Eindhoven University of Technology

Center for computational energy research

Design of a toy model database for predicting potential energy surfaces with neural nets

Tim Heiszwolf (1242343)

Supervisor:
Prof.dr.ir J.M.V.A. Koelman

July 25, 2020

Abstract

In the fields of physics and chemistry, potential energy surfaces are used to predict the properties and behaviors of materials starting from an atomistic description. But sadly, the ab-initio computation of these potential energy surfaces (or their gradients, the associated forces) requires significant computational resources, and scales badly with the number of atoms. With machine learning, a bit of accuracy can be sacrificed to render these calculations many orders of magnitude faster. This machine-learning approach requires large databases of samples of potential energy surfaces. Rather than relying solely on databases (generated at a high cost) using ab-initio methods, in this thesis a framework for generating databases using a toy model will be presented. Also, using a toy-model database, a small investigation into machine learning will be done. The advantage of using a toy model as a first test for machine learning algorithms is that doing so requires vastly less computational resources than testing it using ab-initio methods.

The toy model that has been designed calculates the energy per particle for cubic geometries with periodic boundary conditions. For this it sums all energies associated with particle pairs and triplets, which have at least one particle in the central periodic cell. The associated energy of these pairs and triplets is based on a modified Lennard-Jones potential with build in fast convergence. Using this method, the volume of computations needed is greatly reduced. Care is taken such that all required properties and behaviors of the potential energy surface are indeed obeyed by the toy model. Thus, the model combines a hard-core (excluded volume) behavior with attractive long-range interactions, is continuously differentiable, is symmetric under permutations of identical particles, can not be decomposed in just two particle interactions, and is quite general and easily adaptable.

When generating the database, a relaxation process can be applied to reduce the sampling of unphysical energies. But when doing this, the amount of particle scans for which the relaxation is done should be carefully chosen, since too much relaxation can result in certain configurations becoming very prominent in the data set. After the generation of the database it can also be prepared for machine learning by calculating the eigenvalues of so called proximity matrices. These eigenvalues are continuous, continuously differentiable, and translational and permutation symmetric and thus are ideally suited as input parameters in the machine learning process.

Finally, when designing and training neural networks several small networks were found to work acceptably on a single cell width (especially displaying a very accurate gradient of the energy per particle), but larger networks which are more accurate, and networks which work for various widths, can probably also be designed. It was found that, in general, having fewer inputs, and inputs of which their proximity matrices had a lower order, performed better.

Contents

1 Introduction

1.1 Introduction

A Potential Energy Surface (PES) is a function mapping a configuration of atoms onto a potential energy. In the fields of physics and chemistry, potential energy surfaces are used to link the properties and behaviors of materials to their atomic compositions. This is typically done in Monte Carlo (MC) or Molecular Dynamics (MD) simulations. The discovery and screening of materials optimised for specific energy applications (e.g. for developing better batteries or improved solar cells) requires accurate PES's for rather complex systems.

But sadly, the ab-initio computation of these potential energy surfaces (or their gradients, the inter-atomic forces) takes a very long time. With machine learning, a bit of accuracy can be sacrificed while rendering these calculations significantly faster. However, machine learning approaches (such as neural nets) require large databases containing atom configurations and their associated PES's. In this thesis, a method and framework for generating databases for the training of these networks using a toy model will be presented, and a small investigation into machine learning will be done. The intended use of a database of toy model results is the preliminary testing and tuning (hyper-parameter setting) of novel machine-learning algorithms. Advantage of using a toy model at such an early stage of testing is to avoid wasting large computational resources on machine learning algorithms that might well get discarded. This will rely on the toy model to be capable of generating a (large) database of results significantly faster than generating such a database using ab-initio simulations.

The use of machine learning is not a new thing for calculations of the potential energy surface, but many improvements can still be made. Hopefully a method to generate databases of toy-model results and the few examples given here will help the effort.

1.2 Problem

Thus, the important goal of this thesis is to find a method of generating such a database that contains many different configurations of particles with each an associated energy from a toy model. That energy of such a toy model should only depend on the position of the particles and take into account periodic (infinitely repeating grid of identical cells) boundary conditions (as to make it more applicable to real materials), behave well and according to what you would expect of a physical system, be complex and finally it should also be quite general but it should also be easily adapted into more complex situation. The potential energy (and database) should work in both 2D and 3D, but for sake of simplicity in this thesis all examples will be done in 2D.

The requirements of the toy model potential energy are:

- Depend only on the particle positions and take into account the repeating boundary conditions,
- Be continuously differentiable,
- Be invariant under translations of the periodic grid,
- Be invariant under permutations of identical particles,
- Combine a hard repulsive core with an attractive long range interaction,
- Be more complex than a sum of two-particle interactions,
- Be general and easily adaptable,
- Be faster than DFT simulations.

With this toy model, a database should then be generated. That should then also be prepared for machine learning such that it contains a suitable distribution over energies.

The requirements of the generated and prepared database are:

- Contains a reasonable range of energies,
- Inputs are translation symmetric,
- Inputs are permutation symmetric,
- Inputs are continuously differentiable.

The database should then also be tested to make sure it works and to get some quick insight into what best can be done to optimize the machine learning process (e.g. hyperparameter optimisation).

The resulting neural networks should:

- Be accurate.
- Have a derivative (inter-particle force) that behaves well.

And that neural network should of course then also be properly tested and optimized.

1.3 This thesis

First, some theory will be explored to get an understanding of existing (conventional) approaches. This will be done in chapter 2. Next, to solve the problem it is first crucial that different toy models are designed and evaluated. In chapter 3.1, a method for calculating the potential energy will be defined and different solutions will be tested, evaluated and refined. The structure of the program that will generate and prepare the database is made in chapter 3.2 and 3.3. Here, the structure of the program needs to be easily modifiable and flexible. Next, the speeds of the algorithm needs to be evaluated in chapter 4.1. The settings for optimal database generation need to be found and the results of the database generation should be evaluated. This will be done in chapter 4.2. The application to machine learning is also crucial and thus should also be done on a small scale to verify if the database works and to perhaps also do some hyper-parameter optimisation. This is done in chapter 5. Finally, the things that should be done in the future of continued research should be discussed. This will be done in chapter 6.2.

2 Theory

The potential energy related to forces between atoms/particles is called the inter atomic energy [9]. As stated in the introduction, these inter-atomic potentials form a potential energy surface which can be used to determine the properties of the material, which is very useful. A potential energy surface describes the potential energy of a system of particles as a function of the configuration of these particles.

There are two methods for calculating these inter-atomic potentials. The first is by using quantum mechanics, and solving Schrödinger's equation; this is typically done using Density Functional Theory (DFT) simulations. These calculations are very useful since they can be made quite accurate, and they are generic in the sense that they can be applied to systems containing arbitrary mixtures of atoms. But sadly, DFT computations are very slow and the amount of CPU time required typically scales with the cube of the number of particles present in (a unit cell of) the simulation $\mathcal{O}(N^3)$ [13].

Another method is using toy models. These toy models can often still be applied to real cases by adjusting the parameters of the model. The result will then be a approximation but they can be quite accurate [9]. These toy models are constructed as a sum of pair, triplet, quarter, etc interactions

$$V = \sum_{i,j\neq i}^{N} V_2(\vec{r}_i, \vec{r}_j) + \sum_{i,j\neq i,k\neq j\neq i}^{N} V_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \ldots \tag{1}$$

where V is the total potential, N is the number of particles, V_n is the n-particle interaction energy and $\vec{r}$ is the position vector of a particle. Sometimes, each of these terms can have a different weighing factor to make them more or less important [9]. For dilute gases and simple fluids, two-particle interactions are accurate enough, but for more complex systems such as metals and covalent solids, often two-particle interactions are not sufficient and three-particle and even higher-order interactions are needed [9].

In general, a two particle contribution to the interaction consist of two parts, each part is a function of the distance between the two particles. First, a short range interaction part which rapidly asymptotically approaches zero from the positive direction and makes sure a lot of energy is needed if the atoms are brought close together. Second, there is the long range interaction part which more slowly approaches zero from the negative direction and acts as a attractive force for long distances. Combined together, this results in a hard-core potential when the distance between atoms is small a (negative) minimum at a certain distance which then goes asymptotically to zero as the distance between atoms becomes large [9].

Two-particle potentials that have seen some intensive use in literature are the Morse potential and the Lennard-Jones potential. The Morse potential (see figure 1a) is

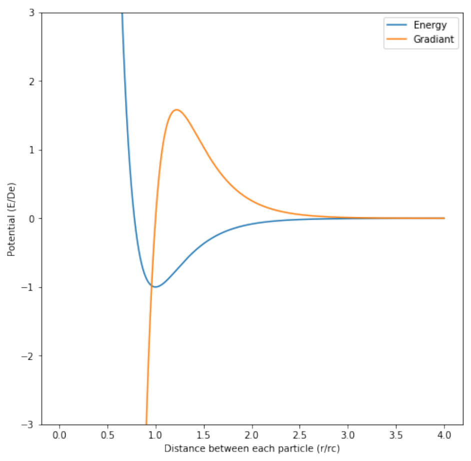
$$V = D_e \left(e^{-2a(r-r_e)} - 2e^{-a(r-r_e)} \right), \quad a = \sqrt{\frac{k_e}{2D_e}} \tag{2}$$

where r is the distance between atoms, r_e is the distance at which the energy is minimum, D_e is how deep the minimum is compared to infinity and k_e is the value of the spring constant of the second order approximation of the energy at the minimum (so it corresponds with how wide the well is). It is a simple atomic potential, and is used a bit in quantum mechanic for diatomic molecules [5].

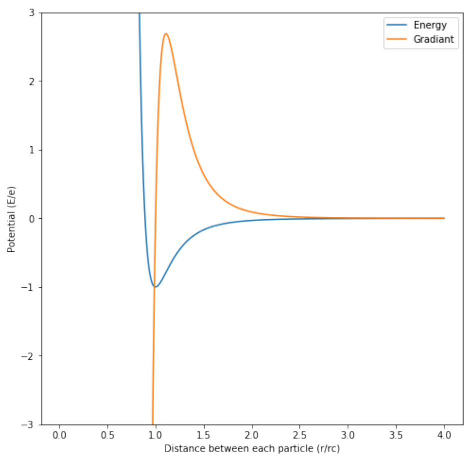
The Lennard-Jones potential (see figure 1b) is

$$V = \epsilon \left((\frac{r_m}{r})^{12} - 2(\frac{r_m}{r})^6 \right) \tag{3}$$

where ϵ is the energy of the minimum and r_m is the location of the minimum. This potential can be used for simple approximations of energies between neutral atoms [9].



(a) The Morse potential with $k_e = 20D_e$.



(b) The Lennard-Jones potential.

Figure 1: Two examples of a two particle interaction potential.

Predicting potential energy surfaces with machine learning (like neural networks) is not a new thing and has already been successfully done. Often, a training data set is generated using DFT simulations (or similar) which is then used to train the neural network. The inputs of these networks are rarely just the coordinates of the particles, instead often the distances between the particles are processed such that makes the job of the neural network easier by for example incorporating symmetries, and by having cutoffs such that far away particles (with little relevance) are not taken into account [3]. These neural networks with prepossessing scale significantly better than DFT simulations and also perform several order of magnitude better than DFT simulations while still only being of by several meV per particle [15].

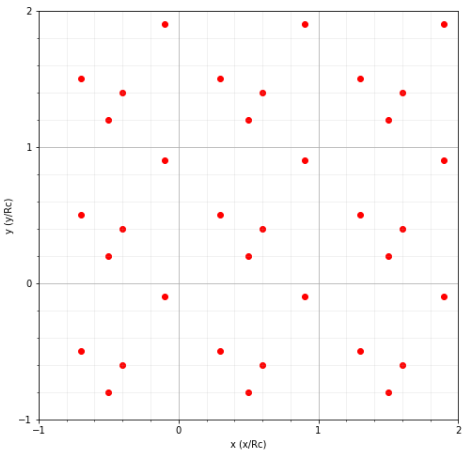
There are several currently popular methods. One is 'high-dimensional neural network potential' (NNP). Here, for each atom in the configuration, a separate network is trained (and the sum of all outputs is the total energy) and uses the local environment around that atom. The NNP performs well for silicon, titanium dioxide, water and more. Another methods is 'Gaussian approximation potential', which works great on transition metals, main group elements and multi component systems [15].

3 Solution

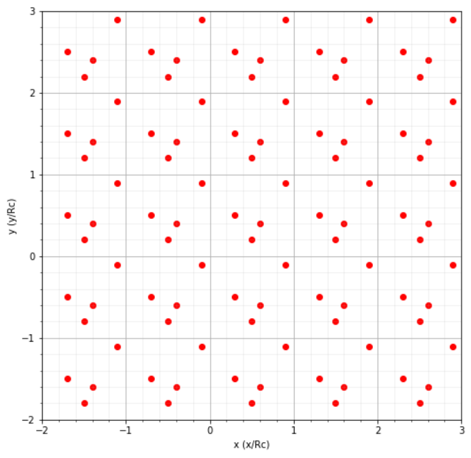
3.1 Potential energy

As stated in chapter 1.2 the toy model for the potential energy should only depend on the position of the particles, be continuous, be continuously differentiable, display complex behavior but be similar to real life potentials, should be relatively fast and should be very general.

For the lattice, a structure composed of square unit cells of with a certain width w_c is chosen. Each unit cell contains a specific number of particles N_p. The cell has infinite repeating boundary conditions. But to make sure the computation can finish, only a certain depth of surrounding cells d_s is taken into account. This is the number of cells that needs to be passed trough to reach the part of the lattice that isn't taken into account anymore from the central cell (see figure 2 for an example). Of course, d_s needs to be large enough such that the potential energy has converged, but a larger depth also means that more particles need to be taken into account and thus more calculations have to be done. When discussing the potential energy, the required depth for good convergence will be explored further.



(a) Depth $d_s = 1$ so 3 by 3 cells.



(b) Depth $d_s = 2$ so 5 by 5 cells.

Figure 2: A plot of the lattice with a cell width of $w_c = 1R_c$ for a different depth of surrounding cells.

As explained in chapter 2 the interatomic potential can be expressed as a function of pair, triplet, quarter, etc interactions [9]. Only taking into account pair interactions is insufficient since that doesn't have enough complexity. So instead, the toy model that is going to be constructed is going to take into account only three-particle interactions. Any higher-order (e.g. four-particle) interactions can in principle be included, but will greatly increase computational requirements. Since the potential energy increases if you increase the amount of particles (and thus doesn't converge in a nice way) what is going to be calculated is the amount of potential energy per particle. Thus, the energy per particle is defined as

$$v = \frac{V}{(2d_s+1)^d N_p} = \frac{\sum_{i,j\neq i,k\neq j\neq i}^{(2d_s+1)^d N_p} V_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)}{(2d_s+1)^d N_p} \tag{4}$$

here v is the energy per particle, d is the dimension of the problem (for simplicity sake all examples will be in 2D), V_3 is the function for the potential of three particles and $\vec{r}$ is the position of a particle. The equation can also be easily expanded to include interactions between two, four, five, etc particles.

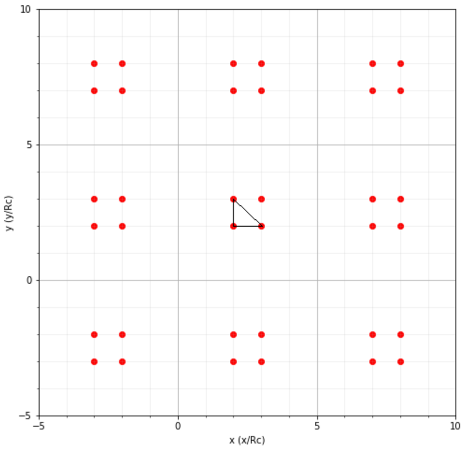
The evaluation of this equation is very compute intensive since it scales with the cube of the total number of particles $\mathcal{O}(d_s^{3d}N_p^3)$. This is because the summation takes into account each and every possible triangle [14] and since for this equation to converge d_s also needs to be large (such that the long range interactions are not truncated) the total number of triangles/calculations must be high. To solve this we can take into account just the central unit cell and ignore every triangle that doesn't have at least one particle at its vertices. This makes the equation for potential energy per particle

$$\begin{aligned}
v &= \frac{V_c}{N_p} = \frac{V_{c1} + V_{c2} + V_{c3}}{N_p} \\
V_{c1} &= \sum_{i}^{N_p} \sum_{j,k \neq j}^{((2d_s+1)^d-1)N_p} V_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) \\
V_{c2} &= \sum_{i,j \neq i}^{N_p} \sum_{k}^{(2d_s+1)^d-1)N_p} V_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) \\
V_{c3} &= \sum_{i,j \neq i,k \neq j \neq i}^{N_p} V_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)
\end{aligned} \tag{5}$$

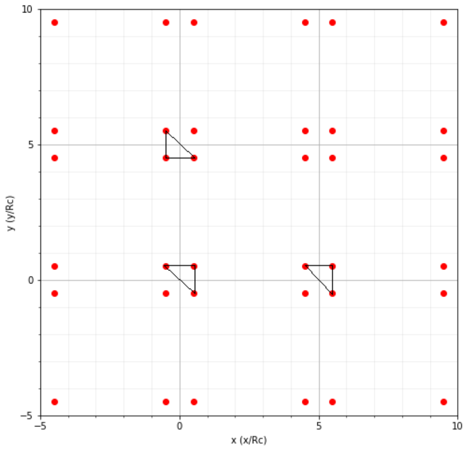
where V_c is the energy per unit cell, V_{c1} is the contribution of the triangles with one particle of the unit cell on its vertices, V_{c2} the contribution of the triangles with two particles and V_{c3} is the contribution of the triangles with the unit cell. This equation still scales cubicly with the number of particles in the unit cell but does so in such a way that d_s can be significantly smaller and in a way that d_s only scales the amount of calculations much less aggressively $\mathcal{O}(d_s^{2d}N_p^3)$ [14]. This is especially beneficially when calculating with more than two dimensions. This equation can still easily be adapted to work with potentials that involve two, four, five, etc particles.

In equation (5), some triangles are counted double or even triple. To reduce the amount of calculations even further, one can exclude the extra triangles by replacing each is-not-equal sign in equation (5) with a is-larger-than sign. But sadly, this will not work, since for equation (5), translational symmetry is not conserved. In figure 3, an example where four particles are placed in a square which are then moved to the edge of the cell is given. The configuration in which the particles are near the edge of the cell is transitionally symmetric to the configuration in which the particles are in the center (take the four particles and translate them from the center to the edge of a cell). But as can be seen the triangle that would be counted once in the central configuration gets counted trice in the configuration with the particles at the edge. This can also be seen in figure 3c where the energy on the outside is around three times as large as on the inside.

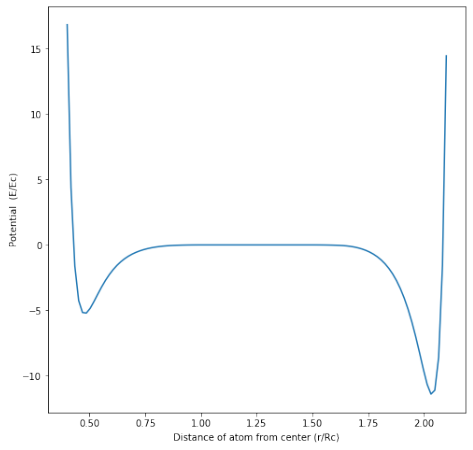
Thus, this approach can't be used for equation (5) where only triangles with at least one vertex in the unit cell are considered. The method can be used for equation (4) where all triangles are taken into account, but this method would only reduce the amount of calculations by a factor 6 at most which is not enough to make it as fast as the other method since it still scales badly. Also counting triangles multiple times only affects the magnitude of the potential energy surface and not the shape. Thus it doesn't affect the use of the potential energy function for machine learning.



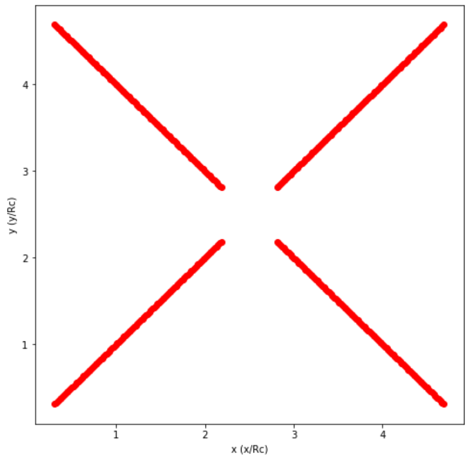
(a) The configuration with all the particles in the middle of the cell.



(b) The configuration with all the particles on near the edge of the cell.



(c) The energy of the particles as they move away from the center and towards the edge (see figure 3d). The distance is half the length of the side of the square. This calculation was done with a higher number of surrounding cells ($d_s = 4$) such that edge effects are negligible.



(d) The movement of the particles in the unit cell. They start at a distance of 0.25 form each other and end at distance 0.25 from the edge of the cell.

Figure 3: The two lattices should yield identical energies since they are transitionally identical but as you can see the example triangle in figure 3a gets counted three times in the transitionally symmetric figure 3b. The asymmetric in energy can be seen in figure 3c.

So now a function for V_3 needs to be designed. As said in chapter 1.2, it needs to display physical behavior, be continuous, be continuously differentiable, should converge well but most importantly be general and easily adaptable. In general these potentials should consists of two parts (see chapter 2): a repulsive short range part V_s that goes asymptotically to zero quickly once the distances increase and a attractive long range part V_l that goes to zero asymptotically but at a slower rate then the short range part.

To make it easily adaptable, it should have a characteristic distance R_c and a characteristic energy E_c similarly to the Lennard-Jones and Morse potential. These are somewhat arbitrarily defined but these do allow such equation be more applicable to different materials since for each of these materials these characteristics can be tweaked. The characteristic distance is defined such that in a equilateral triangle the energy in minimal if the distance between particles is the characteristic distance. This is a nice definition since this give a distance at which the particles would naturally like to be. The characteristic energy is defined as the minimum energy obtained. The general procedure for making such potentials can be found in appendix A.1).

To be able to test, this several tests were designed to see check the properties of the potential energy function. First, the energy of a single equilateral triangle will be calculated as the distance between particles becomes larger. With this, the general curve of the potential can be checked. Next, the same triangle is placed in a cell with a very large width such that only the particles in the unit cell contribute significantly to the total energy. This curve should then result in a identical curve to the first test, but around six times higher (due to triangles being counted multiple times). To check if symmetry is conserved, a test like in figure 3 will also be done. The potential energy surface is also generated and inspected (the configuration from which this was generated can be seen in figure 4).

To see if the function behaves physically accurate, two particles will be placed close together (distance of $\frac{R_c}{2}$) while a third particle moves away. If the energy approaches zero as the particle moves far away, it doesn't behave physically accurate since the two particles should still repel each other (and thus have a high energy).

Finally, the convergence is checked by calculating the potential energy per particle of a standard lattice (see figure 2) for an increasing depth of surrounding cells.

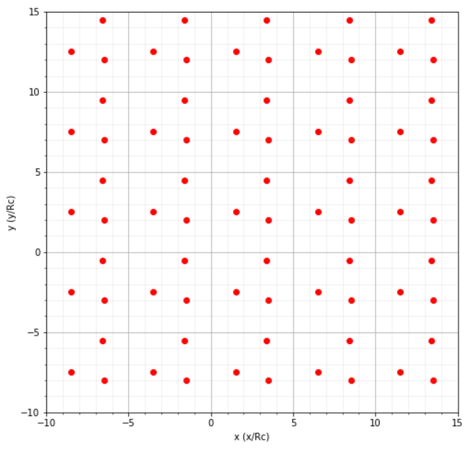


Figure 4: The configuration which is used for the generation of a potential energy surface. A fourth particle is place at each coordinate and the potential is calculated.

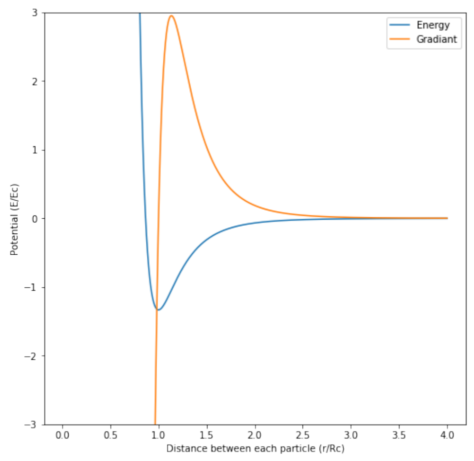
3.1.1 Short range area and pair interactions, long range circumference

The first idea for a function of the triplet interaction was a function in which the short range potential depends on pair interaction that is made complex by multiplying it by the area of the triangle, and the long range potential depends on the circumference of the triangle which also is complex and is not equivalent to a sum of pair interactions. This resulted in the following potential

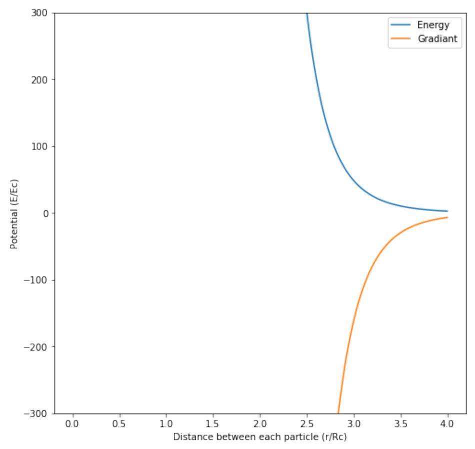
$$V_3 = V_s + V_l, \quad V_s = \frac{16E_c}{3}\frac{A^2}{R_c^4}\left(\frac{R_c^{12}}{a^{12}} + \frac{R_c^{12}}{b^{12}} + \frac{R_c^{12}}{c^{12}}\right), \quad V_l = \frac{-2916E_c}{\left(\frac{a}{R_c} + \frac{b}{R_c} + \frac{c}{R_c}\right)^6} \tag{6}$$

where A^2 is the square of the area of the triangle as calculated with Heron's formula, a, b and c are the lengths of the sides of the triangle formed by the three particles, E_c is the characteristic energy and R_c is the characteristic distance. The pre-factors of the long range and short range parts make sure the characteristic distance and energy are at R_c and E_c. The method for determining these pre-factors can be seen in appendix A.1. The results of the tests can be seen in figure 5. While the curve of the energy of a single triangle looks very good, in the rest strange things can be spotted. First of all, the potential of a triangle in empty space seems to be very high and doesn't have the characteristic dip. The square test doesn't seem to produce symmetric results. The repulsive interaction between two particles seem to work well. Also the energy per particle doesn't converge if the depth of the surrounding cells increases, in fact the energy only grows larger. So this potential is not fit for usage.

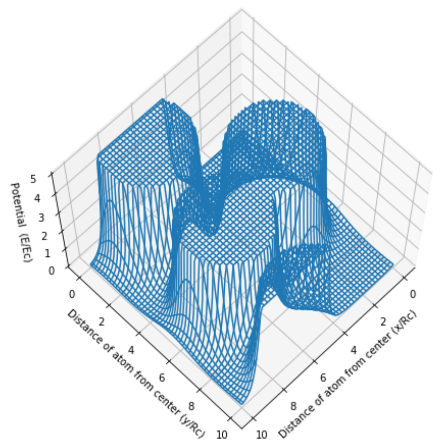
The reason this potential fails is because pair interactions in the short range term can be very large if only two particles are close to each other. If the third particle is very far away the area is large and thus a energy that is not small is obtained. This results in very high energies in empty space when two particles are quite close to each other and the third is in another cell. This also prevent convergence from happening since for increasing depth more particles are added to the lattice (which are also further away), and thus, the area of the two particle interaction becomes larger while the size of the pair interaction stays around the same size.



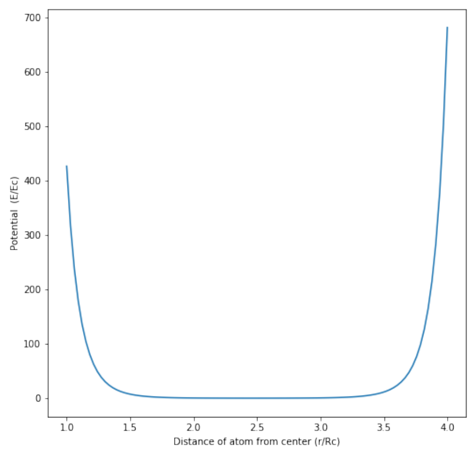
(a) The potential of a single triangle.



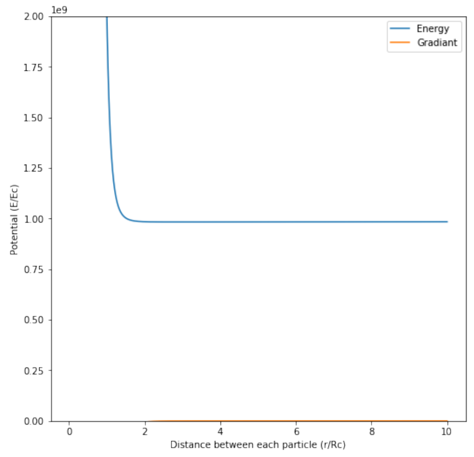
(b) The potential of a single triangle in a large space as calculated with equation (5).



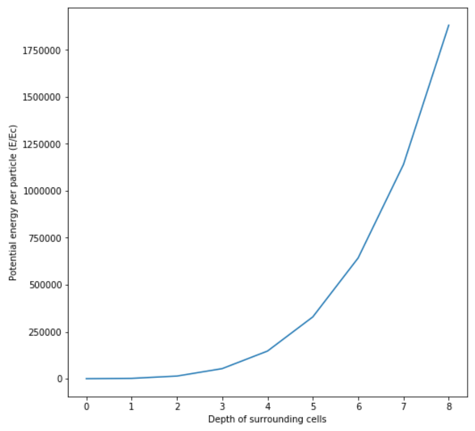
(c) The potential energy surface of the test configuration limited to $5E_c$ and the cell width is $10R_c$ due to high overlap.



(d) The energy of the square of particles moving away from each other and the cell width is $10R_c$ due to high overlap.



(e) The energy of the configuration for testing repulsion.



(f) The convergence of energy per particle for a cell width of 5.

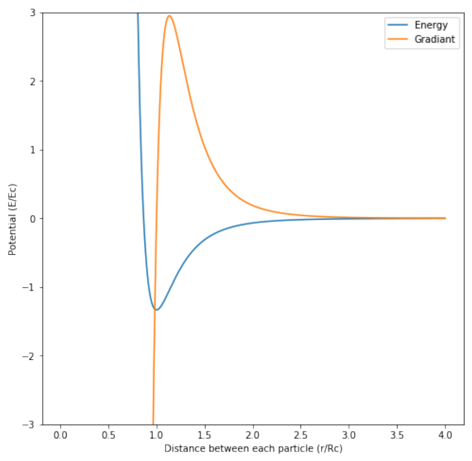
Figure 5: The results of the tests for the potential with short range area and pair interactions and long range circumference.

3.1.2 Short range area and circumference, long range circumference

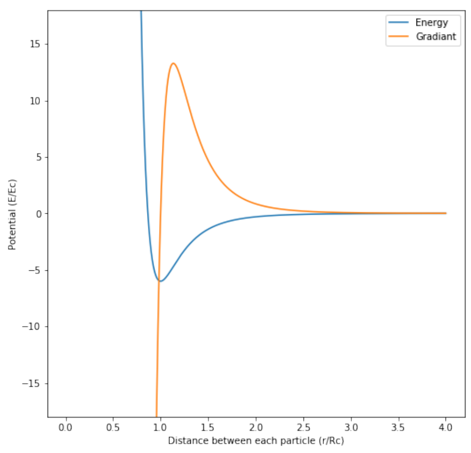
The next idea was to then change the pair interaction term in the short range potential to a circumference depended term. That term would become very small if the third particle was very far away. And thus, the potential became

$$V_3 = V_s + V_l, \quad V_s = 8503056E_c \frac{A^2}{R_c^4} \frac{R_c^{12}}{(a+b+c)^{12}}, \quad V_l = \frac{-2916E_c}{(\frac{a}{R_c} + \frac{b}{R_c} + \frac{c}{R_c})^6} \tag{7}$$

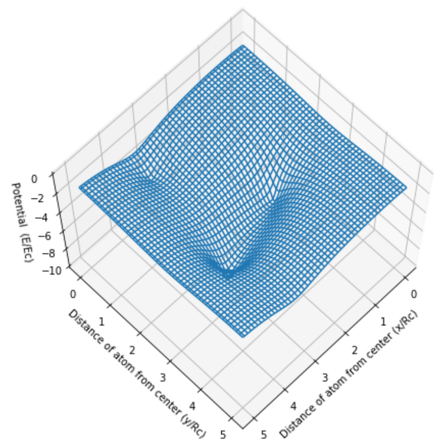
which resulted in the tests as seen in figure 6. Again method for determining the pre-factors can be seen in appendix A.1. The results are much better in the single triangle, three particles in a empty space, square configuration and convergence test. But sadly, in the repulsive tests it fails since the energy approaches zero once when the third particle moves to a large distance and more importantly since the energy becomes negative if the third particle comes close (also see the potential energy surface). This is of course because the circumference term in the short range term becomes very small once the third particle moves away and thus there is nothing to give it high energy. This means the potential energy function can't be used.



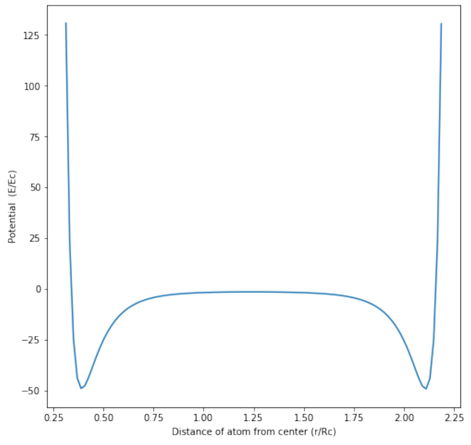
(a) The potential of a single triangle.



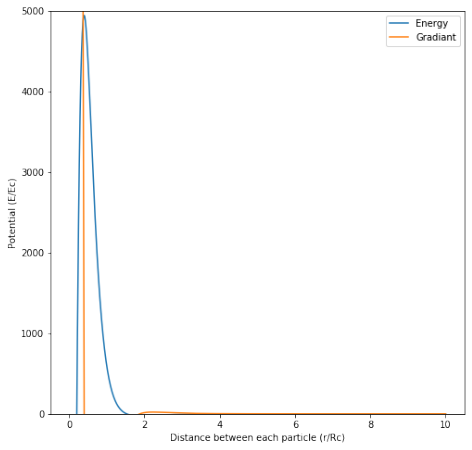
(b) The potential of a single triangle in a large space as calculated with equation (5).



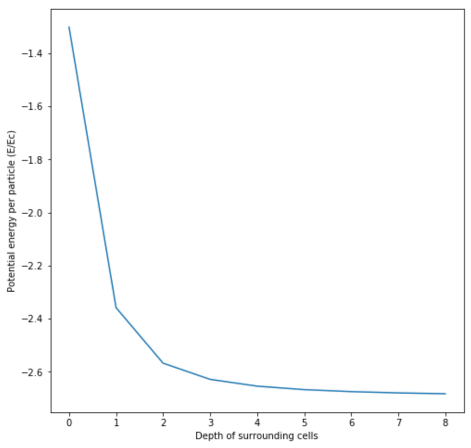
(c) The potential energy surface of the test configuration.



(d) The energy of the square of particles moving away from each other.



(e) The energy of the configuration for testing repulsion.



(f) The convergence of energy per particle for a cell width of 5.

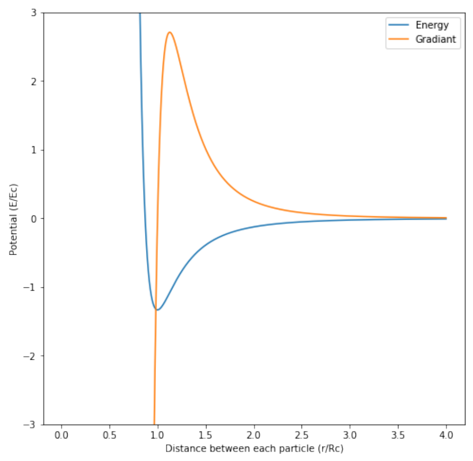
Figure 6: The results of the tests for the potential with short range area and circumference and long range circumference.

3.1.3 Short range pair interaction, long range area and circumference

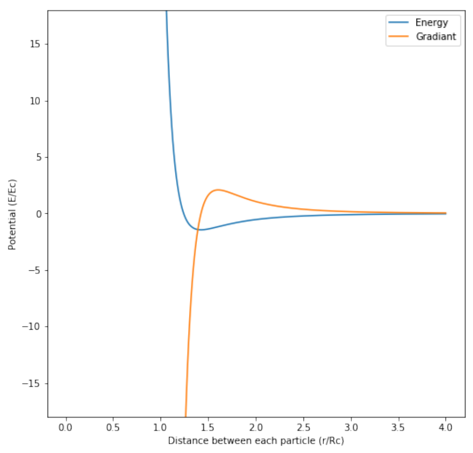
So it is clear that some kind of pair interaction is needed to make sure the repulsive properties are properly captured in the potential energy function. Thus, the short range potential was reduced to a simple pair interaction, but the long range was made more complex by adding the area to the circumference depended term. This resulted in

$$V_3 = V_s + V_l, \quad V_s = \frac{E_c}{6}(\frac{R_c^{12}}{a^{12}} + \frac{R_c^{12}}{b^{12}} + \frac{R_c^{12}}{c^{12}}), \quad V_l = -1458\sqrt{3}E_c\frac{A}{(\frac{a}{R_c} + \frac{b}{R_c} + \frac{c}{R_c})^6} \tag{8}$$

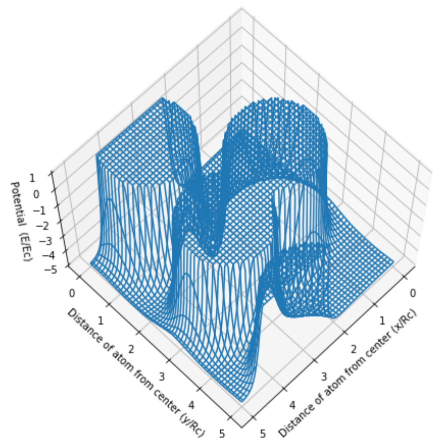
as a potential. The single triangle curve looks acceptable but not great (due to the the long time before it becomes zero), and the repulsion is also done well. But the energy doesn't converge. This is because two particles at short range and a third particle in long range (for example) in another cell cause high contributions (just like the other potential). The square test also seems to yield a non-symmetric result. Thus, this potential energy function was also rejected.



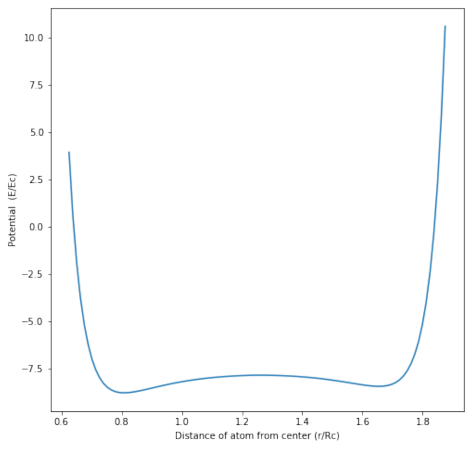
(a) The potential of a single triangle.



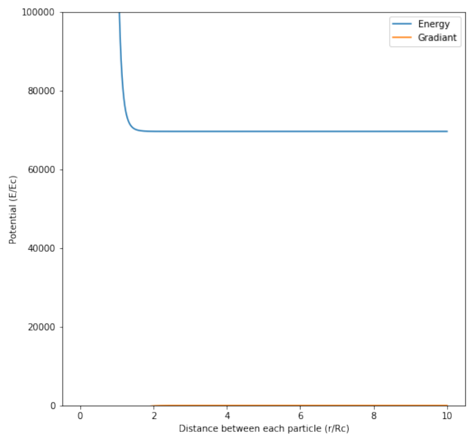
(b) The potential of a single triangle in a large space as calculated with equation (5).



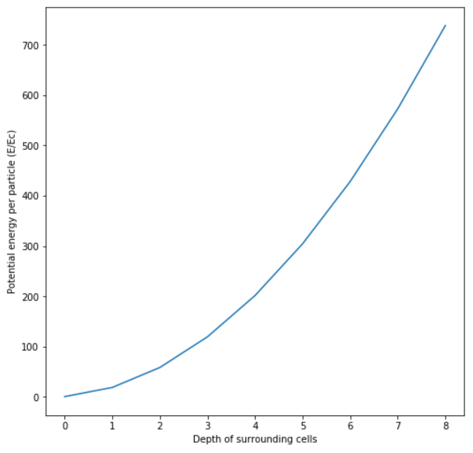
(c) The potential energy surface of the test configuration. It is cut-off beyond E_0.



(d) The energy of the square of particles moving away from each other.



(e) The energy of the configuration for testing repulsion.



(f) The convergence of energy per particle for a cell width of 5.

Figure 7: The results of the tests for the potential with short range pair interaction and long range area and circumference.

3.1.4 A modified Lennard-Jones potential

Since two particle interaction didn't seem to work if it is combined with triplet interaction, a new approach was tried. Two particle interaction was added to equation (5) by adding two terms

$$
\begin{aligned}
v &= \frac{V_c}{N_p} = \frac{3V_{c1} + 3V_{c2} + 3V_{c3} + V_{tc1} + V_{tc2}}{4N_p} \\
V_{tc1} &= \sum_{i}^{N_p} \sum_{j}^{((2d_s+1)^d-1)N_p} V_2(\vec{r}_i, \vec{r}_j) \\
V_{tc2} &= \sum_{i,j\neq i}^{N_p} V_2(\vec{r}_i, \vec{r}_j)
\end{aligned} \tag{9}
$$

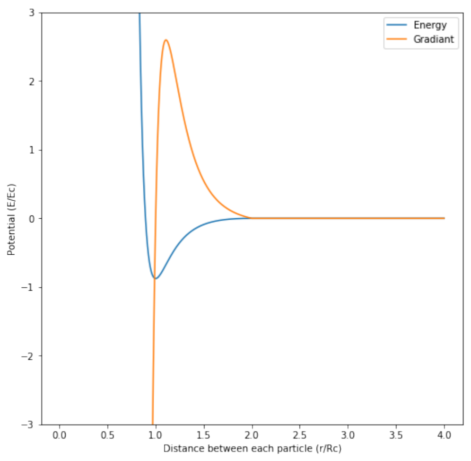
where V_{tc1} is the two particle interaction term with at least one particle in the central cell, V_{tc2} is the two particle term in which both particles are in the central cell and V_2 is a function for a two particle potential. The factor $\frac{3}{4}$ for triplet interaction and factor $\frac{1}{4}$ for pair interaction was chosen such that the domination factor would be the triplet interactions unless two particles got close to each other.

Next, a potential energy function is needed. For this, a modified Lennard-Jones potential was chosen. It was modified such that the minimum was at R_c and such that it has build in convergence at R_0. The potential for a two or three particle interaction is

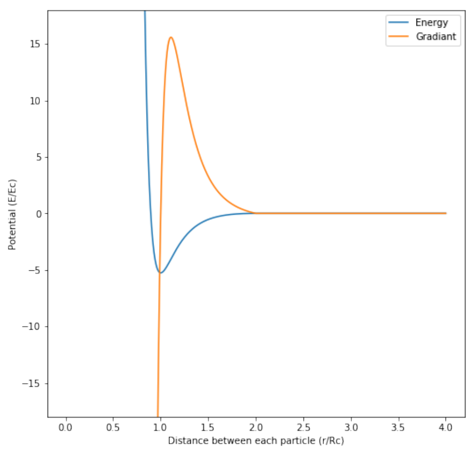
$$
\begin{aligned}
V_{2/3} &= E_c(V_s + V_l + V_c) && \text{for } s < R_0 \\
V_{2/3} &= 0 && \text{for } s \geq R_0 \\
V_s &= \frac{R_c^{2n}}{s^{2n}} - \frac{R_c^{2n}}{R_0^{2n}} \\
V_l &= -2\left(\frac{R_c^n}{s^n} - \frac{R_c^n}{R_0^n}\right) \\
V_c &= -\frac{2n}{R_0^{n+1}}(1 - \frac{R_c^n}{R_0^n})(\frac{s}{R_c} - \frac{R_0}{R_c})
\end{aligned} \tag{10}
$$

where V_c is the term that makes sure the potential is continuous and continuously differentiable at R_0 and s is the semi-circumference of the triangle formed by the three particles (and the distance between the particles for two particles). For $s \geq R_0$, the potential is 0. One disadvantage is that the potential doesn't have a perfect characteristic distance and energy since the minimum is only at R_c and E_c for $\frac{R_0^n}{R_c^n} \gg 1$. But luckily, this condition is quite easy to fulfill. One advantage of this method is that the depth of surrounding cells only needs to be $d_s = \text{roundup}(\frac{3R_0}{2w_c})$ (assuming equation (9) is used). The construction of this potential can be found in appendix A.2.

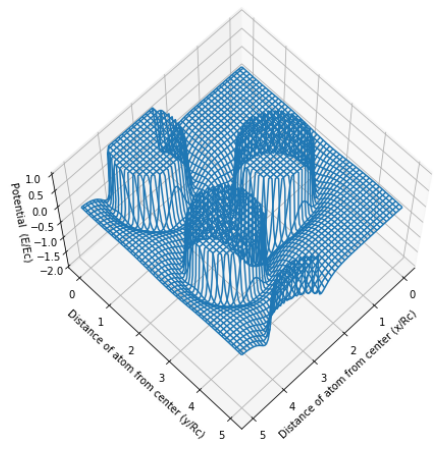
But the important question is: how does it behave? The answer is quite good. For $n = 6$ and $R_0 = 2R_c$ the results are displayed in figure 8. The potential on its own has a nice shape, is continuously derivative and has a minimum at $-0.88E_c$ which is nearly $-E_c$. The triangle in empty space also behaves well. The results of the energy surface and the square test also look good and are symmetric. Since two pair interaction is taken into account, the repulsion also happens when two particles are close to each other, and the convergence is also very good since it converges within only a depth of surrounding cells of $d_s = 1$. The only large disadvantage of this equation is that particles at a distance greater than $2R_c$ from each other don't interact at all (not even a tiny bit). All things considered, this equation was chosen as the potential that was going to be used.



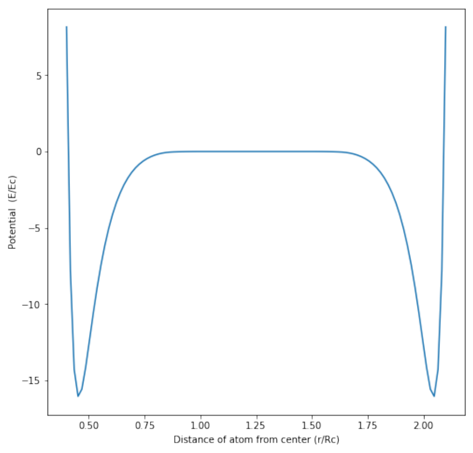
(a) The potential of a single triangle or pair.



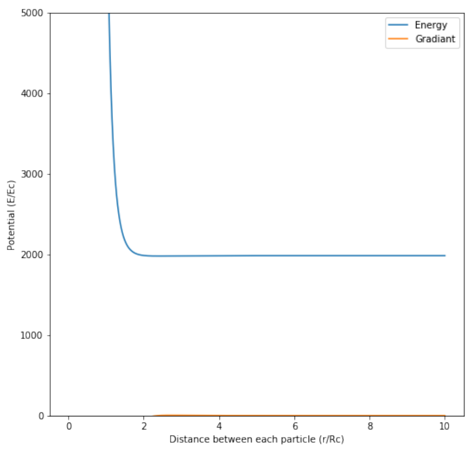
(b) The potential of a single triangle in a large space as calculated with equation (9).



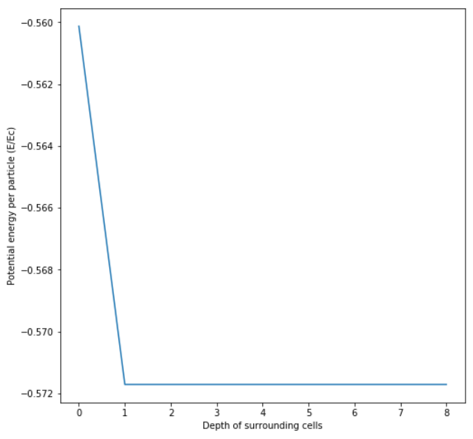
(c) The potential energy surface of the test configuration. It is cut-off beyond E_0.



(d) The energy of the square of particles moving away from each other.



(e) The energy of the configuration for testing repulsion.



(f) The convergence of energy per particle for a cell width of 4 (made smaller because otherwise it head nearly no interaction with the neighbouring cells).

Figure 8: The results of the tests for the modified Lennard-Jones potential with $n = 6$ and $R_0 = 2R_c$.

3.2 Structure and general properties of the program

To generate the database, a script needed to be written. For this several programming languages were considered. Python, Java and MATLAB were considered (these are the languages in which the author has experience). The important factors in picking the language are the speed of development and how readable it is (this makes adjusting code that was already written much easier), the amount of (relevant) modules (which reduce the amount of code that needs to be written), the speed of execution (to make the generation of a large database faster), the popularity of the languages in the scientific world (such that it can easily be used by other people), availability (such that anyone can use it, even those not part of a university) and finally the amount of experience the author has with the languages. A overview and evaluation of these factors can be seen in table 1.

Table 1: The advantages and disadvantages of different programming languages for the database generation. The rating ranges from — to +++ but are somewhat subjective.

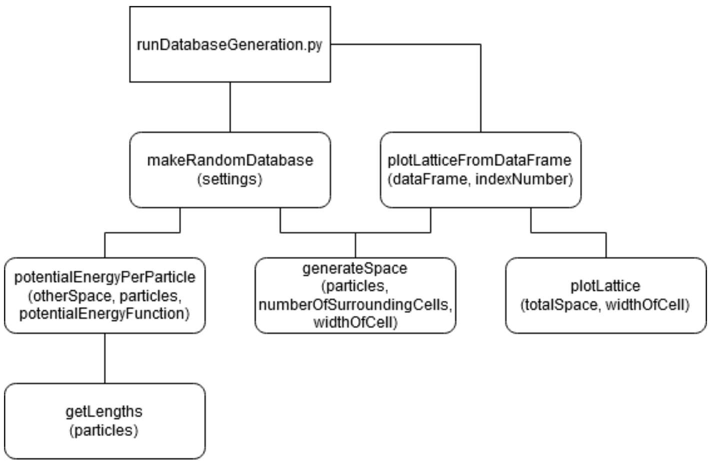
Property	**Python** [2]	**Java** [1]	**MATLAB** [6]
Speed of development and readability	+++	++	+
Amount of (relevant) modules	+++	+	++
Speed of execution	-	++	++
Popularity in science	+++	−	+
Accessibility	+++	+	—
Experience by the author	++	+	-

MATLAB was dismissed because it is an expensive proprietary programming languages and thus can't be run by everybody who might be interested. Java was dismissed because, while it has a large speed advantage over Python, the readability, development time, popularity and relevant modules are worse. So Python was picked which also had the advantage that it is used very much for machine learning which of course is the final use case of such a database.

The structure of the script written can be seen in figure 9. It is started by running the "runDatabaseGeneration.py" script which calls other functions. The starting script contains three parts. First, it has a part in which the user can specify the settings of the database they want to generate for example the amount of data points and which energy function to use (see appendix C.1 for instructions and details on how to operate the script). Next, it generates the database, and finally it tries to analyse and visualise the database.

The function that generates a random database contains several for loops. The first one is to vary the width of the cell. For each cell width, the particles within the cell are generated for each data point by first generating all the x coordinates, then the y coordinates, etc. This is done to prevent patterns from forming the in the data [8]. Next, it is going to refine and calculate the potential energy of each of these data points. Since if the particles are just randomly placed across the space ,some of the energies will be very high and that is physically unlikely. To reduce this, a relaxing process will be applied. It does this by applying a similar principle to gradient decent to the particle position. For a number of loops, each particle is allowed to move in a random direction with a random amount of magnitude (within a specified range). If the energy lower than the original configuration the change in position of the particle is accepted if not the changes will be reverted. Optimal settings for this relaxation process will be discussed in chapter 4.2.1. After the relaxation process is done, the potential energy in calculated in a more accurate manner. To then make sure no extreme outliers are incorporated into the database, a certain percentage of the top energies is deleted. This percentage shouldn't be too high since otherwise it will result in significant amounts of non-outliers being deleted.

Once the database has been generated, it is saved to a .json file. TThe database is then statistically analysed, and distribution plots are made. After this, some data points can be inspected manually (the energies and corresponding lattices will be displayed).



runDatabaseGeneration.py
makeRandomDatabase (settings)
plotLatticeFromDataFrame (dataFrame, indexNumber)
potentialEnergyPerParticle (otherSpace, particles, potentialEnergyFunction)
generateSpace (particles, numberOfSurroundingCells, widthOfCell)
plotLattice (totalSpace, widthOfCell)
getLengths (particles)

Figure 9: A diagram of the code that generates the database. Some of the simple supporting functions and were left out. The program is started by running "runDatabaseGeneration.py" which also contains the settings that can be adjusted and which launches all the functions which generate, save and visualize the database.

3.3 Preparing for machine learning

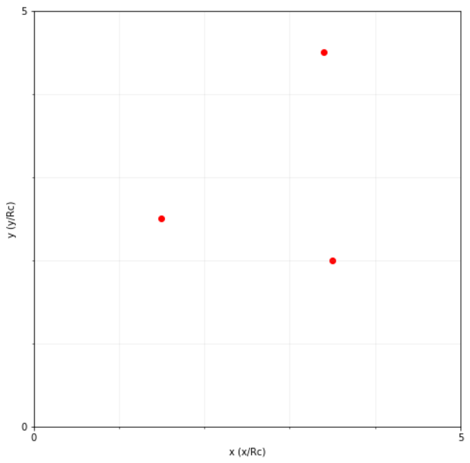
Once such a database has been made, it needs to be prepared for machine learning. In theory, a neural network can be trained on only the raw positional data to target the potential energy per particle, but this will lead to bad results since the problem is translation and permutation symmetric, and such a network will have to learn these symmetries itself. It is better to help the network by making the input symmetric, and thus, the network not having to take that into account.

The first step that can be taken is to use the relative distances between particles, but sadly a problem is immediately discovered. This is not permutation symmetric (if two particles are swapped the relative distances in the input are also swapped). To solve this the relative distances can be ordered from small to large (or vice versa). This has the problem that now the input of the neural network (and thus also the output) is not continuously differentiable. For example two relative distances could both be changing, one is decreasing but the other is increasing, then once the values are equal their places will swap in the input in a continuous way, but since their deviates are not equal a discontinuity will take place. This can be solved by instead first constructing a matrix of their relative distances and then taking the eigenvalues of that matrix [11]. Thanks to the avoided crossing theorem, these values are unlikely to cross [10] and thus, the input will remain continuously differentiable. The final problem is that for each particle you will only get one eigenvalue while the analytical input has two or three inputs (depending on the amount of dimensions), thus the neural network needs more inputs. This can again be solved by making multiple matrices where each time the relative distance is taken to a certain power. The minimum amount of these matrices is equal to the dimension of the database (but creating redundancy by having more such matrices is also possible and might even be beneficial).

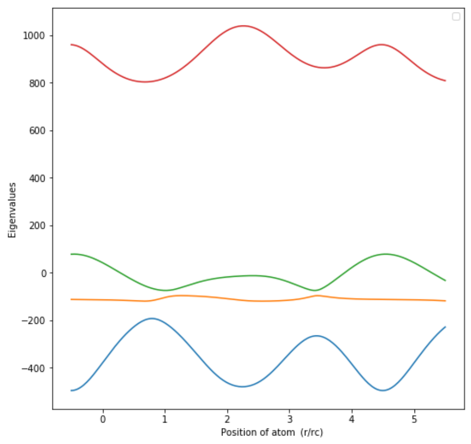
The algorithm described above is implemented slightly differently to make sure it converges and to take into account effects of the surrounding cells (and thus can also derive the width of the cell from the input). The matrices are referred to as proximity matrices. Each of the individual matrix elements are

$$
\begin{aligned}
P^o_{ij} &= \sum_{k}^{(2d_{sp}+1)^d} \left(\frac{R_{0p}}{r_{ijk}} - \frac{r_{ijk}}{R_{0p}}\right)^{-o} && \text{for } r_{ijk} < R_{0p} \\
P^o_{ij} &= 0 && \text{for } r_{ijk} \geq R_{0p} \text{ or } i = j \text{ and } k = 0
\end{aligned}
\tag{11}
$$

where R_{0p} is a distance after which the contribution becomes zero, r_{ijk} is the relative distance between particle i of the central cell to particle j of cell k, o is the order of the proximity matrix and d_{sp} is the depth of the surrounding cells needed for pre-processing (this is roundup($\frac{R_{0p}}{w_c}$) + 1). If i and j are equal the contribution of $k = 0$ is not taken into account this is because this will result in a division by zero. To make sure the input is continuously differentiable, o must be smaller than or equal to -2. Next, to make sure the input is also permutation symmetric, the eigenvalues of each matrix will be sorted (be careful not to sort the combined eigenvalues of multiple matrices because then you will get crossings). R_{0p} can be as large as wanted (although a larger R_{0p} also means more calculations need to be done before it converges), but it is recommended to keep it at least double the maximal size of the cells in the database such at least one surrounding cell will be taken into account when calculating the eigenvalues. In figure 10, the eigenvalues are plotted as they change due to a particle being dragged trough the cell. As you can see the eigenvalues are continuous and don't cross.



(a) The cell trough which the particle is dragged.



(b) The eigenvalues of order -2 of the system as the particle is dragged from bottom to top at $x = 2.5R_c$.

Figure 10: The results of particle being dragged trough a cell.

4 Results

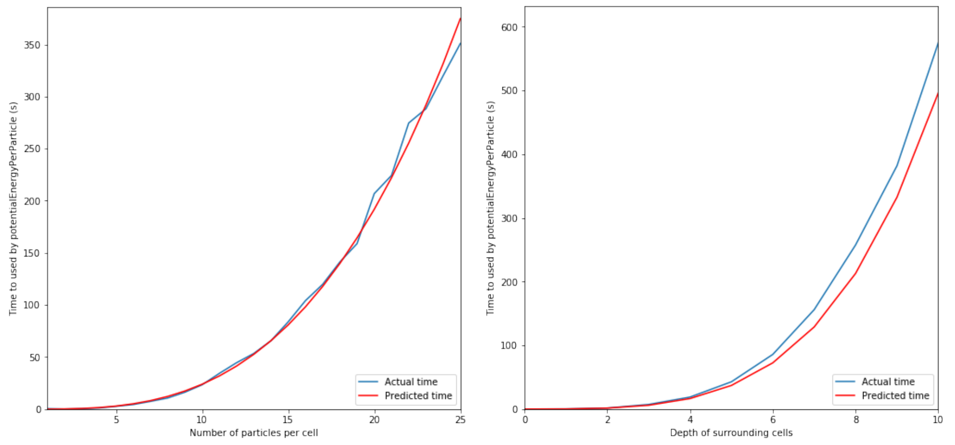
4.1 Algorithm

One of the most important aspects of the algorithm is the speed. The speed of a single potential energy calculates scales like

$$\mathcal{O}\left(d_s^d N_p^2\right) + \mathcal{O}\left(d_s^{2d} N_p^3\right) \tag{12}$$

where the firs term is due to two-particle interactions and the second therm is due to triplet interactions. The complete analysis of the time scaling can be seen in appendix B. This is faster than the $\mathcal{O}(N^3)$ of a DFT simulation (where N is the total number of particles) [13], especially if the depth of the surrounding cells is kept low (which can be done do to the way the potential energy is defined). For example an DFT simulation of around 20 particles takes around 10 minutes while with this toy model, it takes around 135 seconds [12] (although it is important to note that this was done on different hardware around 3 years older than the system these calculations are done on).

In figure 11, the scaling time for differing number of particles per cell and depth of surrounding cells can be seen together with the predicted calculation time. As can be clearly seen, increasing the depth of surrounding cells has a much more significant effect than increasing the number of particles. Also, while not perfect, the predictions match the results reasonably well. The difference might be explained due to the inaccuracy of the time per triangle/pair or the other parts of the program that also take time but are not taken into account when predicting the calculation time.



(a) The calculation time for increasing number of particles per cell with $d_s = 2$ in 2D.

(b) The calculation time for a increasing depth of surrounding cells with $N_p = 4$ in 2D

Figure 11: The calculation time for differing number of particles per cell and depth of surrounding cells with the predicted time.

The speed at which the preparations of the database can be done is also important since that ultimately decides how much faster the calculations with the trained neural network are compared to DFT simulations. The construction of the proximity matrix scales like $\mathcal{O}(d_{sp}^d N_p^2)$ (since the size of the matrix scales quadratically with the amount of particles per cell) and the calculation of the eigenvalues scales like $\mathcal{O}(N_p^3)$, although depending on the method used this can be slightly lower (in this case a general numpy algorithm was used). This means the total preparation scales like $\mathcal{O}(d(d_{sp}^d N_p^2 + N_p^3))$ (the factor d is added since for each extra dimension minimally one extra set of eigenvalues is needed). Which term dominates depends on which how efficient each process itself is and on what the value of N_p is (for example a process which consists of N^2 which take 20 seconds will be faster than a process which consist of N steps which take 1 second as long as $N < 20$).

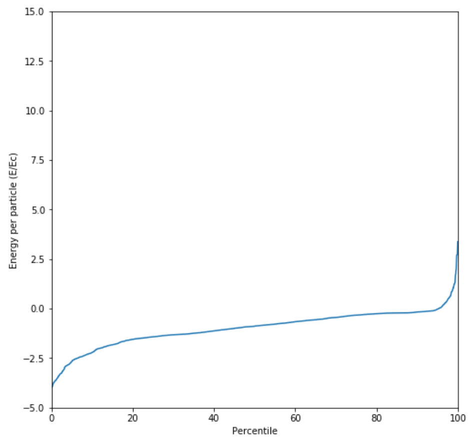
To see if it is actually faster the preparation of a cell with 20 atoms was measured. This took 0.53 ± 0.06 seconds which is very low. The other factor is of course the neural network and this scales like $\mathcal{O}(dN_p)$ and, also depends on the size of the network but that part generally is very fast.

4.2 Databases

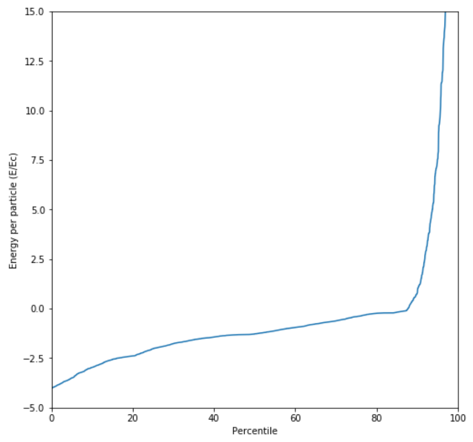
To judge the quality of the algorithm and potential energy function several, databases were generated to analyse. This was done in 2D and with 4 particles per cell to keep it simple. Initially, this was also done for only one cell width of $5R_c$. For the depth of surrounding cells 3 was chosen for the final calculation and a depth of surrounding cells of 1 was chosen for the relaxation process.

4.2.1 Number of relaxation epochs

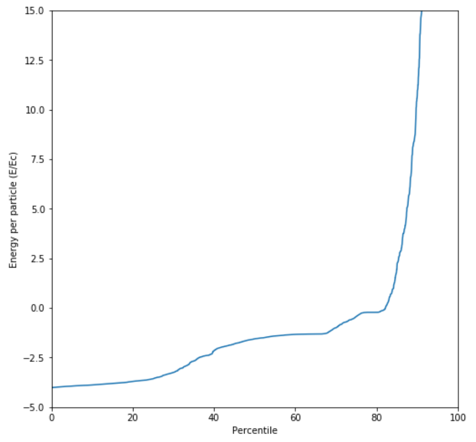
The first interesting thing that was investigated is what a good number of particle scans (number of times each particle is allowed to have the chance to move) for the relaxation process is. For this, several databases were generated which used different numbers of scans of relaxation (see figure 12).



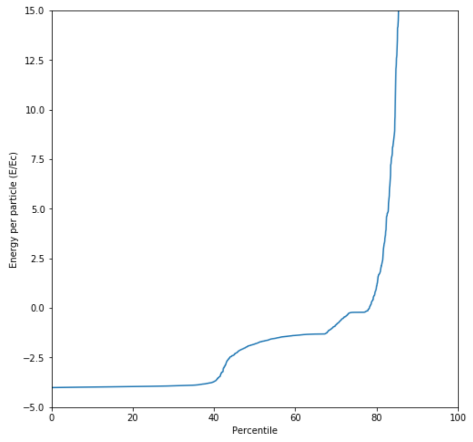
(a) 30 scans. Energy per particle reaches up to $3.7E_c$.



(b) 50 scans. Energy per particle reaches up to $125E_c$.



(c) 100 scans. Energy per particle reaches up to $1000E_c$

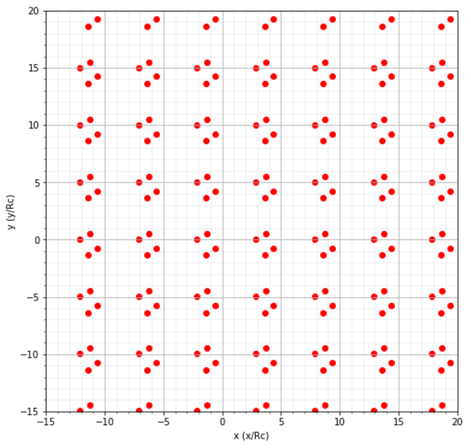


(d) 200 scans. Energy per particle reaches up to $20000E_c$.

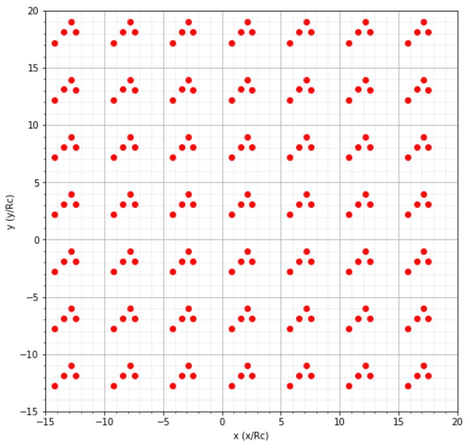
Figure 12: Percentile plots for databases with different amounts of scans of relaxation.

As can be seen for a larger number of scans, the curve of the lower energies will be pulled 'flatter' and once the plots displaying the (final) lattice are investigated it can be seen why. For a larger number of scans, one can see a larger fraction of the data points have one of two structures. The first possibility is that they naturally gravitate towards a square configuration; it can also gravitate towards a triangle configuration or that two pairs form (see figure 13). Interestingly, as the amount of scans increase, the amount of square (and triangles) becomes larger as the triangles (and pairs) merge with the other particles.

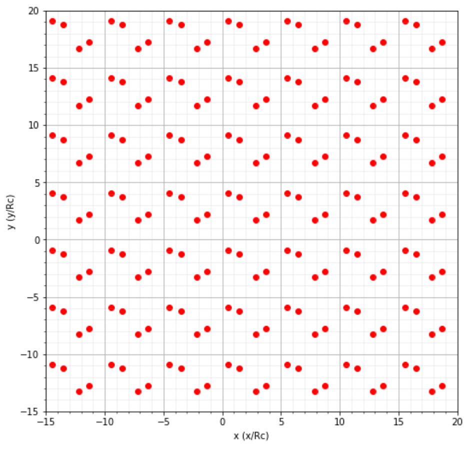
It is unknown why the higher energies also get more common as the number of scans increase.



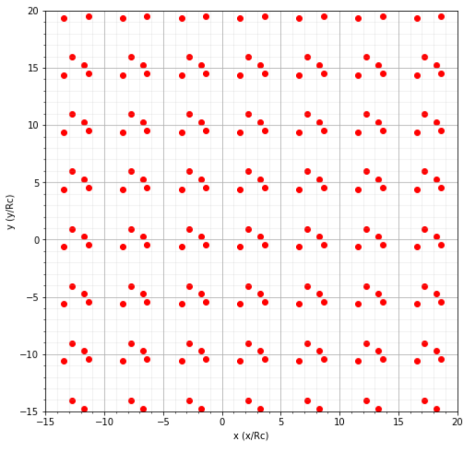
(a) A data point in which a squared formed due to the relaxation process.



(b) A data point in which a triangle formed due to the relaxation process. Note the fourth particle sitting close to the triangle but not forming a square.



(c) A data point in which a pairs formed due to the relaxation process.



(d) A configuration of particles which is no clear shape has formed but seems to be in the process of forming two pairs (which might merge into a square).

Figure 13: In a database different kind of structures are prominent depending on the amount of relaxation scans and what the maximum distance was that each particle could move.

A database which has a more diverse amount of shapes is beneficial for the neural net since then it also is adaptable for configurations which are not super common. One way to get a indication of how much this has occurred is the acceptance rate. This is the percentage of particles whose movement is accepted while the relaxation happens. Important to note that for large empty spaces the acceptance rate will be nearly 100% because a movement is also accepted if there is no change in potential and due to the build in convergence of the potential energy function if particles are far away (if a potential is used without a build in convergence a starting acceptance rate of around 50% will be seen). If for a large space, the acceptance rate is significantly lower than 100% that means that pair, triangle, or square must have formed. While there is no exact way to measure it, it was found that at an acceptance rate of around 20% for a width of $5R_c$ the database was reasonably diverse while also having a nice range of energies. This means that the amount of scans needs to be between 30 and 40. An example of such a wanted configuration can be seen in figure 13d as can be seen the four particles seem to be in the process of either forming two pairs or a square but they haven't completed their process yet (their energy is $0.29E_c$).

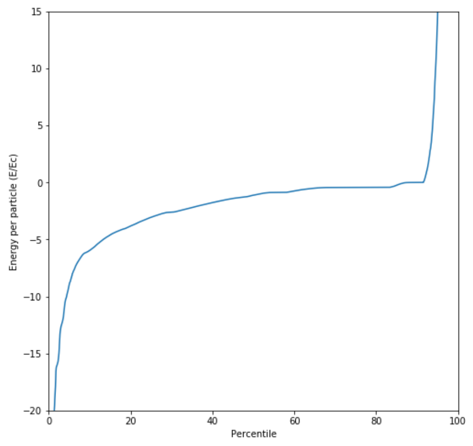
Table 2: The average acceptance rate of the final 10 epochs of the database generation for different amount of scans and cell widths. The maximum amount each particle could move per epoch was $0.1R_c$ each cell contained 4 particles.

Scans vs Width of cell	$2.5R_c$	$5R_c$	$10R_c$	$20R_C$
20	$25.6 \pm 4.3\%$	$32.0 \pm 4.7\%$	$66.9 \pm 4.7\%$	$91.8 \pm 2.7\%$
30	$22.4 \pm 4.2\%$	$23.6 \pm 4.2\%$	$64.9 \pm 4.8\%$	$89.6 \pm 3.0\%$
40	$15.8 \pm 3.9\%$	$18.6 \pm 3.9\%$	$60.3 \pm 4.9\%$	$85.2 \pm 3.6\%$
50	$13.5 \pm 3.4\%$	$16.1 \pm 3.7\%$	$58.1 \pm 4.9\%$	$91.3 \pm 2.8\%$
60	$9.8 \pm 2.9\%$	$12.3 \pm 3.3\%$	$59.2 \pm 4.9\%$	$89.6 \pm 3.1\%$
70	$8.6 \pm 2.8\%$	$13.3 \pm 2.8\%$	$55.9 \pm 4.9\%$	$84.2 \pm 3.6\%$
80	$7.6 \pm 2.6\%$	$13.7 \pm 3.4\%$	$52.8 \pm 5.0\%$	$87.9 \pm 3.2\%$
90	$8.7 \pm 2.8\%$	$10.2 \pm 3.1\%$	$52.1 \pm 5.0\%$	$84.3 \pm 3.6\%$
100	$6.1 \pm 2.4\%$	$9.2 \pm 2.9\%$	$49.1 \pm 5.3\%$	$85.2 \pm 3.5\%$
150	$4.2 \pm 2.0\%$	$4.4 \pm 2.1\%$	$47.4 \pm 4.9\%$	$84.3 \pm 3.6\%$
200	$2.9 \pm 1.7\%$	$3.1 \pm 1.7\%$	$40.9 \pm 4.9\%$	$76.6 \pm 4.2\%$
300	$1.5 \pm 1.2\%$	$2.7 \pm 1.6\%$	$37.3 \pm 4.8\%$	$72.6 \pm 4.4\%$
400	$1.5 \pm 1.2\%$	$0.8 \pm 0.9\%$	$29.8 \pm 4.5\%$	$65.6 \pm 4.8\%$
500	$0.7 \pm 0.8\%$	$0.6 \pm 0.8\%$	$27.8 \pm 4.6\%$	$52.2 \pm 6.4\%$

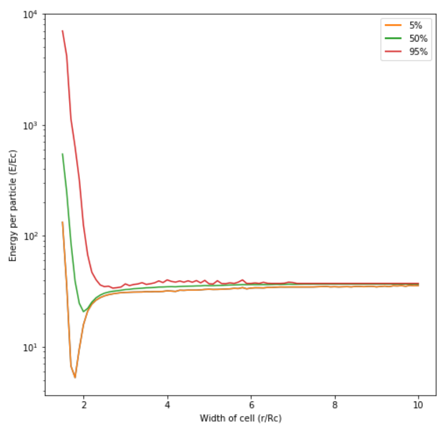
The number of scans needed to reach this ideal situation also varies based on how wide the cell is. Generally, this can be solved by either scaling the amount of scans with the width of the cell or by scaling the distance a particle is allowed to travel within one epoch. The disadvantage of scaling the number of scans is that larger cells take longer to calculate, and the disadvantage of scaling the distance the particle is allowed to travel is that accuracy is lost for larger cells. Thus the particle could potentially 'overshoot' a local minimum. It is important to note these guidelines don't necessarily apply to extreme situations such as when $w_c < R_c$ or $w_c \gg R_c$ because in those situations, relaxation might not be a effective tool at all. Also, this principle becomes less effective if a potential energy function which has build in convergence is used due to particles which are far away from other particles (and thus not influenced) not attracting each other (and only displaying random motion).

4.2.2 A database with a range of cell widths

A database contain a range of cell widths was also generated. For this, a cell width between $1.5R_c$ and $10R_c$ was used and 40 scans of relaxation with a maximum distance each particle could move per scans of $0.1R_c$. With this range of cell widths, both the hard core of the potential energy function and empty space can be simulated reasonably well. The resulting distributions are displayed in figure 14 and look good. Most of the energy is in a reasonable range (between $-5E_c$ and 0), but it also contains a few minor outliers which is just what is wanted in such a database. The distribution of energy as a function of the cell width is also as expected with a large peak for small cell widths, a minimum for a cell width of around $2R_C$ (it is slightly lower this is due to the desired distance for a square being slightly less than $1R_c$) and an asymptotic approach to zero for larger cell widths.



(a) The distribution of the energies in the entire database.



(b) A plot of the different energies in the database for different cells widths. It is a log plot (to include the high energies) with a ofsett of $-37.1E_c$.

Figure 14: The distributions of the energies in the database with different cell widths.

5 Machine learning

Finally, to test the databases generated in chapter 4.2 different neural networks will be trained and tested. For the making and training of the neural networks, the Keras module for Python was used. It was first attempted to train the network on a database containing a range of cell widths (like in chapter 4.2.2), but sadly, the author had little success in training a network on this dataset. The author has no doubts that such a network can be trained but more research is needed into it. Instead, a database only containing cells with a cell width of $5R_c$ was used (the database was created using a relaxation process of 40 scans and has 25000 data points of which 20% was used as validation data (see figure 15)).

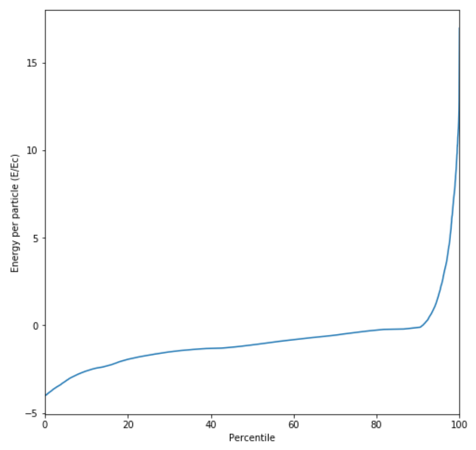


Figure 15: The distribution of energy in the database used for the training of the neural networks. It was made using a relaxation process of 40 scans and all cells have a width of $5R_c$.

To make sure the data is accurate, each type of network was trained at least 3 times to make sure the amount of luck involved was minimal. To investigate how good a fit is, several things will be inspected. First of all, the loss and validation loss will be inspected. Next, a scatter plot of the predicted energy per particle and the real energy per particle will compared. And finally a particle will be dragged through a cell to see how the potential energy surface behaves. The cell can be seen in figure 10.

The design of the networks was kept relatively simple. The most successful networks were linear networks, but networks with one or two hidden layers which had a width of one, two, three of four times the amount of inputs were also tried. For the initial part in which the general properties of the training were discovered, the linear network and single hidden layer were mostly used (but due to the similar results below only the results of the linear network are mentioned when determining the general properties of the network).

5.1 Loss and optimizers

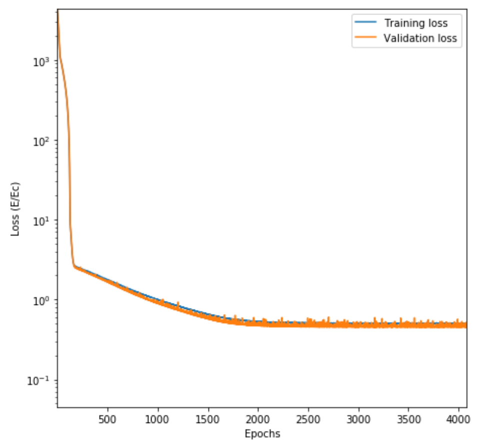
The error/loss function was decided to be the mean absolute error (MAE). This was done because for example the root mean square error (RMSE) places more emphasis on outliers and this is unwanted since the most important thing is that the predictions are good quality for the most common energies[7]. A percentage error was also not chosen since the thing we are interested in is the absolute value of the energy per particle and not the relative value (since the potential energy can be shifted shifted up or down without it changing how the configuration behaves). Next, the optimizers were tested. SGD, ADAM, NADAM, adagrad, rsmprop and adadelta were tried and, except for SGD, they all performed well and didn't show any major differences (except for the amount of epochs needed to reach the minimum). So instead, ADAM was picked because it is a popular, easy to use and modern optimizer [4].

The learning rate was also investigated and it was found that a lower learning rate is very beneficially for the results (although of course it results in a much longer training time). For example, training a linear network with a learning rate of 10^{-4} resulted in a loss of $2.5E_c$ while a learning rate of $5 \cdot 10^{-5}$ resulted in a loss of $1.1E_c$ and a learning rate of 10^{-5} resulted in a loss of $0.4E_c$ (all for a linear network using a order -2 and -3 input). While an even lower learning rate might have resulted in an even lower loss (assuming it is not limited by the architecture of the network or the size of the database), but sadly, it took to much time to train such networks. So a learning rate of 10^{-5} was picked.

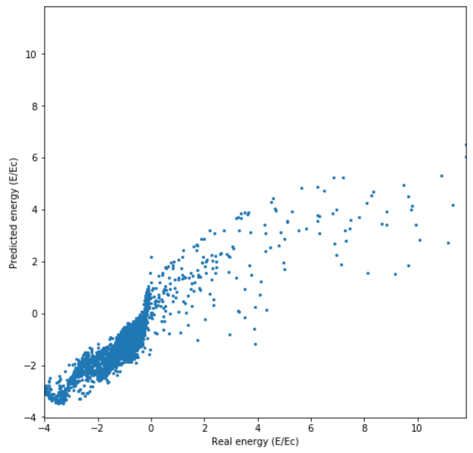
5.2 Inputs and architecture

One of the interesting questions is what kind of inputs should be used. At a minimum the same number of proximity matrices (and their associated eigenvalues) should be generated as there are dimensions but can perhaps more inputs (and thus having a over determined system) result in a better trained network? As a starting point, two inputs were used with order -2 and -3 (see chapter 3.3) and this resulted in a loss of $0.47E_c$ (see figure 16). These results are already pretty acceptable especially when looking at the gradient of the particle dragged trough the cell (although it does struggle at some of points).

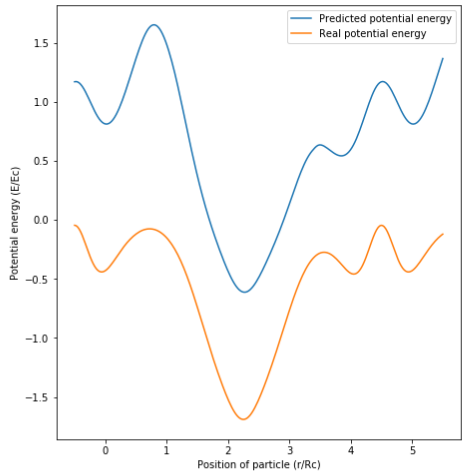
First, more (and thus higher order) inputs were tried, but sadly these made the results worse. A network with inputs of order -2, -3 and -4 resulted in a loss of $1.6E_c$, and a network with inputs of order -2, -3, -4 and -5 resulted in a loss of $7.8E_c$. Next, only higher inputs were tried, but while these results were better than more inputs, they were still bad. A input of order -3 and -4 resulted in a loss of $0.7E_c$ and a input of order -4 and -5 resulted in a loss of $1.4E_c$. So both more and higher orders of the input resulted in worse networks. This is likely because a higher order input results in more extreme values which is hard for the network to process. When adding more inputs, the network likely struggles with the over determination.



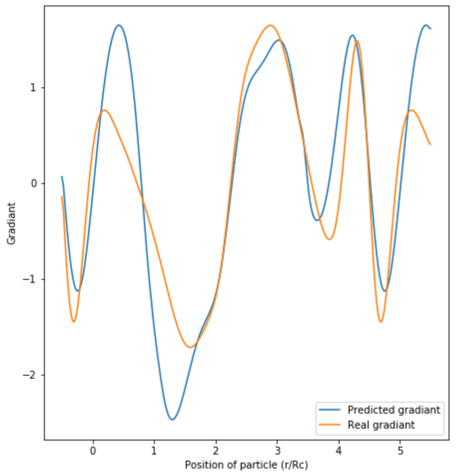
(a) The loss during the training of the network.



(b) A scatter plot of the predicted energies per particle vs real energies per particle.



(c) The energy per particle as the fourth particle moves trough the cell.



(d) The gradient of the energy per particle as the fourth particle moves trough the cell.

Figure 16: The results of the test for a linear network with bias parameters trained on inputs of order -2 and -3 with a learning rate of 10^{-5}.

When adding the hidden dense layers, softplus activation layers was used since those are the most effective for potential energy surfaces [11]. These hidden layers had the width equal to one, two, three or four times the input size. When adding a hidden layer (with bias) with the width equal to the amount of inputs, the results were nearly identical to a linear network, it had a loss of $0.44E_c$, had a similar scatter plot, displayed a offset for the drag test and also had one two points at which the gradients didn't match. When making the hidden layer wider the loss, scatter plot and drag test kept similar results but in the gradient large variations were occurring (see figure 17). These even seemed nearly discontinuous at times (although that was impossible due to the softplus activation layer). Replacing the single large hidden layer with two layers with a width equal to one times the input size had similar results.

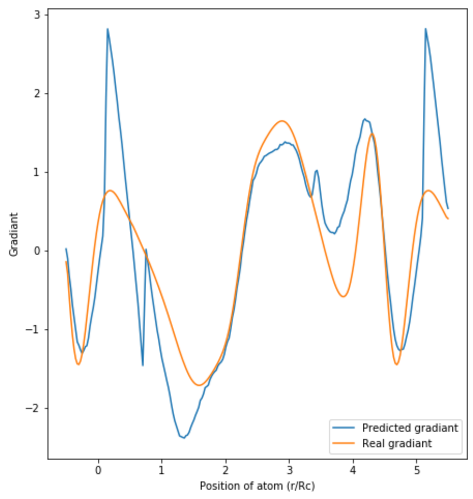


Figure 17: The gradient of the energy per particle as the fourth particles moves trough the cell of a network with one hidden layer of width 32 with inputs of order -2 and -3

So the best network that was made was the linear network with biases which took eigenvalues of proximity matrices with order -2 and -3. This is a simple and easily trainable network. It is also fast since it only takes 275 ± 2 μs to execute a prediction. The results are still not great since the loss is still around $0.47E_c$, but it is likely that with more effort (and perhaps more data points) a larger network with a higher accuracy can also be trained.

6 Conclusion and discussion

6.1 Conclusion

In this project, a successful toy model for generating databases for the purpose of hyper parameter optimisation of neural networks was developed. The toy model that was designed around square/cubic cells containing a certain number of particles with periodic boundary conditions. The energy per particle calculated is based on a sum of potentials associated with each possible pair or triplet that has at least one particle in the central unit cell. Using only pairs/triplets that have at least one particle in the unit cell greatly reduces the computations needed to calculate the energy per particles since this measure reduces the amount of pairs and triangles that need to be calculated and reduces the amount of surrounding cells because it converges much faster.

The potential energy function of each pair/triplet is calculated using a modified Lennard-Jones potential. It contains a characterised energy and distance (which is equal to the energy and distance of the minimum) which makes it more generally applicable. This function meets the requirements set. It is continuously differentiable, combines short-range repulsion with long-range attraction, and is more complex than two particle interaction. It also has built in convergence which has the advantage that the amount of compute is greatly reduced, but sadly does make it harder for the software to relax the configuration.

The relaxation process is a feature the software can do which reduces the extreme energies in the database. For this, a acceptance rate of around 20% seemed to produce the best database containing a large range of energies and configurations. A higher amount of particle scans which results in a higher acceptance rate will result in certain kind of configurations becoming very prevalent in the database. When inspecting the database produced with various cell widths, one can also nicely see that once the width of the cell becomes too small, only higher energies will occur due to the particles being in each others 'hard cores'. One can also see a nice minimum as all particles have just enough space to be at optimal distance from each other.

The software written is significantly faster than DFT software. This is because it is a relatively simple analytical function. For example, a data point with 20 particles takes around 10 minutes using DFT software but only 135 seconds using the toy model alternative. It also scales better being only $\mathcal{O}\left(d_s^d N_p^2\right) + \mathcal{O}\left(d_s^{2d} N_p^3\right)$ where d_s is the depth of the surrounding cells and N_p is the number of particles in the unit cell. Of course, this software can't replace DFT simulations for the training of the final network, but it can better carry out research into what the optimal settings for a neural network and database are.

With the database that has been generated machine learning can also be done, but for this the data needs to be prepared, which is done by generating multiple proximity matrices in which a sum of proximities is made. For the proximity, a function was designed which depends on the relative distance between two particles but which also has build in convergence. The eigenvalues of these matrices are then calculated and these satisfy the needs for our inputs. They are continuous, continuously differentiable and transnational and permutation symmetric. It also is extremely fast scaling like $\mathcal{O}(d(d_{sp}^d N_p^2 + N_p^3))$ where d are the amount of dimensions and d_{sp} is the depth of the surrounding cells. For example, when preparing a data point containing 20 particles it only took 0.53 ± 0.06 seconds.

The design of the neural networks has been less successful. A working network which was trained for multiple widths sadly couldn't be found but when only training for a single width, more success was seen. The ADAM optimizer was chosen (although except for SGD every optimizer that was tried performed fine). It was found that a lower learning rate can greatly increase the accuracy of the model. It is unknown what the lower effective limit is but it is at least a learning rate of 10^{-5}. It was also found that having less inputs and having inputs from a proximity matrix with a low order is also better than having more inputs from higher order proximity matrices. A simple linear network with biases already produced remarkably acceptable results having only a loss of $0.47E_c$ (around half a characteristic energy unit) and doing very well in a test were the gradient of the neural network was compared to the gradient of the real potential. Sadly, the results of this couldn't be improved by adding hidden layers and in fact, sometimes adding layers made it worse. It is very likely that, just like making a network with a range of cell widths, it is possible to achieve this.

6.2 Discussion and future research

The scope of this thesis was initially focused more on machine learning than on designing a toy model. As the latter turned out to be harder than predicted, the focus of this thesis shifted towards creating the framework for the toy model. One of the obvious things that can be done as future research is to focus more on the machine learning with these methods for generating a database for training.

For such research, the method for prepossessing the input could be examined and perhaps improved. But also, the best way to train the neural networks should be a focus area. The size of the database required for successful training could be investigated. The problem of a lower learning rate not being feasible could also be solved. A possible method to solving this problem might be a system which adapts the learning rate while the training is happening (thus first having a higher learning rate and then lowering it once the loss has stagnated). The architecture of the neural networks can also be investigated and networks with more and wider layers could be attempted. Finally, using multiple networks might also be a option.

Regarding the software used to generate the databases one mistake that was made here was the choice of Python as a programming language. Because, while it certainly is a versatile language and easy to understand and write, it is not very quick (which was one of the main goals); thus, reimplementing it in another faster language is a option that could be explored. A less work intensive method might be to use a Python to machine code compiler.

What also could be investigated is if a potential energy function that doesn't have build in convergence might work better when relaxing the database since that is one of the large disadvantages of the modified Lennard-Jones potential. An alternative method for doing the relaxation could also be implemented which doesn't suffer from this problem.

Of course, once the optimal hyper parameters for a neural network trained on the databases produced by this toy model have been found, it would be interesting to see how well these settings translate to a neural network that will be trained on data from DFT simulations.

A Potential energy construction

A.1 General method

Below, there is a example of how to make a potential that satisfies the needs specified in chapter 3.1. For the mathematical operations, Wolfram Mathematica will be used. As a example the potential with short range depending on the area and polynomials and the long range depending on the circumference of the triangle will be used.

First you define the general shape and characteristics of your potential. Multiply the short range potential by V_{sc} (short range potential correction) and the total potential by M_c (magnitude correction):

```
s = (a + b + c) / 2;
area2 = s * (s - a) * (s - b) * (s - c);
Vshort = area2 * (a^-12 + b^-12 + c^-12);
Vlong = -1 / (a + b + c)^6;
Vtotal = Mc * (Vlong + Vsc * Vshort);
```

Here, a, b and c are the lengths of the triangle. Next, calculate the potential of a equilateral triangle and calculate for what distance of the sides it has its minimum. Determine for which value of V_{sc} that is with sides of size 1:

```
Vtriangle = Vtotal/.{a->r, b->r, c->r};
minR = r/.Solve[D[Vtriangle, r]==0, r][[2]][[1]];
(*The brackets are to select the correct solution*)
Vsc = Vsc/.Solve[minR==1, Vsc][[1]][[1]]
(*The brackets are for the same reason*)
```

In this case, $V_{sc} = \frac{4}{2187}$. So now that the minimum is in the right place it needs to be made the right size so next M_c is determined as so that the minimum is at -1 and then make a nice plot to verify the result.

```
Mc = Mc/.Solve[(Vtriangle/.{Vsc->Vsc})==-1, Mc][[1]][[1]]
Plot[Vtriangle/.{Vsc->Vsc, Mc->Mc}, {r, 0, 4}, PlotRange->{{0, 4}, {-2, 2}}]
```

Which results in a value of $M_c = 2916$ and a plot as seen in figure 18.

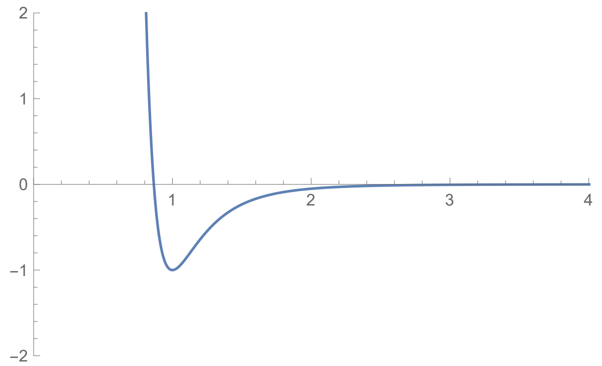


Figure 18: The potential energy as a function of distance in a equilateral triangle.

Finally, you can construct the final equation and simplify it (see figure (13)). Pay attention that you divide each length by R_c and multiply each term with E_c to make sure the units are correct.

$$V_3 = V_s + V_l, \quad V_s = \frac{16E_c}{3}\frac{A^2}{R_c^4}(\frac{R_c^{12}}{a^{12}} + \frac{R_c^{12}}{b^{12}} + \frac{R_c^{12}}{c^{12}}), \quad V_l = \frac{-2916E_c}{(\frac{a}{R_c} + \frac{b}{R_c} + \frac{c}{R_c})^6} \tag{13}$$

A.2 Modified Lennard-Jones potential

The potential that we want to construct in chapter 3.1.4 has the extra requirement that it needs to have build in convergence and that the other properties still need to be present (continuous, continuously differentiable, has a minimum at $-E_c$ and R_c, etc).

To construct the modified Lennard-Jones potential, we start with the Lennard-Jones potential as seen in chapter 2

$$V = E_c\left(\frac{R_c^{2n}}{s^{2n}} - 2\frac{R_c^n}{s^n}\right)$$

this is then adjusted such that at R_0 the potential becomes zero.

$$\begin{aligned} V &= E_c\left(\left(\frac{R_c^{2n}}{s^{2n}} - \frac{R_c^{2n}}{R_0^{2n}}\right) - 2\left(\frac{R_c^n}{s^n} - \frac{R_c^n}{R_0^n}\right)\right) && \text{for } s < R_0 \\ V &= 0 && \text{for } s \geq R_0 \end{aligned}$$

Now, the minimum is not exactly at $-E_c$ anymore but if

$$\frac{R_c^N}{s^n} = \frac{R_c^n}{R_c^n} = 1 \gg \frac{R_c^n}{R_0^n} \longrightarrow \frac{R_0^n}{R_c^n} \gg 1$$

then the minimum will also be close to $-E_c$ (for $n = 6$ it will be around $-0.96E_c$ which is good enough).

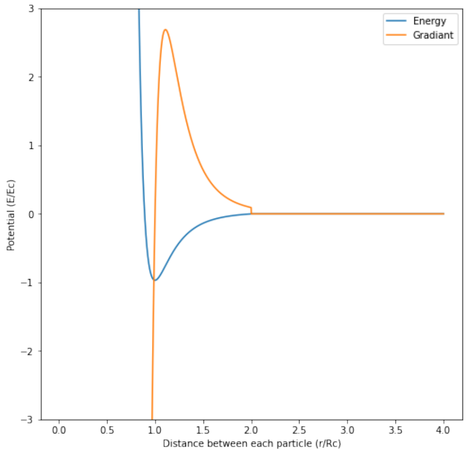


Figure 19: The Lennard-Jones potential but only adjusted to make it zero at R_0 (in this case $R_0 = 2R_c$.

this equation is continuous and differentiable but not continuously differentiable at R_0 (see figure 19) thus a correction term will have to added to make sure it is. This correction needs to be zero at R_0 but have gradient of opposite magnitude than the current function. For this a simple linear function can be used

$$V_c = -\frac{\partial(V_s + V_l)}{\partial s}|_{s=R_0}(s - R_0) = -E_c\left(\frac{2nR_c^n}{R_0^{n+1}} - \frac{2nR_c^{2n}}{R_0^{2n+1}}\right)(s - R_0) = \frac{2nE_c}{R_0^{n+1}}\left(1 - \frac{R_c^n}{R_0^n}\right)(s - R_0)$$

which is the correction term that can be used to make the total modified Lennard-Jones potential energy as seen in chapter 3.1.4. This correction term also slightly changes the height of the minimum, but as long as $R_0^n \gg 1$, it will be close to zero around $s = R_c$. Again, as a example, for $n = 6$ and $R_0 = 2R_c$, it will be around $-0.88E_0$, which is close enough.

B Algorithm speed

The amount of time the algorithm takes depends on the number of triangles and pairs a potential energy needs to be calculated for. The amount of triangles present in a calculations are:

$$N_{triangle} = (\sum_{i}^{N_p} \sum_{j,k\neq j}^{((2d_s+1)^d-1)N_p} 1) + (\sum_{i,j\neq i}^{N_p} \sum_{k}^{(2d_s+1)^d-1)N_p} 1) + (\sum_{i,j\neq i,k\neq j\neq i}^{N_p} 1)$$
$$= ((N_p-2)(N_p-1)Np + ((1+2Ds)^d-1)(Np-1)Np^2 + ((1+2Ds)^d-1)Np^2(((1+2Ds)^d-1)Np-1))$$

and the number of pairs is

$$N_{pairs}(\sum_{i}^{N_p} \sum_{j}^{((2d_s+1)^d-1)N_p} 1) + (\sum_{i,j\neq i}^{N_p} 1)$$
$$= ((Np-1)Np + ((1+2Ds)^d-1)Np^2)$$

this means means that for a single potential energy calcualtions the predicted time it will take is

$$t_{predict} = t_{triangle}N_{triangle} + t_{pair}N_{pairs}$$

where $t_{triangle}$ is the time required for a single triangle to calculate (on my PC 40 μs) and t_{pair} is the time required for a single pair to calculate (on my PC 30 μs). With this equation the total time a database takes to generate can be predicted.

C Code usage

The code, several databases and a few trained networks can be found on the Github page of this thesis `https://github.com/TimHeiszwolf/Quantum_energy_ML`. More details can be found on the page and in the section below. If afterwards you still have questions feel free to contact the author.

The program consist of three parts. The first part generates a database (see chapter 3.2 and appendix C.1) which is outputted to a .JSON file. The next part prepares that database for machine learning (see chapter 3.3 and appendix C.2). Finally a Jupyter Notebook can be opened. In the notebook several methods for generating neural networks are already implemented and examples on how to use them are displayed.

C.1 Database generation

The function 'makeRandomDatabaseMinimum' from 'makeRandomDatabase.py' is the hart of the database generation. But controlling it directly is not very nice (lot's of inputs and very little feedback). So a script called 'runDatabaseGeneration.py' was made in which the settings can easily be adjusted and from which some feedback and analysis of the database that is produced is done.

The script produced a database (according to the method of chapter 3.2, as a pandas DataFrame) and then saves it as a .json file which can then easily be imported as a pandas DataFrame again. The DataFrame contains details like what the particle coordinates are, what the width of the cell of each data point is, what the potential energy of the data point is and with how many depth of cells it is created (in databases generated using older versions of the software this column might be names 'numberOfSurroundingCells').

The available settings are:

- numberOfDatapoints: The number of data points the database will approximately have (since splitting the work on multiple cores or compensating for the filtrating can result in some roundup errors).
- numberOfParticlesPerCell (N_p): The number of particles placed in each cell.
- numberOfDimensions (d): The number of dimensions the space in the database has. The generation process has been tested for 2D and 3D but should also work for higher dimensions.
- potentialEnergyFunction: Is the function used to calculate the potential energy of each pair/triplet. The function must take a single array of number (the relative distance between each of the particles).
- widthOfCell (w_c): Is a list of size 2 containing a lower and a upper limit for the size of the cell.
- numberOfWidths: The number of steps widths represented in the database (including the lower and upper endpoints).
- depthOfSurroundingCells (d_s): The depth of the number of cells taken into account when calculating the potential energy. This needs to be high enough such that convergence has taken place (for each possible cell size).
- relaxationDepthOfSurroundingCells: The depth of the number of cells taken into account when calculating the potential energy for the relaxation process. It is best to make this values lower than numberOfWidths since accuracy is less important and speed is more important.
- amountOfScans: Is the amount of scans the relaxation process is applied. During a scan every particle is allowed to move in a random direction and the change is accepted if the new potential energy is lower than the original.
- maxDeltaPerScan: Is the maximum magnitude of the movement of a particle during a scan. The actual amount is random between this and 0.
- cutoff: Is above which percentile the energies will be cuttoff and the configurations won't be taken into the database.
- filename: Is the name of the .json while to which the database will be saved (without the file extension).

To generate a database the 'runDatabaseGeneration.py' script should be executed (using Python). When it ask how many processes should start the user should give the amount of (logical) threads their processor has for the fastest preparation. While running the generation process you will receive updates about each data point and receive statistics about at the end of the generation process. This includes raw statistics but also distribution plots and samples of raw data.

C.2 Database preparation

The part of the code that prepares the database for machine learning has a similar structure to the code that generates the database. The 'prepareDatabaseForMachineLearning' in 'prepareDatabaseForMachineLearning.py' is the hart of the software and does all the work while the script in 'runDatabasePreperation.py' has the adjustable settings and starts the software.

It loads the database that was generated and then adds two columns. First the eigenvalues of the proximity matrices (as specified in chapter 3.3), the second are the raw relative distances between particles within the central cell (this was used for training in the early versions of the software but generaly doesn't work very well). This part is very fast.

The available settings are:

- filename: The name of the database that needs to be prepared (without the file extension). This needs to be a .json file of a saved pandas DataFrame.
- orderOfMatrix: Is a list of the orders which the proximity matrices should have.
- R0p (R_{0p}): Is the distance at which the contributions of the proximity matrix should become zero (see chapter 3.3). It also determined how much depth of the surrounding cells is going to be used when calculating the proximity matrices.

To prepare a database the 'runDatabasePreperation.py' script should be run. When it ask how many processes should start the user should give the amount of (logical) threads their processor has for the fastest preparation. The resulting prepared database will then be saved as a .json file again (as the same name but with 'Prepared' added to it's name). When giving the name of a database that has already been prepared the eigenvalue of relative distance columns will be overwritten during the new preparation.

C.3 Jupyter notebooks (for machine learning)

There are also two Jupyter notebook available. First is the 'testingPotential' in which the quality of potentials can be investigated with various graphs and tests.

The other (more important) notebook is the 'machineLearning' notebook. In this notebook several functions for making, training and investigating neural networks are displayed. Also methods for importing the data and preparing it for use in the neural network is given.

The important functions are 'plotHistory' which takes the history of a fit and makes nice plots of it, 'makeGradiantPlot' makes the plots for the test of dragging the particle trough the cell, 'makePredictionPlot' makes the scatter plot in which the predicted potential energy per particle is compared to the real energy per particle and finally 'plotAndPredict' takes a random sample for the validation data and plots the lattice for it and gives the prediction and real value of the energy per particle.

References

[1] Altexsoft. The good and the bad of java programming, August 2018. Available at https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/ (2020-06-20).

[2] M. Basel. Python pros and cons: What are the benefits and downsides of the programming language, June 2018. Available at https://www.netguru.com/blog/python-pros-and-cons (2020-06-20).

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

[4] P. Cheng. Quick notes on how to choose optimizer in keras, February 2018. Available at https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/ (2020-07-01).

[5] D. Jeffrey Griffiths and D.F. Schroeter. *Introduction to quantum mechanics*. Cambridge University Press, 2019.

[6] JavaTPoint. Advantages and disadvantages of matlab programming language, February 2017. Available at https://www.javatpoint.com/advantages-and-disadvantages-of-matlab (2020-06-20).

[7] J.M.V.A. Koelman. From linear regression to deep learning. TU/e DIFFER, 4 2019. For the course 'Machine Learning for Physicists 3EBX0' of TU/e.

[8] Pierre Lecuyer. Uniform random number generation. *Annals of Operations Research*, 53(1):77–120, December 1994.

[9] R. LeSar. *Introduction to Computational Materials Science*. Cambridge University Press, 1 edition, 2013.

[10] A.D. McNaught and A. Wilkinson. Avoided crossing of potential-energy surfaces, February 2014. Available at http://goldbook.iupac.org/terms/view/A00544 (2020-06-260).

[11] R.J.H. Mollevanger. Inter-elemental transferable potentials: Predicting energies using deep learning. Master's thesis, TU/e CCER, Eindhoven, NL, March 2020.

[12] P. J. Pedro. Quantum chemistry speed test, January 2014. Available at https://github.com/PedroJSilva/qmspeedtest (2020-07-01).

[13] N. Schuch and F. Verstraete. Computational complexity of interacting electrons and fundamental limitations of density functional theory. *Nature Physics*, 5(10):732–735, 2009.

[14] M. Sipser. *Introduction to the Theory of Computation*. Cambridge University Press, 3 edition, 2012.

[15] Y. Zuo, C. Chen, X. Li, Z. Deng, Y. Chen, J. Behler, G. Csányi, A. V. Shapeev, A. P. Thompson, M. A. Wood, and S. P. Ong. Performance and cost assessment of machine learning interatomic potentials. *The Journal of Physical Chemistry A*, 124(4):731–745, 2020. PMID: 31916773.