of motions takes a lot less effort and time than solving the Schrödinger equation, the classical approach is used when it is possible. But using the classical approach comes at a sacrifice of accuracy. That is why most of the time ML potentials are trained on DFT data. The ML approach tries to interpolate the potential energy surface that is given by the sampled data points. The accuracy of the ML potential is therefore limited by the accuracy of the training data. This means that generating precise training data is of great importance to create reliable ML potentials. Avoiding expensive calculations while still having high accuracy is the idea behind hybrid models, that try to combine the advantages of both the classical and the ab initio approach. One method is the Single Point Calculator. It first samples the configuration space using a classical simulation and then picks specific configurations and runs DFT on them. In this way the amount of DFT computation time is greatly reduced, while hopefully keeping the accuracy. The quality of this approach depends on how well the classical configuration space sampling fits the DFT configuration space. It is also important how the configurations are chosen.

The are potential problems with the Single Point Calculator method. If the classical configuration space contains the ab initio configuration space, then a classical simulation run can produce all the configurations that can occur in an ab initio run. But there also will be configurations sampled that do not contribute to describing the relevant part of the configuration space and you generally want to avoid adding these to your training data. It is also possible that the ab initio configuration space contains the classical configuration space. In this case it is possible to sample the subspace using classical MD. Then some of the sampled configurations can be used as seeds for ab initio simulations in order to add the missing parts. It is also possible that configurations generated by the two approaches are generally different and have little overlap. In this case the there are some configurations in the ab initio data that can not be reproduced by the classical simulation. As a result a potential that is trained using the Single Point Calculator will likely differ from a potential that was trained on pure ab initio data. That is why we want to compare the configuration space of ab initio simulations with the configuration space of classical simulations.

In this work we use the REMatch kernel as described in chapter 7, to compare configurations. Now we want to find ways to use the REMatch kernel to characterize not only configurations but the whole configuration space.

12.1. Systems to Study

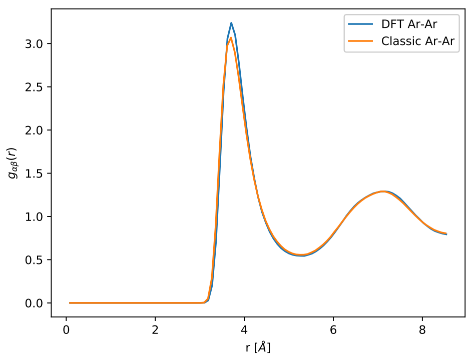
To study the configuration space, 3 systems are picked. Each of the systems is simulated one time using classical MD and one time using DFT. We use the same parameters like number of atoms, length of the timestep and initial configuration for both simulations in order to make the two runs comparable. The MD simulations are conducted with lammps [24] while the DFT simulations are run using CP2K [18]. The atom numbers are kept low, so that the DFT simulations are still feasible.

The first system is liquid Argon at 85 K, which is chosen because of its simplicity; it is monoatomic and the particle interactions are rather simple. It has been demonstrated that many properties of argon can be simulated accurately using a simple Lennard-Jones interaction model [1]. The Argon system is simulated using 108 atoms. The Lennard Jones parameters for the classical simulation are $\epsilon = 0.2381\,\mathrm{kcal/mol}$ and $\sigma = 3.405\,\text{Å}$. The potential has a cutoff of 13 Å. The box length is 17.042 Å resulting in a particle density of $\rho = 0.0218\,1/\text{Å}^3$ and the timestep is 10 fs for 50000 integration steps. The DFT simulation uses a refitted Perdew and Yang 86, and a Perdew, Burke and Ernzerhof functional. To account for dispersion a Revised Vydrov-van Voorhis nonlocal van der Waals density functional is used.

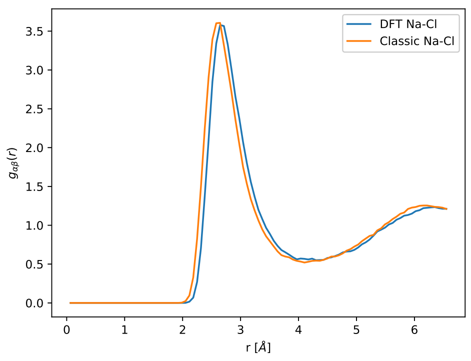
The second system is molten NaCl at 1400 K. There is plenty of experimental data on molten NaCl. Also it consists of two different species making the structure more complex than that of a monoatomic noble gas. Additionally the particle interaction is more complicated, than that of Argon. Interactions have been described for the most part using pairwise effective rigid ion or shell models giving reasonable results for structure and dynamics [14]. The Born-Mayer-Huggins-Tosi-Fumi (BMHTF) potential is used to model the particle interactions in the classical simulation. Additionally the PPPM algorithm is used to calculate the electrostatic interactions. The NaCl system uses 64 atoms and a timestep of 2 fs for 10000 integration steps and a total run time of 20 ps. The box length is 13.1 Å resulting in a particle density of $0.0284\,1/\text{Å}^3$. The DFT simulation also uses a refitted Perdew and Yang 86, and a Perdew, Burke and Ernzerhof and Revised Vydrov-van Voorhis nonlocal van der Waals density functional.

The third system is 1-Butyl-3-methylimidazolium tetrafluoroborate (BMIM BF_4). It belongs to the group of ionic liquids, which is a salt in liquid state [10]. Sometimes ionic liquids are also refered to as salts whose melting points is below 100°C. In contrast to other well extensively studied liquids like water, which consist of neutral molecules, ionic liquids consist of ions. BMIM BF_4 is a complex molecule with many H-H and C-H bonds, that have the ability to rotate and vibrate, making it more difficult to parameterize an emipirical potential. The system consists of 300 atoms and the box length is 14.69 Å resulting in a particle density of $0.094\,63\,1/\text{Å}^3$. The timestep is 1 fs for 50000 integration steps and a total run time of 50 ps. The DFT simulation uses the Becke 97 exchange correlation functional and the Grimme D3 method for the dispersion interaction.

Several radial distribution functions comparing the classical and the DFT data for the systems are depicted in figure 12.1. The radial distribution functions of Argon Ar-Ar



(a) Ar



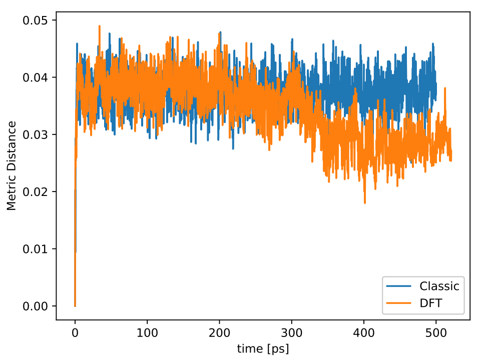
(b) NaCl

Figure 12.1.: Comparing the radial distribution function of the classical to the DFT simulation for a) the Argon system b) the sodium-chloride system.

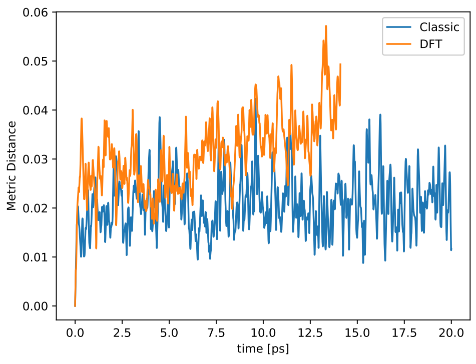
look pretty similar. The first peak of the DFT rdf at $r = 4\,\text{Å}$ is slightly higher than the peak of the classical rdf. The tails match up well. The NaCl rdfs also match well, but the classical Na-Cl rdf is slightly shifted towards smaller radii compared to the DFT Na-Cl rdf. This could indicate that the particle size is a little bit underestimated by the classical model.

12.2. Comparing Trajectory

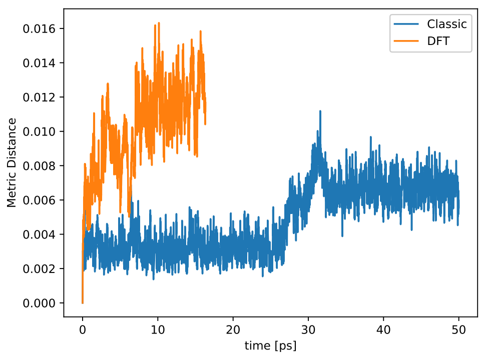
We compare the distance from the first configuration to all the other configurations of one simulation. So we calculate a distance time series of how far the simulation moves away from the first configuration. The time series are depicted in figure 12.2. We compare the time series of the classical simulations with the DFT simulations for every system. The classical series of Argon jumps from zero to a stable distance, that it fluctuates around. This might indicate that the system stays in one part of the configuration space for the whole simulation. The DFT series looks very similar in the beginning, but after around 300 ns the distance starts to fluctuate around a new smaller value. This change in the behavior might indicate that the simulation accesses a distinguishable part of the configuration space.
The time series of NaCl are depicted in figure 12.2 b). The classical series has a jump in the beginning and then fluctuates around a constant value. The relative fluctuations are bigger than the ones of Argon. In contrast the DFT series increases over time and is not stable. It might very well be the case that 14 ps are not enough to sample the DFT configuration space adequately and more distinguishable configurations would appear, if the simulation was run for longer. The BMIM BF_4 series are depicted in figure 12.2 c). The plots for the classical and DFT run look very different. The classical series fluctuates around a stable value in the beginning and after around 30 ps it starts to fluctuate around



(a) Ar



(b) NaCl



(c) BMIMBF4

Figure 12.2.: Comparing the metric distance to the first configuration in dependence of time for Ar, NaCl and BMIMBF4 using the REMatch kernel.

a new bigger distance. The distances of the DFT simulation are significantly bigger than the classical ones. It looks like only a part of the DFT configuration space is sampled and an even smaller part of the DFT space.

The fact that the DFT simulations look like they are too short to sample the the whole configurations space of realistic configurations indicates why it makes sense to apply something like a Single Point Calculator. The BMIM BF_4 DFT simulations were run for weeks. It therefore does not look like a feasible solution is to simply run the simulation for longer. The reason why the distances of the NaCl and BMIM BF_4 DFT simulations are bigger could very well be because the potentials are more complex and can capture behavior like rotational and vibrational degrees of freedom better. That could lead to more complex structures, which makes the configurations more distinguishable.

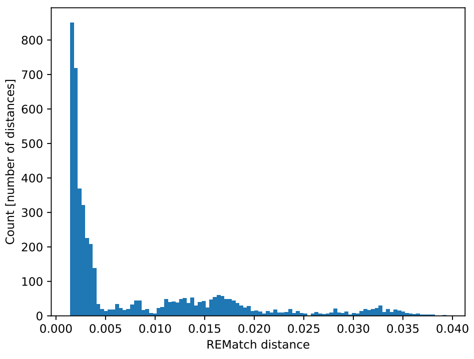
12.3. Comparing distributions

The comparisons of the time dependent distance series discussed in the last section miss information about the distances between any configurations that do not include the first one. Now we take a look at the distributions of the distances. Distributions include information about all the different combinations but do not include any information about the time dependence. We want to compare the differences of the configuration space between the classical and the ab initio runs. More specifically we want to find out whether there is an equivalent match for every configuration in the data sets. First we pick a configuration from the DFT data set and compare it to every configuration in the classical data set. Out of all these comparisons we select the one with the smallest distance, which corresponds to the configuration that is the most similar and add it to the distance histogram. We do this process for every configuration in the DFT data set. This way we have created a histogram with the distances of all the most similar configurations between the data sets.

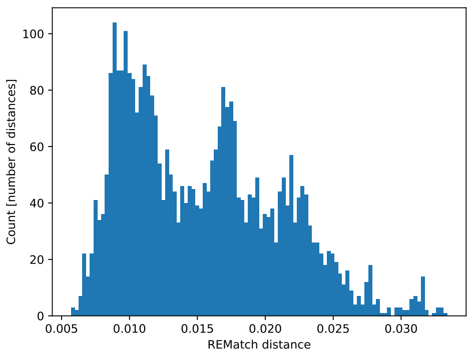
The histogram for Argon can be seen in figure 12.3 a). There is a high peak at small distances and low counts at further distances. This indicates that it was possible to find an equivalent match for almost all configurations.

The situation for NaCl is a bit more complicated and is depicted in 12.3 b). The histogram has a wide peak at smaller distances and also smaller peaks at bigger distances. In general the distances are much more evenly distributed than Argon. This indicates that there are matching but also not so well matching configurations.

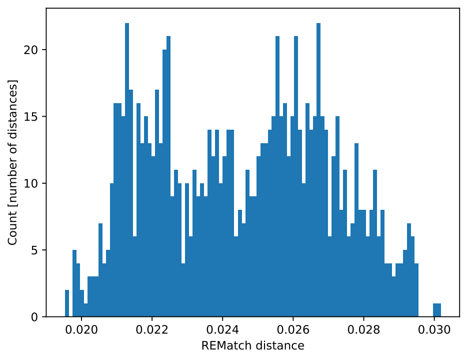
The histogram of BMIM BF_4 has no counts at smaller distances $d < 0.018$. There are a wide range of counts at medium and larger distances. This indicates that it is hard to find equivalent matches for $BMIMBF_4$. It is possible that the classical model can not reproduce the molecule vibrations and rotations accurately enough.



(a) Ar



(b) NaCl



(c) BMIMBF4

Figure 12.3.: Comparing the distribution of metric distances of Ar, NaCl and BMIMBF4 using the REMatch kernel. The distances are measured between configurations of the classical simulation and configurations of the DFT data.

13. Implementing an Active Learner

The previous two chapters dealt with identifying and characterizing 'bad configurations' i.e. configurations that can cause the simulation to fail. The aim of this chapter is to implement an Active Learner into MLSuite, that is flexible, scalable and adaptable. The goal is also to solve the problem of encountering bad configurations that lead to a failure of the simulation, by identifying and adding them to the training data, thus creating a more stable potential.

Active Learning is a subfield of Machine Learning where an algorithm can interact with a user to choose and label new data points. It aims to achieve an increased data efficiency, meaning better predictions with less training data. Active Learning is often employed when the labeling of the data is expensive, for example in speech recognition. Here the labeling is very time expensive and requires trained workers [28].

In this work active learning is applied to create improved Potential Energy Surfaces. The idea is to start with an initial training data set, which is used to train a potential. Subsequently this potential is used to run MD simulation steps. If the simulation encounters a configuration that is significantly different from the training data the simulation is stopped and this configuration is added to the training data, thus creating a new bigger training data set. Now a new improved potential is trained and the simulation is run again until it encounters another unknown configuration. This process can be repeated as often as you like.

13.1. MLSuite

MLSuite is a Software package. It is build as a pipeline for the whole ML process, from beginning to end. MLSuite is written in python and is easy to use. It aims to make every step of the ML process available with only one line of code. It uses DVC and ZnTrack to compute a computational graph that keeps track of the parameters and the stages of the ML process. In order to train a ML model you need data. MLSuite contains methods to generate data by interfacing with MD engines. The resulting data points and labels like energies, forces and pressures are stored in an ASE [20] database. It has been shown, that it makes a big difference which parts of a dataset are chosen as input for the training data. For this reason MLSuite has different data selection methods like uniform energetic, uniform force, random and uniform temporal. This allows the user to chose the most efficient data points to train on. MLSuite provides a variety of different models, for example there is GPR using GAP or NNs using FNET. Once the training is finished it can be tested by comparing predictions of the model with data points of the validation data set. The user has now the option to continue training using Active Learning methods or run the model directly.

13.2. Algorithm Structure

The Algorithm consists of five different parts that are separated into classes as it is depicted in Figure 13.1. The active learner class acts as the main class that calls methods from the other four classes. The Active Learner Class starts the program and initializes the ML Model Class, which uses an initial training data set to train a potential. Then the Simulator Class is started. It reads a parameter file containing information about atoms, positions and further simulation parameters and starts a simulation. After a fixed amount of simulation steps, the simulation pauses. The Active Learner Class reads out the current state of the simulation and hands the configuration over to the Configuration Compare Class. Subsequently, the Configuration Compare Class compares the new configuration to the the configurations in the training data set. If the new configuration is too similar to the training data the simulation continues and runs again for a fixed amount of steps. Again the last configuration is handed of to the Configuration Compare Class. If the new configuration is dissimilar enough, then the configuration is handed over to the Observable Calculator Class. The Observable Calculator Class assesses the energy and forces of the configuration and adds it to the training data. Then the ML Model Class is started and a new improved potential is trained. The Active Learner Class again starts the Simulator Class and the cycle is repeated.

In this work a simulator class is used based on the Lammps simulation engine. It reads a Lammps parameter file and sets up the simulation. The communication with MLSuite is done using Pylammps, a package that allows the interaction of the Lammps source code with python. Also Lammps is able to read GAP potentials using the QUIP package.
The Configuration Compare Class in this paper is based on the REMatch kernel. It calculates the similarity between the new configuration and every configuration in the current trainings data set. The result of a similarity calculation is a number that lies between 0 and 1. If the configurations are the same, then the result is 1. If the configurations are very dissimilar then the configuration will be significantly smaller than 1. The Rematch Compare Class compares all the similarity numbers to a by the user chosen critical similarity number. If there is no number higher than the critical similarity, the configuration will be added to the training set. This can be interpreted as the training set does not contain this configuration yet.
The Observable Calculator class used in this work is also implemented using the Lammps simulation engine. The class creates a second simulation box, that reads the current configuration and places the atoms accordingly. In contrast to the simulation box of the Simulator Class it uses a classical force field and no simulation steps are conducted. The simulation box is merely used to calculate the energy of the whole system, the forces on the particles and the pressure tensor. This information is added to an ASE database

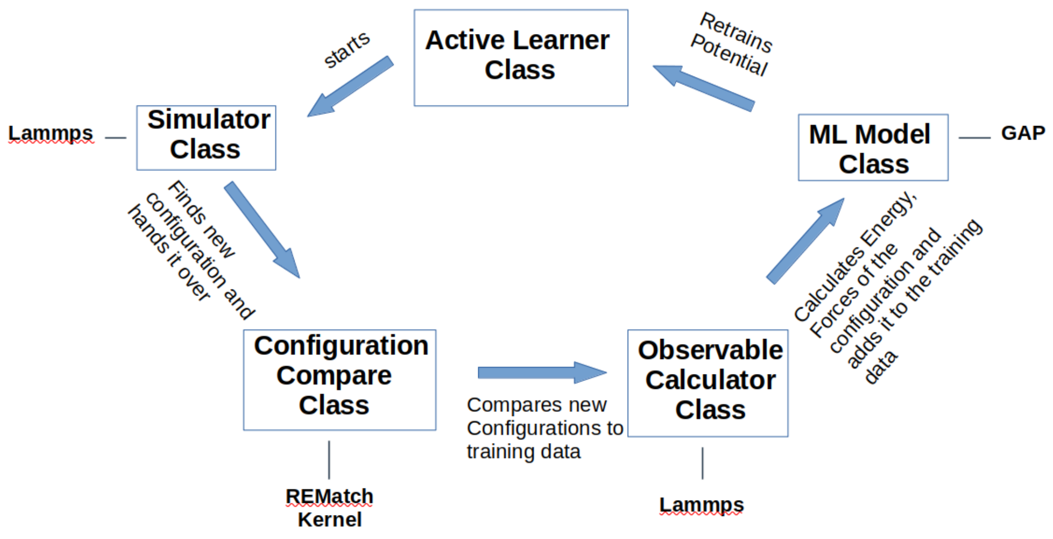

Active Learner Class
starts
Retrains Potential
Simulator Class
Lammps
ML Model Class
GAP
Finds new configuration and hands it over
Calculates Energy, Forces of the configuration and adds it to the training data
Configuration Compare Class
Compares new Configurations to training data
Observable Calculator Class
REMatch Kernel
Lammps

Figure 13.1.: Programming Structure of the Active Learner. It consists of 5 Classes. The Active Learner Class interacts with the four other Classes.

with the rest of the training data.
The ML Model class is based on GAP. It reads the training configurations from the database and trains a potential based on parameters chosen by the user.

Separating the code into different parts has the advantage, that each part can be replaced. For example the ML Model Class is based on GAP, but it is possible to write another class that is based on a Neural Network and plug that into the Active Learner making the algorithm very flexible by making use of polymorphism. In the future it would make a lot of sense to implement another Observable Calculator Class that is based on a DFT engine like CP2K. This would allow to create much more accurate active learning potentials, that are physically more interesting.

An example of how the Active Learner can be used is given with the Code example below. First mlsuite needs to be imported, then the four different classes need to be instantiated and finally handed over to the active Learner class.

```
import mlsuite as mls
from mlsuite.data.database import Database
from mlsuite.pes.ttv.ttv import TTV
from mlsuite.pes.models.gap import GAP
from mlsuite.pes.agents.simulator_active import LammpsSimulator
from mlsuite.pes.agents.lammps_observable_evaluator import
    LammpsObservableEvaluator
from mlsuite.pes.agents.delta_rematch import DeltaRematch

# project
project = mls.Project()
```

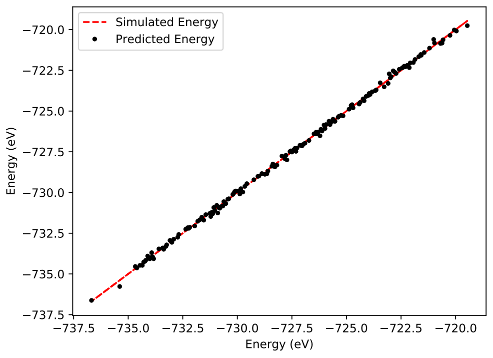
```
project.create_dvc_repository()
project.add_data(r"nacl_data.extxyz")
project.build_dataset(method="uniform_energy", n_configurations=200)
project.build_ttv(method="random", split=[0.5, 0.0, 0.5])

species_info = {'Na': {'lammps_type': 1, 'atomic_number': 11},
                'Cl': {'lammps_type': 2, 'atomic_number': 17}}

simulator = LammpsSimulator(parameter_file='in_start.lmp', restart_file='
    system_data_trained.extxyz', temperature=1400)

observable_evaluator = LammpsObservableEvaluator(potential_file='
    observable_evaluator_init.lmp')

gap = GAP(n=5, l=4, cutoff=10, use_energy=True, use_forces=False,
        use_virial=False)

compare_rematch = DeltaRematch(critical_similarity=999997, nmax=5, lmax=4,
    cutoff=10)

project.perform_active_learning(gap, simulator, observable_evaluator,
    compare_rematch, species_info,
        comparison_step=10, max_run_iterations=50000,
            max_new_configurations=30, initial_trainings_data=None)
```

13.3. Results

There are a couple of things, that can go wrong doing an active Learning scheme like this. It is possible that two particles get fairly close during the simulation process of the active learner. The ML potential might give the configuration a reasonable looking energy, but the observable calculator will give the configuration a very high or very low energy. Including these kinds of outliers in the training data can make the potential worse. These configuration would never happen in a realistic simulation of course. This problem mainly occurs when the active learning is started with a bad initial potential. This often happens when there are not enough configurations in the initial training data or when the potential is only trained on energies.

We test the active learner by training an NVT potential. The initial training set contains 160 configurations and to that 15 configurations are added. The potential is trained on energies and forces. The comparison of the predicted values on the validation data set is depicted in figure 13.2. The predictions are close to the simulated data.

A more interesting case is applying the active learner to a potential, that initially failed. Here the active learner is applied to a potential that caused the simulation to explode. The observables of the run agree with the expected value until a problematic configuration is encountered that the ML Potential does not know. By applying active learning, configurations like this can potentially be identified and included into the potential. The initial NPT potential is trained on 147 configuration using energy, forces and virials and


(a) Energy

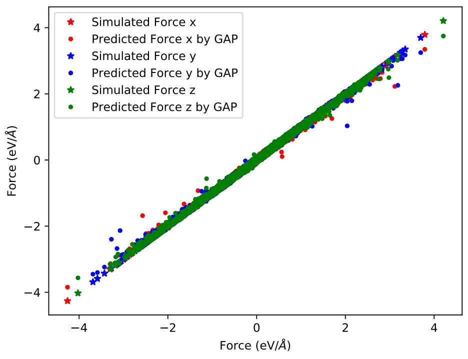
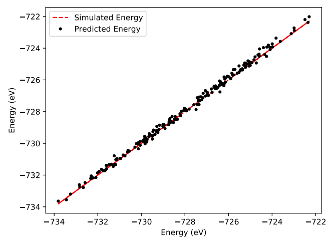
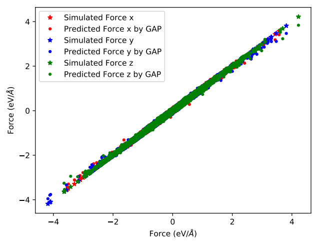

(b) Forces

Figure 13.2.: Comparing the predictions with the validation data of a NVT ML potential, that was trained using the active learner.


(a) Energy


(b) Forces

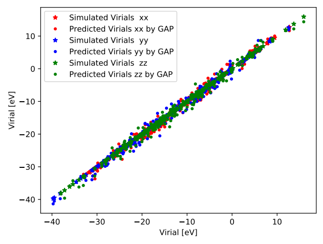
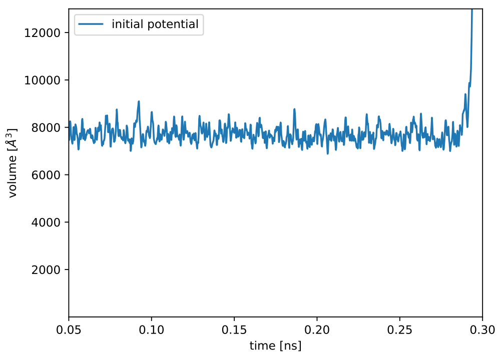

(c) Virial

Figure 13.3.: Comparing the predictions with the validation data of a NPT ML potential, that was trained using the active learner.

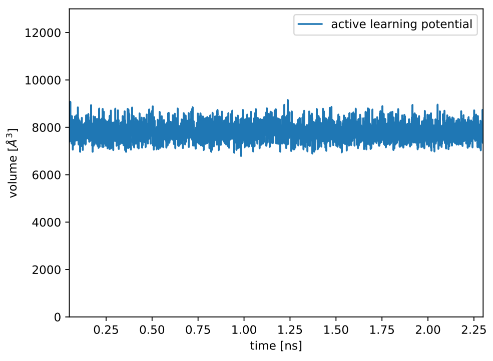
the simulation exploded after about 150 000 timesteps around 0.3 ns as depicted in figure 13.4 a). Using the active learner 33 configurations are added for a total of 180 configurations. The prediction for the energies, forces and virial on the validation data set can be seen in figure 13.3. The simulation of the active learner potential does not explode as it is depicted in 13.4 b). This indicates that the active learning process made the potential a lot more stable. It is unclear though if this is because the configurations selected by the active learner are selected very carefully, or if the same improvement could have been achieved by adding configurations for example with a uniform energy method.

13.3.1. Convergence

The potential in the last section seemed to become more robust by adding more configurations to the training data set using the active learner. Now we want to investigate if



(a) Initial Potential



(b) Active Learner Potential

Figure 13.4.: Volume in dependence of time of an NPT simulations. The initial potential fails (a) and the improved version is successful (b).

this is reflected in the accuracy of the predictions that the potential makes. We use the root-mean-square-error of the energy to assess the quality of the predictions

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_t - y_t)^2}{N}} \tag{13.1}$$

where N is the number of values, $\hat{y}_t$ is the correct value and y_t is the predicted value. An active learning cycle is conducted and every time the potential is retrained the energy RMSD is computed on a validation data set. The validation data is taken from the same MD trajectory that was also used to create the training data set for the initial potential. The RMSD energy values in dependence of the number of added configurations is depicted in Figure 13.5. The error fluctuates and does not converge. This indicates that the potential does not become better at predicting the configurations in the validation data set.

The new added configurations are not sampled with the same method that is used to create the validation data and the resulting configurations of both methods can be generally different. We therefore do not necessarily expect that the potential becomes better at predicting configurations of the validation data set. Additionally the purpose of the active learner was not to improve the predictions of configurations that are already predicted very well, but to improve on configurations that previously would have been predicted very poorly.

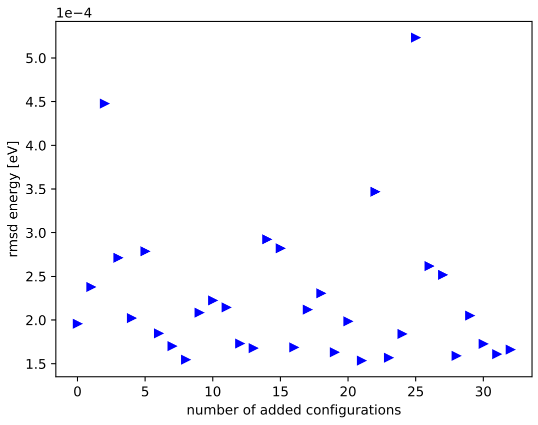


Figure 13.5.: Root mean square deviation of the energy in dependence of the number of configurations that were added to the training data set by the active learner. The RMSD is calculated on the validation data set that was created by a classical MD simulation.

14. Summary

This thesis focused on identifying, characterizing and fixing of ML potential problems. It starts out with investigating why Machine Learning NPT simulations often times fail. In order to do that, we look at possible causes. First the influence of long range interactions is investigated and specifically interactions that are longer then the descriptor cutoff. We set up two test systems, one uses the Coulomb potential and the other one uses the Wolf potential. The interactions of the Wolf potential are shorter than the descriptor cutoff, while the coulomb interactions are unlimited. Then we train Machine Learning potentials on both of those systems. We analyze the results by comparing the Machine Learning predictions of the energies to the validation data sets. The predictions of the coulomb system are more noisy then the Wolf system, but the mean field is learned correctly in both cases.

Next we investigate the influence of the training process. We train systems on different combinations of energy, forces and pressure and compare the results. The comparisons of the ML predictions with the validation data show, that it is necessary to train on pressure for accurate predictions of the pressure. Using this knowledge we are able to train an accurate well working NVT potential. But when this potential is used to run a NPT simulation, it fails. Next we explore the influence of the training data especially the density variation. We sample the training data set from five different NVT simulations with different volumes above and below the equilibrium density. This results in an NPT capable ML potential.

Sometimes it can happen that a Machine Learning simulation runs accurately until it encounters a configuration, that it can not predict. This can cause the simulation to fail. We try to identify, which configuration is responsible for the failure by looking at the physical quantities. By identifying the configuration, that is responsible, you can fix the potential for example by adding the configuration to the training data. When we look at the time dependence of the physical quantities we find that neither the energies, forces, temperature, velocities nor pressure can predict that the simulation will fail. This means we have to change the approach.

We compare the configuration space of DFT and classical simulations. We conduct a classical and a DFT simulation. Then we take a configuration out of the DFT data set and compare it to every configuration in the classical data set. The comparisons are conducted using the REMatch kernel, which can assess how similar two configurations are using the SOAP descriptor. This process is performed for Argon, NaCl and BMIMBF$_4$. Argon was the most similar followed by Nacl. BMIMBF$_4$ had the biggest differences in the configuration space between classical and DFT. This is probably because it is a lot harder to model more complex systems with empirical potentials. Knowledge about this is valuable for using methods that try to sample the configuration space to generate training data.

As a last project we implemented an active Learner into MLSuite, which is a software package. The algorithm consists of five different parts. The active learner tries to improve an existing potential. It starts by running a simulation a fixed amount of timesteps. Then it takes the final configuration of the simulation and compares it to the training data using the REMatch kernel. If the configuration is unique, it gets added to the training data. Then a new more robust potential is trained. The active learner algorithm is used to get a NPT potential to work, that previously exploded.

15. Zusammenfassung

Diese Arbeit konzentriert sich auf die Identifizierung, Charakterisierung und Behebung potenzieller Probleme von ML. Zunächst wird untersucht, warum ML NPT-Simulationen häufig fehlschlagen. Zu diesem Zweck werden mögliche Ursachen untersucht. Zunächst wird der Einfluss von Wechselwirkungen mit großer Reichweite untersucht, insbesondere von Wechselwirkungen, die länger sind als der Deskriptor-Cutoff. Wir stellen zwei Testsysteme auf, von denen eines das Coulomb-Potenzial und das andere das Wolf-Potenzial verwendet. Die Wechselwirkungen des Wolf-Potentials sind kürzer als der Deskriptor-Cutoff, während die Coulomb-Wechselwirkungen unbegrenzt sind. Anschließend trainieren wir die ML Potenziale auf diese beiden Systeme. Wir analysieren die Ergebnisse, indem wir die Vorhersagen der Energien mit den Validierungsdatensätzen vergleichen. Die Vorhersagen des Coulomb-Systems sind stärker verrauscht als die des Wolf-Systems, aber das mittlere Feld wird in beiden Fällen korrekt gelernt.

Als nächstes untersuchen wir den Einfluss des Trainingsprozesses. Wir trainieren die Systeme auf verschiedene Kombinationen von Energie, Kräften und Druck und vergleichen die Ergebnisse. Die Vergleiche der ML-Vorhersagen mit den Validierungsdaten zeigen, dass es notwendig ist, auf Druck zu trainieren, um genaue Vorhersagen für den Druck zu erhalten. Mit diesem Wissen sind wir in der Lage, ein genaues, gut funktionierendes NVT-Potenzial zu trainieren. Wenn dieses Potenzial jedoch zur Durchführung einer NPT-Simulation verwendet wird, versagt es. Als nächstes untersuchen wir den Einfluss der Trainingsdaten, insbesondere der Dichtevariation. Wir nehmen den Trainingsdatensatz aus fünf verschiedenen NVT-Simulationen mit unterschiedlichen Volumina oberhalb und unterhalb der Gleichgewichtsdichte. Das Ergebnis ist ein NPT-fähiges ML-Potenzial.

Manchmal kann es vorkommen, dass eine ML Simulation genau läuft, bis sie auf eine Konfiguration trifft, die sie nicht vorhersagen kann. Dies kann dazu führen, dass die Simulation fehlschlägt. Wir versuchen anhand der physikalischen Größen herauszufinden, welche Konfiguration für den Fehler verantwortlich ist. Indem man die verantwortliche Konfiguration identifiziert, kann man das Problem beheben, indem man beispielsweise die Konfiguration zu den Trainingsdaten hinzufügt. Wenn wir uns die Zeitabhängigkeit der physikalischen Größen ansehen, stellen wir fest, dass weder die Energien, Kräfte, Temperatur, Geschwindigkeiten, Kräfte noch der Druck vorhersagen können, dass die Simulation versagen wird. Das bedeutet, dass wir den Ansatz ändern müssen.

Wir vergleichen den Konfigurationsraum von DFT- und klassischen Simulationen. Wir führen eine klassische und eine DFT-Simulation durch. Dann nehmen wir eine Konfiguration aus dem DFT-Datensatz und vergleichen sie mit jeder Konfiguration im klassischen Datensatz. Die Vergleiche werden mit dem REMatch-Kernel durchgeführt, der anhand des SOAP-Deskriptors beurteilen kann, wie ähnlich sich zwei Konfigurationen sind. Dieser Prozess wird für Argon, NaCl und $BMIMBF_4$ durchgeführt. Argon war am ähnlichsten, gefolgt von Nacl. $BMIMBF_4$ hatte die größten Unterschiede im Konfigurationsraum zwischen klassischer und DFT. Dies liegt wahrscheinlich daran, dass es sehr viel schwieriger ist, komplexere Systeme mit empirischen Potenzialen zu modellieren. Das Wissen darüber ist wertvoll für die Verwendung von Methoden, die versuchen, den Konfigurationsraum abzutasten, um Trainingsdaten zu erzeugen.

Als letztes Projekt haben wir einen aktiven Lerner in MLSuite, ein Softwarepaket, implementiert. Der Algorithmus besteht aus fünf verschiedenen Teilen. Der aktive Lerner versucht, ein vorhandenes Potenzial zu verbessern. Er beginnt damit, eine Simulation mit einer festen Anzahl von Zeitschritten durchzuführen. Dann nimmt er die letzte Konfiguration der Simulation und vergleicht sie mit den Trainingsdaten unter Verwendung des REMatch-Kernels. Wenn die Konfiguration einzigartig ist, wird sie zu den Trainingsdaten hinzugefügt. Dann wird ein neues, robusteres Potenzial trainiert. Der aktive Lernalgorithmus wird verwendet, um ein NPT-Potenzial zum Laufen zu bringen, das zuvor explodiert ist.

16. Acknowledgments

I want to thank my supervisor Sam Tovey, for the advice and possibilities he gave me. I want to thank Fabian Zills for his help when technical programming details were needed and I want to thank Christian Holm for giving me the resources and possibility to do my master's thesis at the ICP.

Bibliography

[1] J.A. Barker, R.A. Fisher, and R.O. Watts. Liquid argon: Monte carlo and molecular dynamics calculations. *Molecular Physics*, 21(4):657–673, 1971.

[2] Albert P. Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Phys. Rev. B*, 87:184115, May 2013.

[3] Albert P. Bartók, Mike C. Payne, Risi Kondor, and Gábor Csányi. Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons. *Phys. Rev. Lett.*, 104:136403, Apr 2010.

[4] Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.*, 98:146401, Apr 2007.

[5] Jörg Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *The Journal of Chemical Physics*, 134(7):074106, 2011.

[6] Kieron Burke. Perspective on density functional theory. *The Journal of Chemical Physics*, 136(15):150901, 2012.

[7] Eve Bélisle, Zi Huang, Sébastien Le Digabel, and Aïmen E. Gheribi. Evaluation of machine learning interpolation techniques for prediction of physical properties. *Computational Materials Science*, 98:170–177, 2015.

[8] Stefan Chmiela, Huziel E. Sauceda, Klaus-Robert Müller, and Alexandre Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature Communications*, 9(1):3887, Sep 2018.

[9] Sandip De, Albert P. Bartók, Gábor Csányi, and Michele Ceriotti. Comparing molecules and solids across structural and alchemical space. *Phys. Chem. Chem. Phys.*, 18:13754–13769, 2016.

[10] Swapnil A. Dharaskar, Kailas L. Wasewar, Mahesh N. Varma, Diwakar Z. Shende, and ChangKyoo Yoo. Synthesis, characterization and application of 1-butyl-3-methylimidazolium tetrafluoroborate for extractive desulfurization of liquid fuel. *Arabian Journal of Chemistry*, 9(4):578–587, 2016.

[11] Mark Ebden. Gaussian processes: A quick introduction, 2015.

[12] Felix A. Faber, Luke Hutchison, Bing Huang, Justin Gilmer, Samuel S. Schoenholz, George E. Dahl, Oriol Vinyals, Steven Kearnes, Patrick F. Riley, and O. Anatole von Lilienfeld. Prediction errors of molecular machine learning models lower than

hybrid dft error. *Journal of Chemical Theory and Computation*, 13(11):5255–5264, Nov 2017.

[13] Daan Frenkel and Berend Smit. Chapter 4 - molecular dynamics simulations. In Daan Frenkel and Berend Smit, editors, *Understanding Molecular Simulation (Second Edition)*. Academic Press, San Diego, second edition edition, 2002.

[14] N. Galamba and B. J. Costa Cabral. First principles molecular dynamics of molten nacl. *The Journal of Chemical Physics*, 126(12):124502, 2007.

[15] Lauri Himanen, Marc O.J. Jäger, Eiaki V. Morooka, Filippo Federici Canova, Yashasvi S. Ranawat, David Z. Gao, Patrick Rinke, and Adam S. Foster. Dscribe: Library of descriptors for machine learning in materials science. *Computer Physics Communications*, 247:106949, 2020.

[16] Haoyan Huo and Matthias Rupp. Unified representation of molecules and crystals for machine learning. 2018.

[17] R. V. Krems. Bayesian machine learning for quantum molecular dynamics. *Phys. Chem. Chem. Phys.*, 21:13392–13410, 2019.

[18] Thomas D. Kühne, Marcella Iannuzzi, Mauro Del Ben, Vladimir V. Rybkin, Patrick Seewald, Frederick Stein, Teodoro Laino, Rustam Z. Khaliullin, Ole Schütt, Florian Schiffmann, Dorothea Golze, Jan Wilhelm, Sergey Chulkov, Mohammad Hossein Bani-Hashemian, Valéry Weber, Urban Borštnik, Mathieu Taillefumier, Alice Shoshana Jakobovits, Alfio Lazzaro, Hans Pabst, Tiziano Müller, Robert Schade, Manuel Guidon, Samuel Andermatt, Nico Holmberg, Gregory K. Schenter, Anna Hehn, Augustin Bussy, Fabian Belleflamme, Gloria Tabacchi, Andreas Glöß, Michael Lass, Iain Bethune, Christopher J. Mundy, Christian Plessl, Matt Watkins, Joost VandeVondele, Matthias Krack, and Jürg Hutter. Cp2k: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics*, 152(19):194103, 2020.

[19] Marcel Florin Langer, Alex Goessmann, and Matthias Rupp. Representations of molecules and materials for interpolation of quantum-mechanical simulations via machine learning. 2020.

[20] Ask Hjorth Larsen, Jens Jørgen Mortensen, Jakob Blomqvist, Ivano E Castelli, Rune Christensen, Marcin Dułak, Jesper Friis, Michael N Groves, Bjørk Hammer, Cory Hargus, Eric D Hermes, Paul C Jennings, Peter Bjerre Jensen, James Kermode, John R Kitchin, Esben Leonhard Kolsbjerg, Joseph Kubal, Kristen Kaasbjerg, Steen Lysgaard, Jón Bergmann Maronsson, Tristan Maxson, Thomas Olsen, Lars Pastewka, Andrew Peterson, Carsten Rostgaard, Jakob Schiøtz, Ole Schütt, Mikkel Strange, Kristian S Thygesen, Tejs Vegge, Lasse Vilhelmsen, Michael Walter, Zhenhua Zeng, and Karsten W Jacobsen. The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017.

[21] Eric Mjolsness and Dennis DeCoste. Machine learning for science: State of the art and future prospects. *Science*, 293(5537):2051–2055, 2001.

[22] Frank Noé, Alexandre Tkatchenko, Klaus-Robert Müller, and Cecilia Clementi. Machine learning for molecular simulation. *Annual Review of Physical Chemistry*, 71(1):361–390, 2020. PMID: 32092281.

[23] Eric Paquet and Herna Viktor. Computational methods for ab initio molecular dynamics. *Advances in Chemistry*, 2018:1–14, 04 2018.

[24] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.

[25] Evgeny V. Podryabinkin and Alexander V. Shapeev. Active learning of linearly parametrized interatomic potentials. *Computational Materials Science*, 140:171–180, 2017.

[26] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108:058301, Jan 2012.

[27] Robert F. Sekerka. 19 - canonical ensemble. In Robert F. Sekerka, editor, *Thermal Physics*, pages 305–335. Elsevier, Amsterdam, 2015.

[28] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[29] M.P. Tosi and F.G. Fumi. Ionic sizes and born repulsive parameters in the nacl-type alkali halides—ii: The generalized huggins-mayer form. *Journal of Physics and Chemistry of Solids*, 25(1):45–52, 1964.

[30] Mark E Tuckerman. Ab initiomolecular dynamics: basic concepts, current trends and novel applications. *Journal of Physics: Condensed Matter*, 14(50):R1297–R1355, dec 2002.

[31] Jiang Wang, Simon Olsson, Christoph Wehmeyer, Adrià Pérez, Nicholas E. Charron, Gianni de Fabritiis, Frank Noé, and Cecilia Clementi. Machine learning of coarse-grained molecular dynamics force fields. *ACS Central Science*, 5(5):755–767, May 2019.

[32] Shuwen Yue, Maria Carolina Muniz, Marcos F. Calegari Andrade, Linfeng Zhang, Roberto Car, and Athanassios Z. Panagiotopoulos. When do short-range atomistic machine-learning models fall short? *The Journal of Chemical Physics*, 154(3):034111, 2021.

[33] T. Zykova-Timan, D. Ceresoli, U. Tartaglino, and E. Tosatti. Physics of solid and liquid alkali halide surfaces near the melting point. *The Journal of Chemical Physics*, 123(16):164701, 2005.