Modeling molecular trajectories using long short-term memory with stacked state pooling

Otto Lindfors

Master's thesis in computer science

Supervisors: Sepinoud Azimi Rashti, Sebastien Lafond

Faculty of Science and Engineering

Åbo Akademi University

June 2021

Abstract

Atomistic molecular dynamics can be used for simulating large molecular systems with great accuracy. The downside to this is that simulations are computationally expensive and thus take a long time to perform. Many times, one is only interested in a subset, or a summarizing statistic, of the information available in the results of the simulations. This raises the question whether it is necessary to run full molecular dynamics simulations when one is interested in only partial outputs or if similar results can be achieved with a cleverly designed machine learning algorithm. In this thesis, an attempt to answer this question is made by proposing a deep learning model for solving general many-body problems. The model generates long sequences of particle positions by predicting one time step at a time, using its previous outputs as new inputs. It is demonstrated that this model is capable of running simple molecular dynamics. Suggestions for further experiments to measure the capability of the model to generalize to complex many-body problems are presented.

Keywords: CNN, deep learning, Gaussian mixture model, LSTM, machine learning, molecular dynamics, probabilistic model, RNN, sequence-to-sequence learning, social-LSTM, stacked state pooling

Contents

1 Introduction

Molecular dynamics is a collection of methods for simulating the motion of interacting particles. The particle motion is simulated by numerically solving Newton's equations of motion in order to obtain trajectories as the solution. This is done by numerically integrating the equations of motions with respect to time and approximating the forces to be constant over short time intervals Δt, where Δt is the step size used in the numerical integration. A smaller step size will lead to a smaller error and in the limit of $\Delta t \to 0$ one obtains the true integral. Therefore, a sufficiently small step size must be chosen when performing molecular dynamics simulations. This means that a large number of calculations must be done in order to simulate a relatively short period of time. Furthermore, when the simulated system contain many particles the total number of calculations is further increased. Therefore, simulations of large molecular structures are computationally very expensive.

The extended aim of the work in this thesis is to develop a machine learning based method for performing computationally less expensive molecular dynamics simulations than conventional methods but still having sufficient precision for calculating accurate summarizing statistics on the results. The scope of this thesis is limited to demonstrating the viability of the approach in basic many-body problems. It is shown that the model is able to perform reasonably well even when being trained on solutions (trajectories) with sub-optimal sampling rates.

Two approaches are taken. The first approach, presented in section 3, attempts to model the surface molecules in a large drug carrying nanoparticle, consisting of 40 000 atom, using a naïve LSTM based neural network. However, this turned out to be a challenge and the particle trajectories were unsuccessfully modeled. In addressing the shortcomings of the first approach, a second approach where a sophisticated model inspired by the works of [1] and [2] is developed with the specific task of modeling particle-particle interactions in mind. The model generates a sequence of predictions one step at a time, always using the previous prediction as new input. At each time step, the particle interactions are modeled on a per-particle basis. The approach has

similarities with molecular dynamics although being fundamentally different in that no rules for the interactions and position predictions are given. Instead these rules are learned in a completely data-driven fashion.

In section 2 the various model components are presented in detail. In section 2.6 an approximation of inter-molecular electrostatic forces, central to some of the subsequent sections, is presented. Section 3 and 4 present the first and second experiments, respectively, and detailed descriptions of the specific model architectures and implementation details. Section 5 present the conclusions and suggested future work.

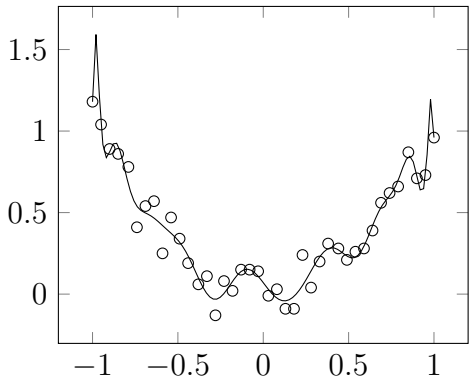
1.1 Previous Work

In previous work such as Wang et. al [3] and Pihlajamäki et. al. [4] a similar deep learning approach is taken. A deep learning model is used as a component for estimating the inter-atomic energies in popular molecular dynamics software in order to speed up computations. The work in this thesis uses similar ideas but approaches the problem by simplifying the simulated systems to only contain a partial description of the atomic structure and letting the deep learning model replace the entire molecular dynamics simulation. Further experiments are needed in order to tell whether or not this is a viable approach for general molecular dynamics problems but the initial proof of concept performed in this thesis shows promising results (see section 4.5).

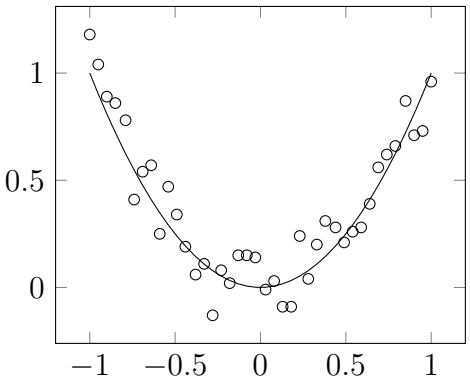
2 Theory

On the most general level, a neural network is a function f_θ parameterized by θ that maps values from some M-dimensional domain to some N-dimensional codomain, $f : \mathbb{R}^M \to \mathbb{R}^N$. Although the neural network in this thesis is defined to be real valued it is not a requirement. [5] Machine learning deals with problems that are too complex or too large to be feasibly solved analytically and are therefore approached using numerical methods. [6] In case of supervised learning, the problem is to find a function that describes some other observed function with as little error as possible by numerically fitting the neural network to samples of the observed function.

The neural network is fitted to the samples by maximizing the likelihood that the parameters θ correspond to the neural network f_θ, which best describes the samples. In practice, the neural network is optimized using a finite set of samples, a training set. Because the training set is finite, the difficulty lies in finding a function f_θ that correctly maps values from the domain to the codomain for values that are not present in the training set. This is referred to as generalization of the model. The goal is not to learn the specific samples that are present in the training set but instead to learn the underlying function or phenomenon that generated the samples in the first place. As a simple example, consider figures 1a and 1b where a n:th order polynomial is fitted to some noisy data generated by a second order polynomial. As a more abstract example related to deep learning, one can consider the task of image recognition where the true function that produce the training samples is the human consciousness. The goal would in this case be to model the human decision process that takes the form of images being labeled as, say, ”cat” or ”dog”. The challenge in this is that, the samples only contain the *result* of the human decision process, and so, only the *result* of a very small portion of the conscious process of the human mind will be mimicked, at most. [6]



(a) The polynomial is memorizing this specific dataset by describing many of the small variations in the dataset.



(b) The polynomial captures the general trend in the dataset.

Figure 1: Fitting a n:th order polynomial to noisy data generated by a quadratic function. Figure 1a is an example of overfitting. Figure 1b is an example of a well fitted polynomial.

2.1 A Note on Tensors

In this work, multidimensional arrays will be referred to as tensors, which is different from the mathematical tensors. A mathematical object A whose elements are identified by tree or more indices will be called a tensor. As an example, the elements of a matrix are (a_{ij}), and the elements of what will be called a tensor are $(a_{ijk\ldots})$. The reason for this (ab)use of the meaning of the word tensor is the common terminology used in the field of machine learning.

2.2 Long Short-Term Memory

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture, which was introduced in 1997 as being better than its predecessors at capturing long-term dependencies in temporal sequences. [7] Since then, the LSTM has been successfully applied to a broad range of sequence modeling tasks, such as handwriting imitation, speech recognition, machine translation, computer vision, and prediction of human social behavior. [1, 2, 8, 9, 10, 11, 12] The basic idea behind an RNN is to iterate over the same RNN cell multiple times while using feedback connections to previous iterations. At each iteration, the RNN accepts an input vector and a state

vector. The input vector is used for updating the state vector, which is forwarded to the next iteration through the feedback connection, thus giving the algorithm its name. The feedback connection connects the cell to (in principle) all previous inputs. Therefore, the state will contain residues of all previous inputs. Because of this, RNNs are suitable for modeling sequences where elements depend on the ones preceding them. Naturally, such sequences are often time series. [7, 13, 14]

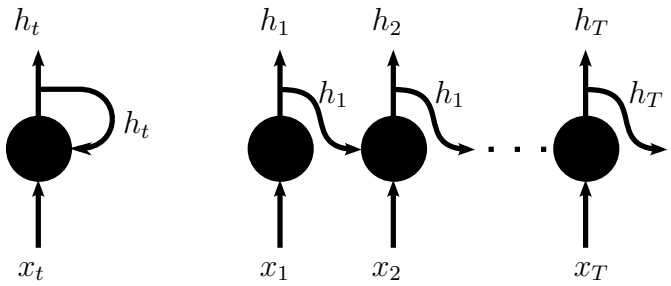


Figure 2: Unfolding of an RNN. Left: A single time step computation of the RNN. Right: An unfolded sequence of RNN computations.

When training an RNN, iterating the same computation is equivalent to traversing an unfolded graph of consecutive RNN cells, as visualized in figure 2. However, the feedback connection causes the state to be unstable. If the derivative of the cell's activation function has values between, and excluding, -1 and 1, the product of the partial derivatives that are calculated by the backpropagation algorithm, when applying the chain rule for calculating gradients, will decay as the number of time steps increases. Similarly, if the activation function has a derivative with values greater than 1 or smaller than -1, the product of partial derivatives will grow as the number of time steps increases. When gradients are decaying, the errors from long-term dependencies will become insignificant in comparison to the errors from short-term dependencies, and vice-versa for large gradients. This issue was a bottleneck for early implementations of RNNs. [14, 15] The long short-term memory is an attempt to address these issues. The LSTM has gating mechanisms that control to what extent the cell states are affected by new inputs and previous cell states. The LSMT has two internal states, a memory state for capturing long-term dependencies and a conventional cell state for capturing short-term dependencies. [7, 16]

The forward pass of the LSTM is described by

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{5}$$

$$h_t = o_t \odot \tanh c_t \tag{6}$$

where the sigmoid functions f_t, i_t and o_t have values beween 0 and 1. The Hadamard products work as a gating mechanism for controlling to what extent the elements of the cell state c_t and the memory state h_t are updated. [15] The forget gate f_t determines to what extent c_{t-1} remains unchanged, while i_t controls to what extent the state is updated with new information in $\tilde{c}_t$. A computational graph of equations (1) - (6) is shown in figure 3.

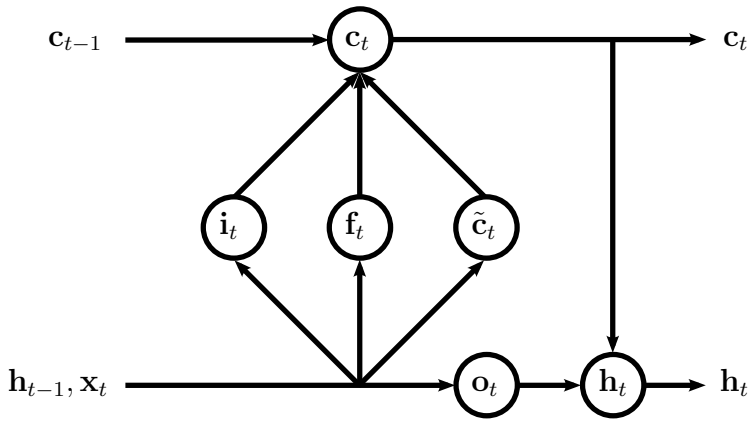


Figure 3: The LSTM represented as a computational graph.

2.3 Pooling the LSTM States of Spatially Nearby Particles

In the model that uses state pooling, the particles are represented by LSTMs. Each particle will be modeled by a separate LSTM, such that there are as many LSTMs

as there are particles. The LSTM state can be seen to contain an abstract representation of the physical properties of the particle that the LSTM represents. Some of these physical properties can cause an interaction between particles and some other properties can determine the nature of the interaction. For example, the charge distribution of an object determine the magnitude of its electric field and whether it behaves like a monopole, dipole, quadrupole, etc. (see section 2.6). Depending on their respective properties, two particles do not necessarily interact. To allow for the possibility that particles may interact, but without requiring it, and to allow for the possibility that different particles interact according to different rules, the properties of each particle must be taken into consideration when calculating the next position and state for a particle.

In dense systems, it is reasonable [1] to simplify the problem by only considering the closest neighboring particles, within some radial distance r, of each particle for which the trajectory is being predicted. A somewhat similar approach is also taken by Wang et. al. where particles outside the cut-off radius are simplified as monopoles. [3] By limiting the set of particles that are considered, it is assumed that the forces contributing to the interactions in the system are only significant within distances of r. Hence, r is a hyperparameter of the model and specific to the system. Using domain knowledge about the physical system, a suitable value for r can be chosen. The value for r can be further improved by hyperparameter tuning.

At each time instant t each particle i is described by a set of coordinates, x_t^i and y_t^i, and some arbitrary number of additional features. Additionally, each particle is also represented by an LSTM with a hidden (memory) state vector $\mathbf{h}_t^i$ and a cell state vector $\mathbf{c}_t^i$. Note that the superscript i represents an index rather than an exponent. To allow for the particles to interact, the hidden states are shared between neighboring LSTMs. As the hidden states are treated as some abstract representations of the physical states, it is unknown how these should be combined in order to model the particle-particle interaction. One of the things that is assumed, based on domain

[1] The magnitude of all interactions are taken to be inversely proportional to some power of the distance, $F \sim r^{-n}$, that is, they decrease with distance.

knowledge, is that the distances and angles to the neighbors are of significance when modeling the interactions. To give the LSTMs access to this spatial information, but without knowing how to interpret the values of the states in a physical sense, and therefore not knowing how to combine the two, state pooling is used as a general representation for the neighborhood of particles.

Similar approaches to the one that will be presented here, which learn interactions by pooling neighboring states, have been shown to be successful. However, these approaches focus mainly on modeling how humans move in crowded spaces. [2, 12] This thesis will explore if state pooling is useful for modeling particle-particle interactions, and will analyze whether or not this technique could be used when modeling molecular systems.

2.3.1 A brief Motivation for using LSTM State Pooling

One may ask why something like state pooling, which will be described shortly, is used. After all, the forces acting on each particle are determined by only the current state of the physical system. As will be seen in section 4, the initial inputs to the model are obtained through molecular dynamics simulations, which contain a complete description of the physical system, including all forces at each time instant. A problem with using any set of features for making predictions is that one still have to derive a description of the relationship between these features in order to accurately describe how the system evolves with time. For this, any sufficient set of features, such as LSTM states that are assumed to contain representations of the physical states of the particles, may be used.

In this work, only measurements of position are given, which means, no time derivatives of these are given. The time derivatives of the position measurements, such as velocities or accelerations, are needed in order to solve Newton's equations of motion. The time derivatives are easily approximated as discrete changes in the measured positions with respect to time. These changes in position are expected to

be found (calculated) by the LSTMs. The LSTM hidden states, as described in section 2.2, are expected to contain information about all such derived variables. State pooling can be seen as introducing a relationship between the inferred properties of multiple particles in a way that should allow the machine learning algorithm to learn the rules for these relationships (interactions), using minimal predefined knowledge. In systems with high particle density, state pooling can efficiently summarize the properties (LSTM states) of all particles in some neighborhood of predefined size by using some summarizing statistic, as will be described in the next sections.

2.3.2 Grid Based Sum Pooling

State pooling is a compact representation of the neighborhood of particles. It combines the states of the neighboring LSTMs while preserving spatial information. It can be seen as a collective state of the neighborhood. State pooling works by placing a square grid of size $N \times N$, centered at the i:th particle, in the coordinate system, where N is the number of cells along one side of the grid. The sides of this square grid have some physical length $l = 2r$, as illustrated in figure 4. The LSTM states of only those particles that fall within the boundaries of this grid will be used to construct the pooling tensor $\mathbf{H}_t^i$.

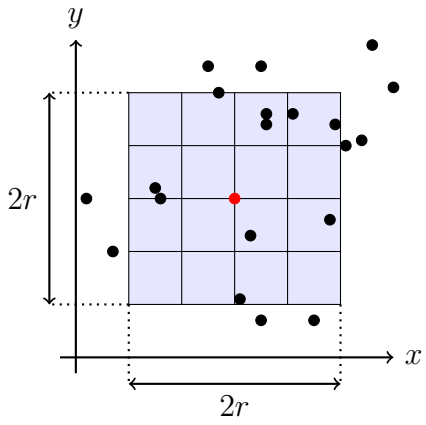


Figure 4: A $N \times N$ square pooling grid were $N = 4$. The primary particle (the i:th particle) is in the center of the grid and is marked with red, and its neighbors $i \neq j$ are marked with black.

Let $\mathbf{h}_t^i$ be the hidden state vector of the LSTM representing the i:th particle at time

t, and let (x_t^i, y_t^i) be the Cartesian spatial coordinates of the particle. The hidden states of the neighboring particles $j \neq i$ are shared with the i:th particle through a pooling tensor $\mathbf{H}_t^i$. Given some hidden state dimension D, the pooling tensor will be of shape $N \times N \times D$. The elements of $\mathbf{H}_t^i$ are given by

$$\mathbf{H}_{t,mn}^i = \sum_{j \neq i} \mathbf{1}_{mn}(x_t^j - x_t^i, y_t^j - y_t^i)\mathbf{h}_t^j \tag{7}$$

where $\mathbf{1}_{mn}(x, y)$ is an indicator function checking if a coordinate (x, y) is in the m, n cell of the pooling grid. This means that the matrix element $\mathbf{H}_{t,mn}^i$ is a D dimensional vector consisting of the sum of all state vectors $\mathbf{h}_t^j$ of those particles whose coordinates (x_t^j, y_t^j) are within the cell m, n of the pooling grid. This is depicted in figure 5.

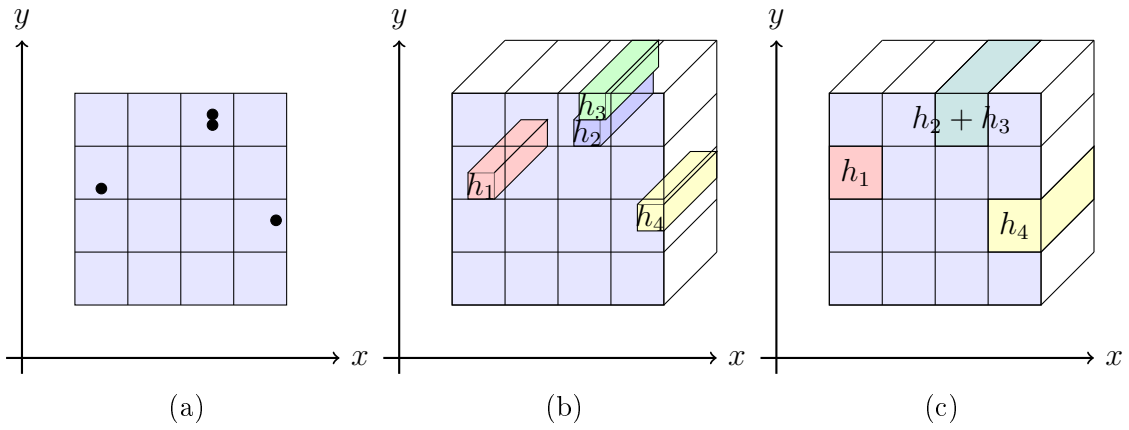


Figure 5: A $N \times N$ square pooling grid, were $N = 4$, is used to construct the pooling tensor H at some arbitrary time instant. In 5a, the pooling grid is placed in a coordinate system where the coordinates of four particles (black dots) will be used to construct $\mathbf{H}_t^i$. In 5b, the same four particles are represented as LSTM hidden state vectors $\mathbf{h}_i$. In 5c, the state vectors, whose corresponding coordinates are within the same grid cell, are summed.

2.3.3 Distance based filtering

Since the pooling grid is of square shape, the maximum allowed range of the interaction is not uniform in all directions. The distance between particles is allowed to be greatest along the diagonals of the pooling grid and smallest in vertical and

horizontal directions. Thus, on average, fewer particles are allowed to interact in directions $\theta = \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$, where θ is the angle from the x-axis in counter-clockwise direction, than in any other direction. In order to avoid introducing such bias into the model, the states $\mathbf{h}_t^j$ are filtered based on their corresponding particles' radial distance from particle i, as depicted in figure 6, before using them when constructing $\mathbf{H}_t^i$.

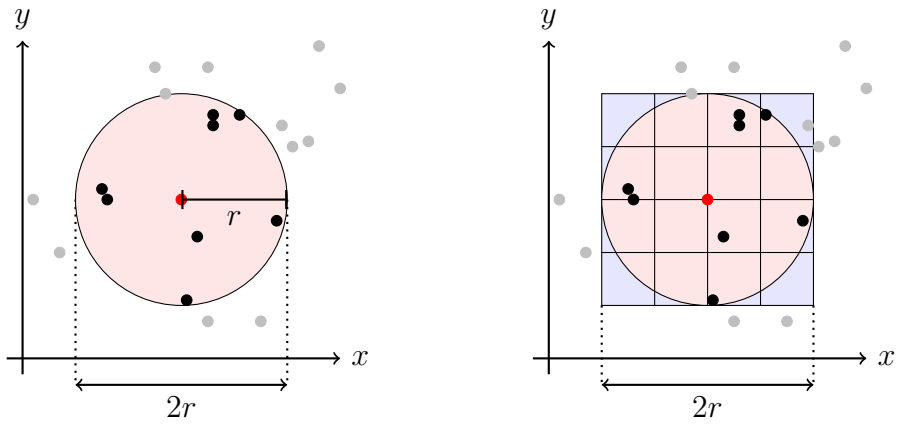


Figure 6: At any time t, only particles whose distance from the reference particle i (the red dot in the center) is less than r are considered when constructing the pooling tensor $\mathbf{H}_t^i$. In 6a, the indicator function $R(x, y)$ takes the value 1 inside the circle with radius r and the value 0 outside the circle. In 6b is depicted how the indicator function $R(x, y)$ is used to exclude some particles when constructing the pooling tensor $\mathbf{H}_t^i$.

The elements of $\mathbf{H}_t^i$ are now given by

$$\mathbf{H}_{t,mn}^i = \sum_{j \neq i} \mathbf{1}_{mn}(x_t^j - x_t^i, y_t^j - y_t^i) R(x_t^j - x_t^i, y_t^j - y_t^i) \mathbf{h}_t^j \tag{8}$$

where $R(x, y)$ is an indicator function like

$$R(x, y) = \begin{cases} 1 & \text{if } \sqrt{x^2 + y^2} \leq r \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

2.3.4 Limitations

State pooling, as described in previous sections, replaces the exact coordinates of neighboring particles $j \neq i$ with discrete evenly spaced coordinates relative to parti-

cle i. The process is very similar to rasterization in image processing. Transforming continuous coordinates to discrete ones introduces uncertainty in the values of the new representation. This uncertainty is greater for coordinates close to particle i compared to particles further away. This is illustrated in figure 7. Because the direction of the force acting on a particle is determined by the angle to the neighboring particle it interacts with, the uncertainty in the direction of the force will grow as the uncertainty in the angle grows. Similarly for the magnitude of the force the uncertainty is proportional to the uncertainty in the radial distance separating the interacting particles. For particles close to each other, the uncertainty arising from the rasterization is significant, and thus the prediction errors are expected to be greater when particles are close to each other compared to when they are far apart. A good balance between the resolution of the pooling grid and the number of trainable parameters has to be found through experiments. This limitation in precision must be considered when setting up the experiments and when evaluating the model's prediction performance.

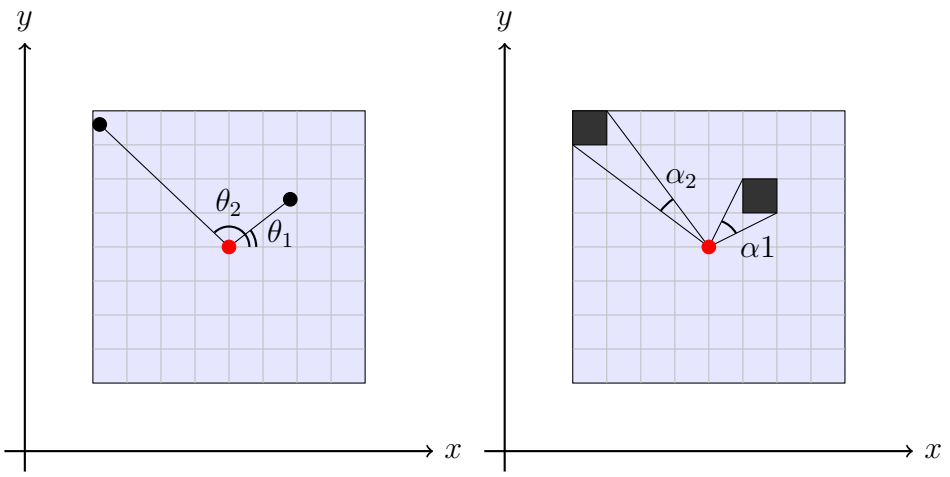


Figure 7: State pooling is similar to rasterization in image processing. The exact positions are replaced by pixels in a matrix. In this thesis, the pixel values are LSTM states. This process makes both the angular and radial coordinates discrete. For coordinates close to the center of the pooling grid, state pooling introduces greater uncertainty in the angular position than for more distant coordinates. This is illustrated by $\alpha_1 > \alpha_2$. Thereby, the force cannot be determined as precisely for a nearby particle as for a distant one. It is expected that there will be greater uncertainty in predictions for position of nearby particles than of distant ones.

2.3.5 Stacked State Pooling - An Attempt to Overcome the Limitations in Resolution

In an attempt to overcome the limited spatial resolution of the pooling grid, stacked state pooling is introduced. The hypothesis is that the resolution can be increased by using multiple identical pooling grids, each corresponding to a different physical size. The resulting pooling tensors can then be combined by calculating the element-wise sum. This is illustrated in figure 8. One can use a small neural network for interpreting the information in the stacked pooling tensor before feeding it as input to the LSTM. Whether or not this will provide any significant improvements in the results will be seen in the experiments of section 4.

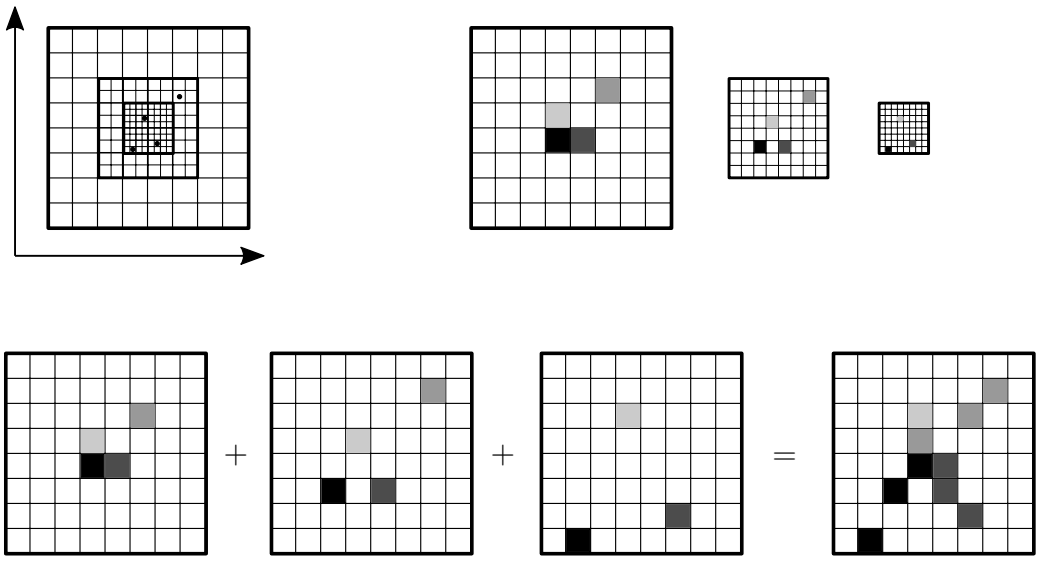


Figure 8: The top row: Three pooling grids of the same 8×8 shape but each corresponding to a different physical size. A small grid will record more precise positions than a large gird. However, the small grid will only detect particles within a small total area. Similarly, a large grid will be able to detect particles within a large total area. Coordinates at short distances from the common center point of the grids can be recorded with greater precision than coordinates at long distances from the common center point. The bottom row: By combining three pooling tensors (in this case by summation), a stacked pooling tensor is obtained. Each distinct particle position in the coordinate system (top left) will lead to unique patterns in the stacked pooling tensor (bottom right).

2.3.6 Motivation for using Sum Pooling from a Perspective of Physics

In section 2.3.2, a type of sum pooling was described. The elements of the pooling tensor $\mathbf{H}_t^i$, given by equation (8), are sums of hidden state vectors $\mathbf{h}_t^j$. When implementing the pooling operation, the hidden states could equally well be pooled by some other summarizing statistic, such as the average. [17]. The choice of summarizing statistic should be motivated by the problem one attempts to solve. From a purely physical perspective, the choice of using a sum as the summarizing statistic can be motivated by reasoning that the net force $\mathbf{F}$ acting on a single particle is a sum of all contributing forces $\mathbf{F}_j$. All forces in the system are taken to be either constant or proportional to some power of the inverse radial distance r, such that $\mathbf{F} \sim \sum_j k_j \frac{\hat{\mathbf{r}}_j}{r_j^{-p}}$, where p is a positive integer and k is a constant that depends on the properties of the interacting particles. Since the forces are approximately invariant to small translations, multiple nearby particles can be approximated as a single particle whose position is given by the mean coordinate.

$$\mathbf{F}(\mathbf{r}_n) \approx \mathbf{F}(\mathbf{r}_m) \qquad \text{for } \mathbf{r}_n \approx \mathbf{r}_m \tag{10}$$

$$\mathbf{F} \sim k_1 \frac{\hat{\mathbf{r}}_1}{r_1^{-p}} + \cdots + k_N \frac{\hat{\mathbf{r}}_N}{r_N^{-p}} \approx (k_1 + \ldots k_N) \frac{1}{N} \left(\frac{\hat{\mathbf{r}}_1}{r_1^{-p}} + \cdots + k_N \frac{\hat{\mathbf{r}}_N}{r_N^{-p}} \right) \tag{11}$$

State pooling replaces continuous coordinates with discrete ones, see figure 6. Because the force is approximately invariant to small changes in position, the properties of multiple particles with the same discrete coordinates can be summed according to equation (11). Particle properties k are represented by hidden state vectors $\mathbf{h}^i$. Thus, it is reasonable to sum the state vectors of all particles that have the same discrete coordinates. The state vectors are summed component-wise, which means that different properties will not be mixed.

$$\mathbf{h}^i + \mathbf{h}^j = \left(h_n^i + h_n^j \right) \tag{12}$$

Therefore, a vector sum will be used when pooling the LSTM hidden states of nearby particles.

2.4 Convolutional Neural Networks

The pooling tensor $\mathbf{H}_t^i$ can easily become very large as it has the shape $N \times N \times D$, where D is the dimensionality of the LSTM states. The tensor $\mathbf{H}_t^i$ will be used as input to an LSTM which uses several fully connected layers, equations (1)-(4), to transform the input. [7, 16] As the pooling tensor is large, these transformations become computationally expensive to perform. More importantly, the number of trainable parameters can become prohibitively large. One can do several things in order to reduce the size of the pooling tensor while at the same time extracting the relevant information from it. The LSTM states $\mathbf{h}_t^i$ can be transformed (embedded) into a lower dimension before pooling, or the pooled states can be embedded into a lower dimension. Another option is to use a convolutional neural network (CNN). CNNs have been shown to be good feature extractors in various computer vision tasks. [11, 18, 19]. Similar to an image of shape $H \times W \times C$, where H and W are the height and width of the image, respectively, and C is the number of color channels, the pooling tensor $\mathbf{H}_t^i$ is of shape $N \times N \times D$. The convolutional layers may extract important features from the pooling tensor similarly to how features are extracted from an image. A CNN have very few trainable parameters compared to a fully connected layer, thereby making it a suitable component to use between the state pooling and the LSTM.

One important weakness of CNNs that should be kept in mind when using them is that CNNs are not generally invariant to translations of the input, only equivariant. This is attributed to the use of subsampling in the convolutional layers. [20, 21, 22] However, this will not be an issue, as the pooling grid is always centered on the primary particle, and so shift invariance is not necessary.

Even though the name suggest a convolution is calculated, a cross-correlation is used in practice for performance reasons. In early CNNs convolution was calculated and the name has stuck since then. [17] The simplest two-dimensional (2D) convolutional layer takes as input a 2D array $A \in \mathbb{R}^2$ and calculates the cross correlation between the input and a 2D filter $K \in \mathbb{R}^2$ that is smaller than the input. The cross correlation

C is calculated as

$$(c_{p,q}) = \sum_{m,n} (a_{p+m-1,q+n-1}) \odot (k_{m,n}) \tag{13}$$

This is visualized in figure 9. When the input is a tree-dimensional array, like the pooling tensor or an color image, the cross correlation is calculated separately for each channel, using a separate filter for each channel. In the case of D channels, there are D filters and the result is D separate cross correlations C_d. The D separate cross correlations are summed together to produce a single output C. This whole operation is called a convolutional kernel. Additionally, one may use multiple kernels. When using B kernels there will be B separate outputs. The result is an array of shape $N_1 \times N_2 \times B$. Finally, the convolutional layer is often followed by a pooling layer whose purpose is to reduce the size of the output by, for example, summing together groups of elements. [17, 23] This operation is visualized in figure 10.

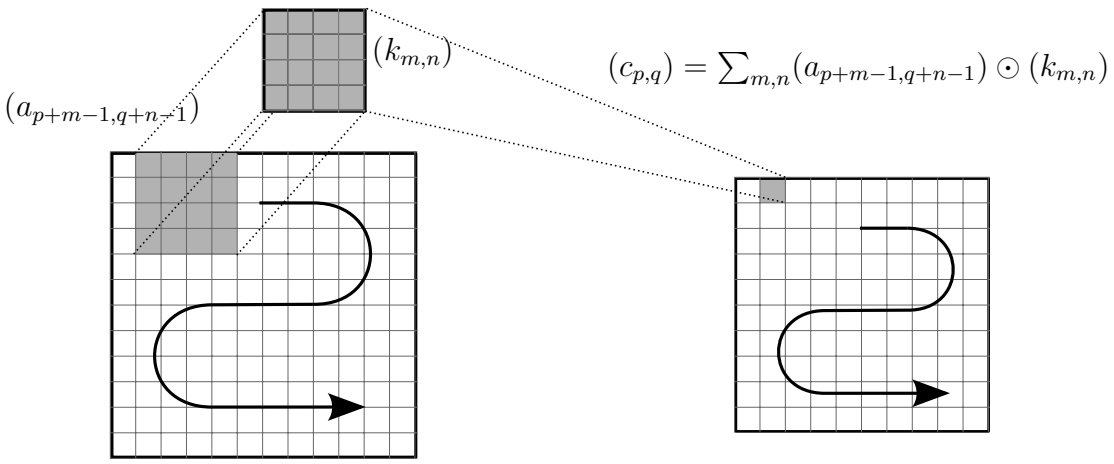


Figure 9: A visualization of equation (13). The cross correlation is calculated between the matrix K (the filter) and the input matrix A. This can be visualized as sliding the filter over the input, while at each position calculating the Hadamard product of the filter and the portion of the input that the filter overlap, and summing the elements of the Hadamard product to give a single scalar.

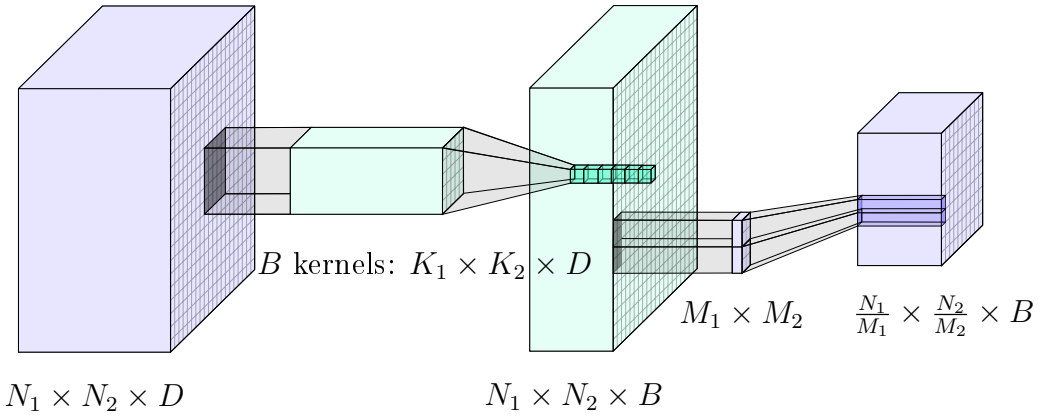


Figure 10: Illustration of a two dimensional convolution layer followed by a pooling layer with stride equal the size of the pooling kernel's size. From left to right: An input of size $N_1 \times N_2 \times D$, on which B kernels of size $K_1 \times K_2 \times D$ operate (only one kernel is illustrated), producing a convolution of size $N_1 \times N_2 \times B$, which is then pooled with a pooling kernel size of $M_1 \times M_2$, producing a final output of size $\frac{N_1}{M_1} \times \frac{N_2}{M_2} \times B$. In the illustration the input is assumed to be appropriately padded to produce a same sized output $N_1 \times N_2$. The illustrated convolutional kernel uses no dilation (i.e. uses the whole input) and is assumed to be shifted by steps of length 1.

2.5 Probabilistic Predictions

State pooling, as described in section 2.3, introduces uncertainty into the coordinates of neighboring particles. Therefore the predicted position estimates for the primary particle will also be inherently uncertain. By this reasoning, the estimates for target positions should not be exact values, but rather expressed as some probability density function conditioned on the history of previous positions. This is achieved by using the model output to parameterize such a probability density function. When the true distribution of the target is unknown, the probability density for a Gaussian distribution should be used, as it introduces the least amount of prior knowledge into the model. [24]

The value of a Gaussian probability distribution $\mathcal{N}$ at some coordinate a gives the relative likelihood of the outcome a being randomly drawn from the distribution $\mathcal{N}$. In other words, it answers the question, given some outcome (training data), what is the relative likelihood that the Gaussian distribution generated by the model

describes that outcome? This means that when the probability density is generated by the model in the form of a prediction, and the sample is the target coordinates in the training data, the value of the probability density at the target coordinates is the relative likelihood that the model would generate the observed data. Thereby, the likelihood $\mathcal{L}$ is a measure of how well the model describes the target data $\mathbf{y}$, and so the likelihood $\mathcal{L}$ should be maximized with respect to the model parameters $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}_0 = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathcal{L}(\mathbf{y}|\boldsymbol{\theta}) \tag{14}$$

At any given time, the predicted probability density describing the position at $t+1$ should be conditioned on the entire history of past positions. The motivation for this can be intuitively understood as follows. The model predicts position as probability densities ξ, rather than of exact positions x. Assuming the position and velocity of some primary particle, and the positions of all neighboring particles are known, the exact [2] position of the primary particle at time $t+\Delta t$ can be predicted, by calculating how much x will change during the time Δt, as described in section 4.2. However, this requires that the exact value for $x(t)$ is known. Because the model uses state pooling for calculating probabilistic predictions, the exact position $x(t)$ is not known. Instead the position is described by a probability density $\xi(t)$. Therefore $\xi(t+\Delta t)$ should be a conditional probability density, conditioned on $\xi(t)$. Similarly, when $\xi(t)$ was predicted, the preceding position was $\xi(t-\Delta t)$. Thus $\xi(t)$ should be conditioned on $\xi(t-\Delta t)$, which in turn should be conditioned on $\xi(t-2\Delta t)$, and so on. Therefore, a prediction at time t should be conditioned on all past positions.

Given some M-dimensional input vector $\mathbf{x} = (x_1, \ldots, x_M)$ and an N-dimensional target vector $\mathbf{y} = (y_1, \ldots, y_N)$, with independently normally distributed target components y_n, the conditional relative likelihood of observing a component y_n is given by

$$p(y_n|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{1}{2}\left(\frac{y_n-\mu}{\sigma}\right)^2} \tag{15}$$

which is the probability density for a one-dimensional Gaussian distribution. The

[2] Exact with respect to the training data.

variables μ and σ are the mean and standard deviation, respectively, of the component y_n. In the two-dimensional case, for which $N = 2$, the relative likelihood of observing both target components y_1 and y_2 simultaneously is given by the product of the likelihoods of observing each component separately.

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{2\pi\sigma_1\sigma_2} \mathrm{e}^{-\frac{1}{2}\left(\frac{y_1-\mu_1}{\sigma_1}\right)^2+\frac{1}{2}\left(\frac{y_2-\mu_2}{\sigma_2}\right)^2} \tag{16}$$

In D dimensions, when the components of $\mathbf{y}$ are independently distributed, one can generalize equation (16) as

$$p(\mathbf{y}|\mathbf{x}) = \prod_{d=1}^{D} p(y_d|\mathbf{x}) = \prod_{d=1}^{D} \frac{1}{(2\pi)^{d/2}\sigma_d} \mathrm{e}^{-\frac{1}{2}\left(\frac{y_d-\mu_d}{\sigma_d}\right)^2} \tag{17}$$

where μ_d and σ_d now are the mean and standard deviation, respectively, of the component y_d in the d:th dimension. Equation (17) will become useful when mixture densities are considered. If the target components in the two-dimensional case are not independent, the correlation ρ must be added to equation (16). This modification will turn equation (16) into the probability density function for a bivariate Gaussian distribution.

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \mathrm{e}^{-\frac{1}{2(1-\rho^2)}\left(\left(\frac{y_1-\mu_1}{\sigma_1}\right)^2+\left(\frac{y_2-\mu_2}{\sigma_2}\right)^2-2\rho\left(\frac{y_1-\mu_1}{\sigma_1}\right)\left(\frac{y_2-\mu_2}{\sigma_2}\right)\right)} \tag{18}$$

$$\equiv \mathcal{N}_n(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) \tag{19}$$

The likelihood of observing the input vector $\mathbf{x}$ and the target vector $\mathbf{y}$ simultaneously is given by the joint probability density [25]

$$\mathcal{L} \equiv p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) \tag{20}$$

The model output $\hat{\mathbf{y}}$ will be used to parameterize a Gaussian distribution by letting the means, standard deviations and the correlation be functions of the output $\hat{\mathbf{y}}$. The output is calculated by a fully connected layer with a linear activation function. Since the means μ_d should have values in the open interval $\mu_d \in]-\infty, \infty[$, the corresponding components of the output vector are used without any modifications. Since the

standard deviations σ_d should be in the open interval $\sigma_d \in]0, \infty[$, a softplus function is applied to the corresponding components of the output vector. A hyperbolic tangent is used for limiting the correlation ρ to the open interval $]-1, 1[$.

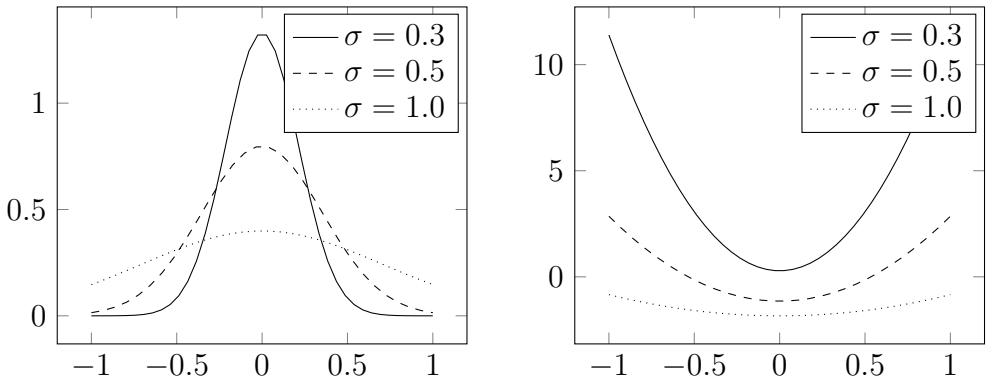
$$\mu_d(\hat{\mathbf{y}}) = \hat{y}_{\mu,d} \tag{21}$$

$$\sigma_d(\hat{\mathbf{y}}) = \text{softplus}(\hat{y}_{\sigma,d}) \equiv \ln(e^x + 1) \tag{22}$$

$$\rho_d(\hat{\mathbf{y}}) = \tanh \hat{y}_{\rho,d} \tag{23}$$

Thus, the likelihood $\mathcal{L}$ of the target vector $\mathbf{y}$ is predicted by the model. Optimizing of the neural network loss is now a matter of maximizing the relative likelihood $\mathcal{L}$ with respect to the layer weights. Since the likelihood $\mathcal{L}$ is expressed by a continuous exponential function, it is convenient to maximize the logarithmic likelihood. Maximizing the logarithmic likelihood is equivalent to minimizing the negative logarithmic likelihood. Thereby, the error function E of the neural network is defined.

$$E = -\ln \mathcal{L} \tag{24}$$



(a) The probability density functions for three normal distributions with different standard deviations σ.

(b) The negative logarithmic likelihood. A smaller standard deviation lead to a more distinct minima.

Figure 11

Everything so far applies to single vector transformations $f : \mathbf{x} \to \hat{\mathbf{y}}$. In order to find a transformation f that can be claimed to describe the true function or phenomenon that generated the target vector $\mathbf{y}$, more than one sample must be considered. Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_S\}$ be a set of input vectors and $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{x}_S\}$ be a set of target

vectors. The simultaneous likelihood of the whole data set is the product of the likelihood of each separate transformation f_s. [9, 15]

$$\mathcal{L} = \prod_s \mathcal{L}_s \tag{25}$$

The error function now becomes

$$E = -\ln \prod_s \mathcal{L}_s \tag{26}$$

$$= -\ln \prod_s p(\mathbf{y}_s|\mathbf{x}_s)p(\mathbf{x}_s) = -\ln \prod_s p(\mathbf{y}_s|\mathbf{x}_s) - \ln \prod_s p(\mathbf{x}_s) \tag{27}$$

$$= -\sum_s \ln p(\mathbf{y}_s|\mathbf{x}_s) - \sum_s \ln p(\mathbf{x}_s) \tag{28}$$

The likelihood $p(\mathbf{y}_s|\mathbf{x}_s)$ is predicted by the model, such that $p(\mathbf{y}_s|\mathbf{x}_s) \equiv p(\mathbf{y}_s|\hat{\mathbf{y}}_s)$ is a function of the model parameters $\boldsymbol{\theta}$ and the input $\mathbf{x}_s$. The second term in equation (28) does not depend on the model parameters and is, thus, a constant in the optimization problem. The error function for the complete set $\mathcal{X}$ is reduced to

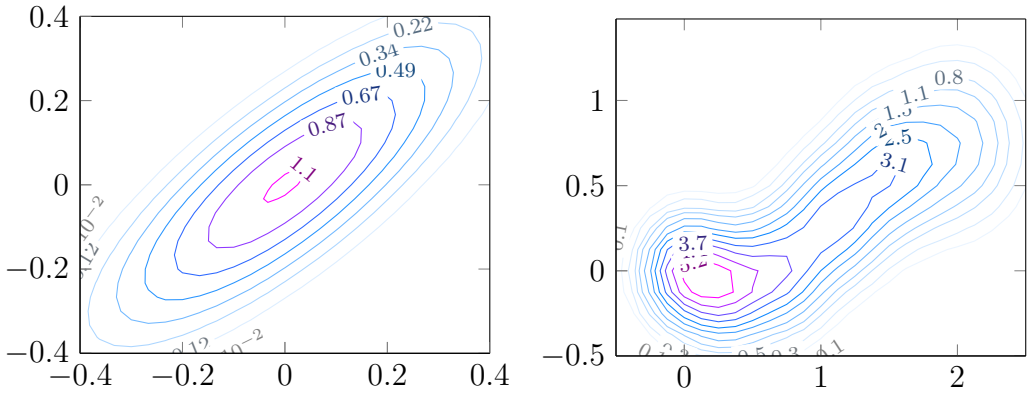
$$E = -\sum_s \ln p(\mathbf{y}_s|\hat{\mathbf{y}}_s) \tag{29}$$

where the relative likelihood $p(\mathbf{y}_s|\hat{\mathbf{y}}_s)$ is given by equation (17) or (18).

2.5.1 Mixture Density Network

It is not always appropriate to calculate a single-valued prediction. This is the case for problems where the solution is multi-valued and the average is not necessarily a correct solution itself. Imagine a particle restricted to some circular motion. If there is uncertainty in the exact angular position of the particle, then the particle is still bound to be found somewhere along a circular trajectory, but now with a (normalized) probability density function $p(\alpha)$ describing the probability of finding the particle at coordinates (R, α), where R is the radius of the circular trajectory. The mean Cartesian position $(\langle x \rangle, \langle y \rangle) = \int_\alpha p(\alpha)(r\cos(\alpha), r\sin(\alpha))d\alpha$ would be somewhere inside the circle at coordinates given by $\langle x \rangle^2 + \langle y \rangle^2 < R^2$. Similarly, for other non-linear trajectories, the mean would not always be a suitable metric for describ-

ing the position of a particle. Relevant to this thesis are such particle trajectories where there is inherent uncertainty in the position estimates at some specific time, and thus the next position would have to be expressed as a probability density along some non-linear trajectory, see figure 12b. This thesis will be restricted to problems in only two spatial dimensions.



(a) A plot of the probability density function of a bivariate Gaussian distribution. Such a probability density can describe linear relationships.

(b) A plot of a Gaussian mixture density. A mixture density can describe non-linear relationships.

Figure 12

A single bivariate Gaussian distribution, given by (18), is limited to describing a linear relationship between two stochastic variables, as illustrated in figure 12a. As spatial trajectories of interacting particles are generally arbitrary, the probability density function $p(\mathbf{y}_s|\hat{\mathbf{y}}_s) \equiv p(\mathbf{y}_s|\mathbf{x}_s)$ should be able to express such arbitrary non-linear functions. Arbitrary density functions can be constructed as a mixture of component probability densities ϕ_i as

$$p(\mathbf{y}_s|\mathbf{x}_s) = \sum_i c_i(\mathbf{x}_s)\phi_i(\mathbf{y}_s|\mathbf{x}_s) \tag{30}$$

where c_i are normalized mixing coefficients and ϕ_i are probability densities for Gaussian distributions given by equation (17) or (18). The index i denotes the i:th component of the mixture. In general, the correlation ρ_i in equation (18) is redundant when used in a mixture density, since a Gaussian mixture with component probabilities given by (17) is a universal approximator, capable of approximating any density

function with an arbitrary non-zero error. The error is proportional to the number of mixture components, and in the limit of $i \to \infty$, it will be zero. [25] A Gaussian mixture density is illustrated in figure 12b. When using component densities like (17), the D separate standard deviations $\sigma_{i,d}$ (one for each dimension d) of the i:th mixture component can be replaced by a global standard deviation σ_i of the mixture component. [26] However, in this thesis, a small number of mixture components will be preferred over a very generalized model, and therefore components given by equation (17) will be used.

When the mixing coefficients $c_i(\mathbf{x}_s) \equiv c_i(\hat{\mathbf{y}}_\mathbf{s})$ and probability densities $\phi_i(\mathbf{y}_s|\mathbf{x}_s) \equiv \phi_i(\mathbf{y}_s|\hat{\mathbf{y}}_s)$ are modeled by a conventional neural network, it gives rise a mixture density network. [26] The mixture coefficients should be properly normalized, so that $\sum_i c_i(\mathbf{x}_s) = 1$. A Boltzmann distribution (sometimes also called a softmax distribution) can be applied to the model output $\hat{\mathbf{y}}_s^c$, corresponding to the mixture coefficients of sample s, in order to normalize the mixture coefficients. The Boltzmann distribution is given by

$$c_i(\mathbf{x}_s) = \frac{\exp(\hat{y}_{s,i}^c)}{\sum_{j=1}^{M} \exp(\hat{y}_{s,j}^c)} \equiv \frac{\exp(\hat{y}_{s,i}^c)}{Z_s} \qquad \text{for } i, j = 1, 2, \ldots, M \tag{31}$$

where the normalization constant Z_s is summed over all mixture components corresponding to a sample s. By substituting the mixture density (30) into the sequence error (29), the error for the mixture density network becomes

$$E = -\sum_s \ln \sum_i c_i(\mathbf{x}_s)\phi_i(\mathbf{y}_s|\mathbf{x}_s) \tag{32}$$

2.5.2 Recurrent Neural Networks and Probability Densities

In the case of temporal sequences, the input sequence $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ is used for calculating a sequence of hidden states $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_T\}$ which, in turn, is used for calculating a sequence of outputs $\hat{\mathcal{Y}} = \{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_T\}$. An RNN will be used for generating an output sequence of arbitrary length by treating the output vector $\hat{\mathbf{y}}_t$ as a new input vector $\mathbf{x}_{t+1}$.

Real particles have properties that are essential for making accurate predictions that are consistent with Newtonian mechanics (see section 4.1). Some of these properties, such as velocities, cannot be determined based on only a single observation of a vector $\mathbf{x}_t$. [3] In order to allow for the initial prediction $\hat{\mathbf{y}}_T$ to be based on some statistical prior, $\hat{\mathbf{y}}_T$ will be based on a short input sequence $\mathcal{X}_{\text{in}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, instead of only a single vector $\hat{\mathbf{x}}_T$. The initial output $\hat{\mathbf{y}}_T$ corresponds to the true target $\mathbf{x}_{T+1}$. As the model will produce an output sequence of arbitrary length T', the complete output sequence will be $\hat{\mathcal{Y}} = \{\hat{\mathbf{y}}_T, \ldots, \hat{\mathbf{y}}_{T+T'}\}$, which corresponds to a true target sequence $\mathcal{X}_{\text{target}} = \{\mathbf{x}_{T+1}, \ldots, \mathbf{x}_{T+1+T'}\}$. Similarly, the sequence of hidden state vectors will be $\mathcal{H} = \{\mathbf{h}_T, \ldots, \mathbf{h}_{T+T'}\}$.

In terms of the negative logarithmic likelihood (29), it does not matter whether the different samples are simultaneous transformations of multiple vectors or repeated transformations of a single vector. Thus, a temporal sequence, which is a repeated transformation of a single vector, can be treated as a set of simultaneous transformations $f : \mathbf{x}_t, \mathbf{h}_{t-1} \to \hat{\mathbf{y}}_t$. The error of the output sequence will be a sum over the temporal samples. [1, 15]

$$E = -\sum_{t=T}^{T+T'} \ln p(\mathbf{x}_{t+1}|\mathbf{x}_t) \equiv -\sum_{t=T}^{T+T'} \ln p(\mathbf{x}_{t+1}|\hat{\mathbf{y}}_t) \tag{33}$$

For a mixture density network, the error will be [1, 26]

$$E = -\sum_{t=T}^{T+T'} \ln \sum_i c_i(\mathbf{x}_t)\phi_i(\mathbf{x}_{t+1}|\mathbf{x}_t) \equiv -\sum_{t=T}^{T+T'} \ln \sum_i c_i(\mathbf{x}_t)\phi_i(\mathbf{x}_{t+1}|\hat{\mathbf{y}}_t) \tag{34}$$

2.6 Multipole Expansion

When modeling interaction between molecules, or any object, it is convenient to use approximations. The electric field due to an arbitrary charge distribution can be

[3] An object with zero initial velocity will start to move in the direction of the force it is subject to. However, if the initial velocity is not zero, the force will only cause a change in the object's initial velocity. Without knowing the initial velocity, one can only guess where the particle will be after some time Δt. Therefore, knowing the initial velocity is essential when estimating the future position.

seen to contain a monopole term, a dipole term, a quadrupole term, and other higher order terms. Given some arbitrary charge distribution, as in figure 13, the potential in the point given by $\mathbf{r}$ is given in Gaussian units by

$$\varphi(\mathbf{r}) = \int_V \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dv' \tag{35}$$

where $\rho(\mathbf{r}')$ is the charge density in the point given by $\mathbf{r}'$. [27] (In SI units φ is scaled by a proportionality constant $k_E = 1/(4\pi\varepsilon_0)$, see for example [28]).

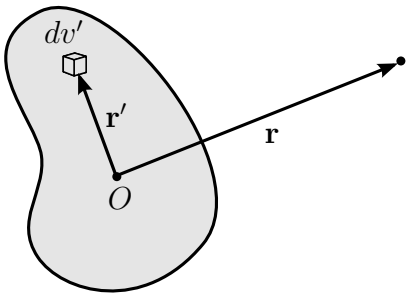


Figure 13: An arbitrary charge distribution centered at origin and an infinitesimally small volume dv' at the point given by the vector r' from origin.

By centering the volume at origin and requiring that $|\mathbf{r}| > |\mathbf{r}'|$, i.e. the point given by $\mathbf{r}$ is outside the volume, the denominator $|\mathbf{r} - \mathbf{r}'|^{-1}$ can be expanded in terms of $\frac{\mathbf{r}'}{\mathbf{r}}$ as follows.

$$|\mathbf{r} - \mathbf{r}'|^{-1} = \left(r^2 - 2\mathbf{r}\mathbf{r}' + r'^2\right)^{-\frac{1}{2}} \tag{36}$$

$$= \frac{1}{r}\left(1 - \frac{2\mathbf{r}\mathbf{r}'}{r^2} + \frac{r'^2}{r^2}\right)^{-\frac{1}{2}} \tag{37}$$

$$= \frac{1}{r}\left(1 - \frac{1}{2}\left[-\frac{2\mathbf{r}\mathbf{r}'}{r^2} + \frac{r'^2}{r^2}\right] + \frac{1}{2}\frac{1}{2}\frac{3}{2}\left[-\frac{2\mathbf{r}\mathbf{r}'}{r^2} + \frac{r'^2}{r^2}\right]^2 - O(r'^3)\right) \tag{38}$$

$$= \frac{1}{r} + \frac{\mathbf{r}\mathbf{r}'}{r^3} - \frac{1}{2}\frac{r'^2}{r^3} + \frac{3}{8}\frac{4(\mathbf{r}\mathbf{r}')^2}{r^5} + O(r'^3) \tag{39}$$

where the binomial expansion

$$(1 + x)^s = 1 + sx + \frac{s(s-1)}{2!}x^2 + ... \tag{40}$$

was used. Substituting the expression for $|\mathbf{r}-\mathbf{r}'|^{-1}$ into equation (35) yields

$$\varphi(\mathbf{r}) = \int_V \left(\frac{1}{r} + \frac{\mathbf{r}\mathbf{r}'}{r^3} + \frac{1}{2}\left[\frac{3(\mathbf{r}\mathbf{r}')^2}{r^5} - \frac{r'^2}{r^3} \right] + O(r'^3) \right) \rho(\mathbf{r}')dv' \tag{41}$$

$$\begin{aligned} \varphi(\mathbf{r}) =& \frac{1}{r}\int_V \rho(\mathbf{r}')dv' + \frac{\mathbf{r}}{r^3}\int_V \mathbf{r}'\rho(\mathbf{r}')dv' \\ &+ \sum_{i=1}^{3}\sum_{j=1}^{3} \frac{1}{2}\frac{x_i x_j}{r^5} \int_V (3x_i'x_j' - \delta_{ij}r'^2)\rho(\mathbf{r}')dv' + O(r'^3) \end{aligned} \tag{42}$$

where $x_1 = x$, $x_2 = y$ and $x_3 = z$. The first term in equation (42) is the monopole contribution, the second term is the dipole contribution, the third term is the quadrupole contribution, and the fourth term is the contribution from higher order moments. [27]

2.6.1 Monopole Approximation

As r grows in proportion to r', the quadrupole moment quickly diminishes. For small molecules, i.e. molecules where r' is small, the contribution of the dipole moment will also quickly diminish. Therefore, it can be assumed that for a system of molecules, where the molecules are small in comparison to the distance separating them, only the monopole contribution needs to be taken into consideration. In such systems, one can make the approximation of point-like particles with no geometric orientation.

2.6.2 Dipole Approximation

In many cases it is not appropriate to only use the monopole approximation. In such cases one can retain more information about the particles by approximating the particles as dipoles, keeping the first two terms in equation (42). Given a particle with some arbitrary charge distribution, the particle can be roughly approximated as a dipole by describing it with the dipole moment

$$\mathbf{p} = q\mathbf{d} \tag{43}$$

where q is a measure of the charge of the dipole and $\mathbf{d}$ is the displacement vector describing the net charge separation. By using a molecule's geometric orientation as input to a neural network, one could let let the neural network build some abstract

description of the molecules geometry and/or dipole moment.

3 A First Experiment - A Preamble to a Second Experiment

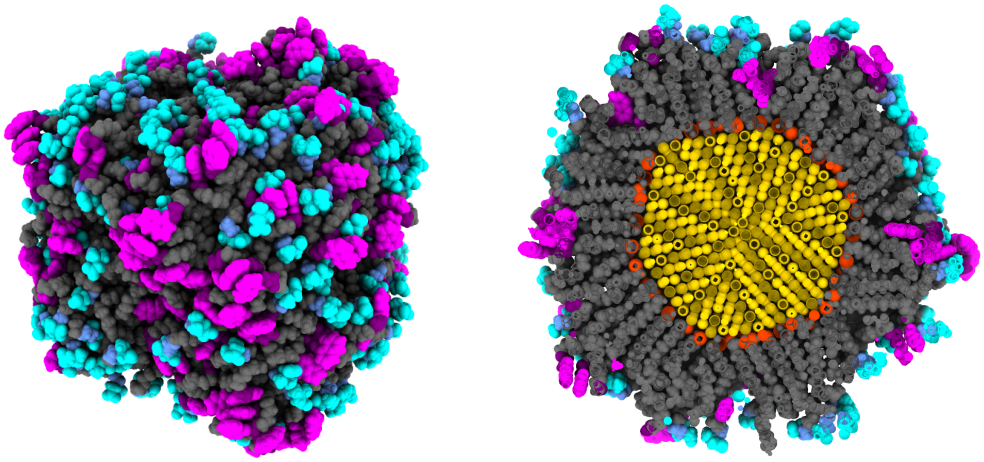
The purpose of the experiment is to explore the whether or not the motion of the surface molecules in a large nanoparticle can be modeled using a simple neural network based machine learning model. The model is trained on molecular dynamics simulations of several functionalized drug-carrying nanoparticles in aqueous environments. The nanoparticles consist of a gold core to which thiolated ligands are attached by the (self-assembling) gold-sulfur bonds that the thiol and the gold form. To the free ends of the ligand either a drug molecule or a background molecule is attached. The drug may either be hydrophobic (quinolinol) or hydrophilic (panobinostat). The main purpose of the hydrophilic background molecule is to increase the solubility of the hydrophobic drug. It is non-trivial to determine an optimal ratio of drug to background molecules, such that the amount of dissolved drug is maximized. See Kovacevic, et. al (2021) for further details of the simulated systems. [29]

When designing drug-carrying nanoparticles like the ones described above, molecular dynamics simulations are run in order to evaluate how the particles' properties are affected by changes in the system. [29] In order to accelerate the search for promising designs, mainly promising drug to background molecule ratios, a machine learning algorithm could help interpreting complex relationships between different atomistic and molecular properties in the system as a whole. The machine learning algorithm could give an estimate for different particle properties, mainly the solvent accessible surface area (SASA), without the need to run full molecular dynamics simulations that are computationally expensive. This first approach tries to model the motion of the surface molecules, the spatial configurations of which determine the solvent accessible surface area, in order to explore whether or not there are some simple patterns present in the motion that could be exploited for estimating the SASA. A few alterations of a simple LSTM centered model was developed. All models take molecule positions and labels as input and tries to predict the future positions from these. The hypothesis is that if there is some generic pattern in the motion of

the molecules, the model may capture it and successfully model the motion of the molecules.

3.1 Data Exploration

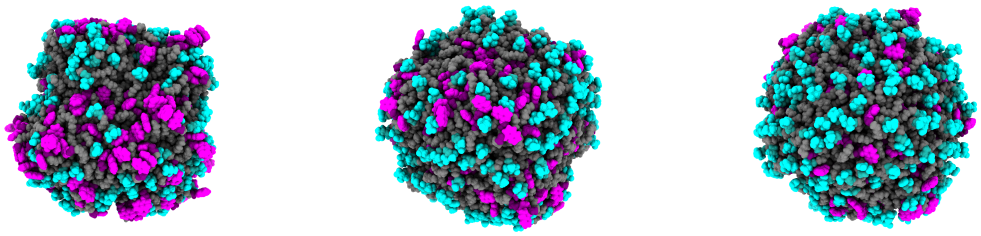
The molecule positions that are used as training data in the machine learning optimization problem are obtained through molecular dynamics (MD) simulations. The Amber molecular dynamics package [30] was used to model the drug carrier consisting of approximately 40 000-50 000 atoms in a water solvent for 300 ns. In total, 44 simulations with different drugs and different ratios of drug to background molecules were created. From each simulation a set of `pdb` files, one per nanosecond, were extracted, thereby creating a timeseries of 300 `pdb` files. A `pdb` file contains the structural information of the system at some specific time instance in the simulation. This structural information includes, among other things, the coordinates of each atom and information about which atoms make a molecule. A visual render of a drug-carrying nanoparticle, based on a single `pdb` file, is shown in figures 14a and 14b. The drug carrier is approximately 30 nm in diameter. During the length of a simulation the drug carrier evolves towards its equilibrium configuration. Since the drug carrier is in a solvent of small molecules, which in this case are water molecules, the motion of the drug carrier's surface molecules are mostly thermal (Brownian) motion. Because of this, the trajectories of the surface molecules are random walks with possibly (depending on the molecule) a trend in some direction. For example, the hydrophobic quinolinol is repelled by the water and is forced towards to the center of the drug carrier and ends up buried among the ligands, whereas the hydrophilic background molecule is attracted by water and is forced to the surface (water and drug carrier interface) of the drug carrier. This is visualized in figure 15 and quantified in figure 16. In the case of a hydrophilic drug, it is competing for space at the water interface with the also hydrophilic background molecule and the ratio of the two determine the final properties of the drug carrier. [29]



(a) A drug-carrying nano-particle with approximately 40 000 atoms. Drug molecules (bright pink) and background molecules (bright turquoise) attached on the ends of ligands (grey) that are attached to a gold core (not visible).

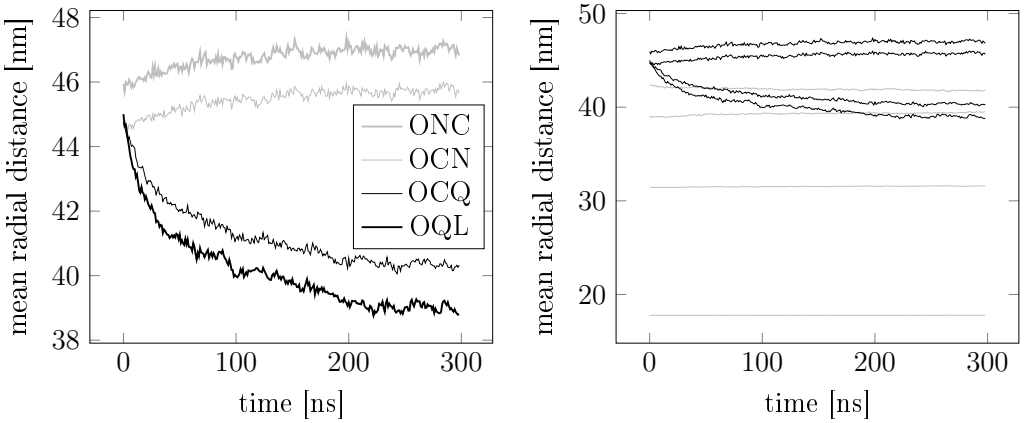
(b) A cross section view of a drug-carrying nanoparticle. The thiolated ligands (gray and orange) are attached to the gold core (yellow) by gold-sulfur bonds. The sulfur atoms are colored orange.

Figure 14: Images are rendered using UCSF ChimeraX. [31]



(a) $t = 1\,\text{ns}$ (b) $t = 150\,\text{ns}$ (c) $t = 300\,\text{ns}$

Figure 15: Hydrophobic drug molecules (bright pink) are repelled by the solvent and are therefore buried among the ligands (grey). The hydrophilic background molecules (bright turquoise) are attracted by water and are therefore forced to the surface of the drug carrier. 15a at the beginning, 15b in the middle and 15c at the end of the simulation. Images are rendered using UCSF ChimeraX. [31]



(a) Radial distribution of the surface molecules with respect to time.

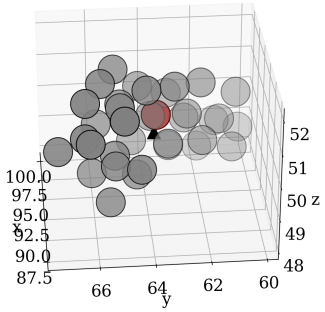
(b) Radial distribution of all molecules (and gold core) with respect to time.

Figure 16: The distance from the drug-carrier center point to each atom, averaged per group. The plot shows how the molecules are radially distributed. As time progresses the drug (OQL) and background molecules (ONC) are separated. The OCQ and OCN are the molecules at the ends of the ligands to which the drug and background molecules are attached, respectively.

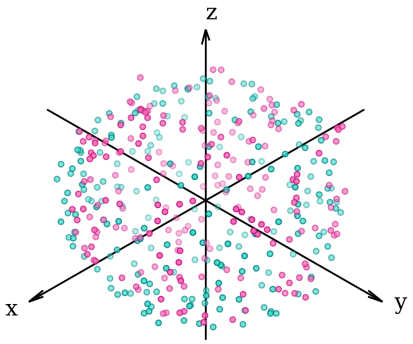
3.2 Data Preprocessing

The aim of the experiment is to explore whether or not there are some simple patterns present in the motion of the active substances (molecules) near the water interface (the drug carrier surface) that could be exploited using an LSTM. The `pdb` files are first preprocessed to produce NumPy files containing the coordinates of each molecule rather the coordinates of each atom. This is done by calculating the geometric center point of each molecule and removing all atoms in each molecule except for the ones closest to the center points. See figure 17a. This reduces the number of atoms from around 40 000-50 000 to approximately 6000. Since this can still be prohibitively many particles when each particle has multiple features, and because the data exploration shows that mainly the drug and background molecules determine the final spatial conformation of the drug carrier, all but the drug and background molecules are removed. Additionally, all water molecules are removed since these act as a heat bath and so the motion of individual water molecules is unnecessary to model. The result of preprocessing a single `pdb` file so far is visualized in figure 17b. In order to obtain a time series, all 300 `pdb` files of a MD simulation are preprocessed

similarly.



(a) Each circular marker represents an atom in a molecule. The small black triangular marker represents the geometric center of the molecule. The red marker is the atom closest to the geometric center.



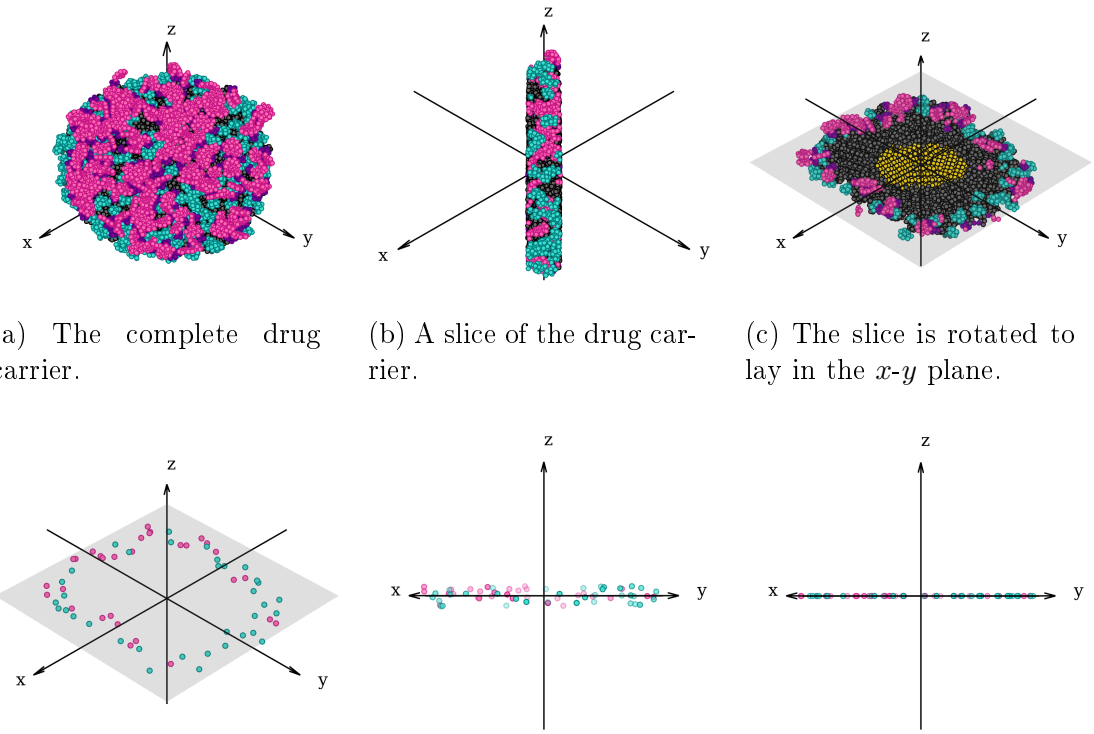
(b) The center atoms of all drug and background molecules in a drug carrier.

Figure 17

I addition to the coordinates of the center atoms, the molecule type will also be used as a feature. The molecule name is encoded using a simple one-hot encoding.

Two different datasets are created, one that is two-dimensional (2D) and one that is three-dimensional (3D). The 3D dataset is what has been described so far. The samples in the 2D dataset are created by taking geometric slices of the drug carriers in the 3D dataset. A slice is taken at the initial time step t_0 along an arbitrary axis d that goes through the same center of the drug carrier by removing all molecules that do not fall within a distance $\pm\Delta d$ along the axis. See figure 18. I order to ensure that the same center point is always used, the drug carrier's centermost gold atom (the same atom is always chosen) is aligned with origin. Multiple slices are taken at different angles so that the whole 3D drug carrier is used when producing slices. At the following time steps t_n the slices are taken differently. All particles but the ones selected at t_0 are removed. This way, complete trajectories are obtained and can be used as time series. Because the trajectories happen to be such that the slices remain thin throughout all 300 ns the slices can be flattened into two dimensions without loss of much information. The flattening is done by simply removing the coordinate

along the thickness axis d. Finally all slices are rotated to lie in the x-y plane. See figure 18 for a visual description of this process.



(a) The complete drug carrier.

(b) A slice of the drug carrier.

(c) The slice is rotated to lay in the x-y plane.

(d) Only the centers of drug and background molecules are selected.

(e) 18d viewed in the x-y plane

(f) The slice is flattened.

Figure 18

The preprocessed files are passed to the model through an efficient input pipeline that creates sliding windows of shorter sequence lengths than the complete 300 time steps. These shorter sequences are then split into an input and a target sequence. The input sequence will be used to predict the target sequence. Additionally the input pipeline creates augmented copies of the sequences through random rotations of the drug carrier (the drug carrier is rotated the same amount at all steps in a sequence).

To summarize, the final temporal sequences consist of multiple time steps. Each time step consists of a set of features describing every molecule at that time instant. The features are the coordinates of each molecule's center atom and the one-hot encoded

molecule name.

The dataset is divided into training and validation sets having 70% and 30% of the samples respectively. Since the sliding window causes the same data point to be included in multiple sequences (at different relative time steps), the divide is made in such a way that the training set and validation set are guaranteed to be non-overlapping, thereby ensuring that any datapoint in the validation set is never observed during training, and vice versa.

3.3 Model

The model takes as input temporal sequences of length T and predicts sequences of length T' that are compared to the target sequences. The input sequence is passed to a time distributed multilayer preceptron (MLP), followed by a set of LSTMs, followed by another MLP. The output is a single-step prediction and an LSTM state for the time step $T + 1$. This will be called the initialization phase as during it the LSTM internal states accumulates information from the whole input sequence before a first prediction is made. The prediction and the LSTM state at time step $T + 1$ are used as input at time step $T + 2$, and so on until the last step T' has been reached. The exact number of layers and nodes were varied in search for an optimal model. An overview of the architecture is shown in figure 19.

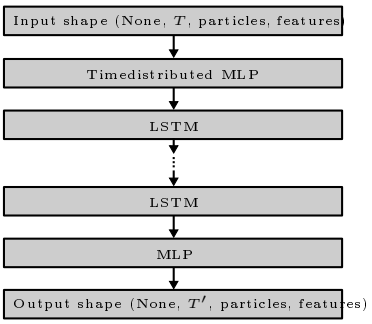


Figure 19

3.4 Result

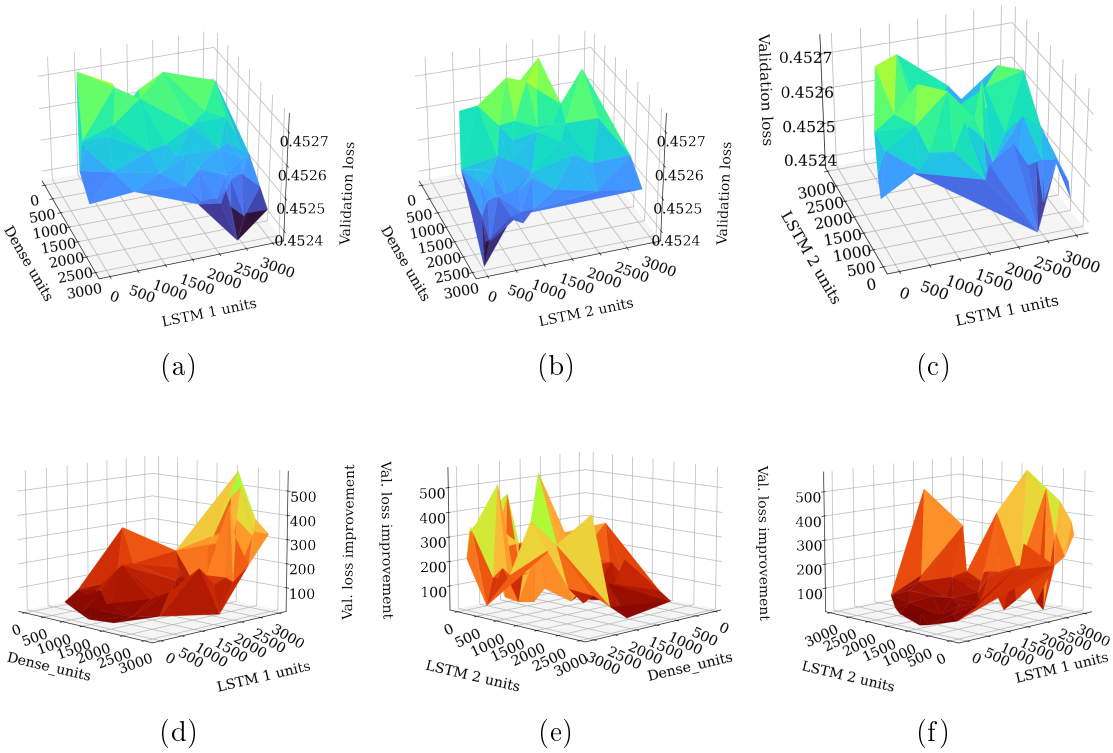
The models were trained for a maximum of 30 epochs with early stopping to reduce overfitting. The sequence lengths were $T = 25$ and $T' = 125$. Multiple models with different hyper parameters were trained. Dropout was applied to the hidden perceptron layers during training. The hyperparameters were selected using Bayesian optimization with the tree-structured Parzen estimator. [32] The hyperparameters were

- the number of units in the layers of the first MLP,
- the number of units in the LSTMs, tuned separately per LSTM,
- the number of units in the second MLP.

A model with two sequential LSTMs had the best performance. A summary of all trained variations of that model is given in figure 20. Any of the trained models were unable to produce meaningful results regardless of whether the 2D or 3D dataset was used. All models fell into one of two categories when evaluating them on the validation set. Either the predicted particle positions would converge into a single configuration regardless of the input, or the particles would oscillate between two meta stable configurations, as seen in figure 21. This suggest that the model in combination with the training algorithm is only able to find the static values for which the loss is small, rather than modeling particle trajectories. There are a few potential reasons for this. 1) The drastic simplifications made in the preprocessing may obscure the information necessary for modeling the motion of the molecules. 2) There is no simple pattern present in the motion of the molecules. 3) The model does not have the capability of expressing the motion. 4) A combination of 1-3.

Alternative 4 is the most likely. Since the molecules in the drug carrier are large and tightly packed their geometric shapes are of importance in order to accurately model their motion. Furthermore the molecules form long chains by forming chemical bonds which further constrain their motion. These properties are not explicitly present in the preprocessed data and may obscure much of the information necessary

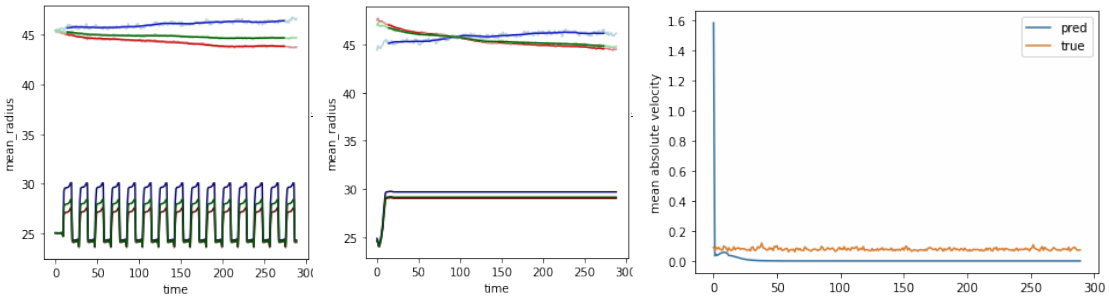
for deriving some rules for the motion. The motion of the molecules is very chaotic but it has a general trend, as described in section 3.2. However, this trend is a result of a stochastic process and should therefore be treated accordingly. The model is very simple and more care should be put into formulating a more sophisticated architecture that has the capability of modeling the drug carrier properly.



(a) (b) (c)

(d) (e) (f)

Figure 20: Validation loss surfaces with respect to hyperparameters. Figures 20a-20c show that the validation loss at the end of training is essentially unaffected by changes in the hyperparameters. Figures 20d-20f show the amount with which the loss decreased during training, with respect to the hyperparameters.



(a) Predicted molecule positions oscillate between two meta stable states.

(b) Predicted molecule positions stays in a single configuration.

(c) The mean absolute velocity of the same molecules as in 21b shows that the predicted motion has truly stagnated.

Figure 21: Predictions on the validation set. Figures 21a and 21b: The mean radial distance (calculated from position predictions) from each molecule to the drug carriers center. Red is drug molecules, blue is background molecules, and green is the mean weighted by the ratio in molecule counts. The smaller distances (bottom) are predictions, the larger distances (top) are true values.

4 A Second Experiment

The shortcomings of the first experiment led to the development of the model whose components are the main focus of the theory sections 2.2-2.5. An attempt at addressing some of the issues associated with the information loss due to preprocessing of the MD simulations is made. The model is tested on a relatively straightforward problem of predicting two-dimensional trajectories of interacting particles. The problem is approached by modeling the particle-particle interactions, rather than trying to model the trajectories directly as in the first experiment. As described in section 2.3, the particle-particle interactions are allowed to be arbitrary. The machine learning algorithm has the potential to find these arbitrary rules for the interactions by not being programmed with any domain knowledge of the nature of these interactions.

The core idea behind this approach is that the molecules do not undergo chemical reactions and interact mainly through electromagnetic forces and collisions. Generally, molecules have some charge distribution and can therefore be approximated in terms of a multipole expansion, as described in section 2.6. This means that over long distances, compared to the size of the molecule, the electric field can be approximated as a monopole field, meaning that the molecule itself can be approximated as a point-like particle in terms of the charge. A practical example of the monopole approximation being successfully used in a similar deep learning approach to molecular dynamics can be seen in the work of Wang et. al. [3] On shorter distances one must also account for the dipole moments, and eventually, as the intermolecular distance is further decreased the quadrupole and higher order moments must also be considered. [27] It is therefore convenient to first test the model using only the monopole approximation as a proof of concept, which is what this experiment will focus on. In order to directly measure the model's performance on modeling the intermolecular forces, only electrostatic interactions are considered, no collisions.

The hypothesis is that when irregularly shaped molecules are approximated as point-like particles (or dipoles, etc.), as described by section 2.6.1, the model could potentially account for the missing information about the higher order moments and the

missing geometric shape information by modeling the corrections for these using the LSTMs. By observing the motion of a molecule (including collisions) together with the trajectories of the neighboring molecules the LSTM could perhaps infer some (abstract) representation of an approximate charge distribution or geometric shape. This is left as a suggestion for future work to test.

Multiple many-body systems are simulated and used as training data. Trajectories of simulated particles are obtained by solving the classical equations of motion numerically, using velocity Verlet integration. The model's loss on the simulated trajectories is minimized with respect to the layer weights, using gradient descent. That is, the model is tasked with predicting the trajectories of simulated particles in a many-body system and the optimization algorithm is tasked with minimizing the prediction error of the model.

4.1 Newtonian Many-Body Mechanics

Assuming an inertial reference frame, the motion of the i:th particle in a many-body system is given by the second order ordinary differential equation

$$\mathbf{F}_i(\mathbf{r}_i(t)) = m_i \frac{\partial^2 \mathbf{r}_i(t)}{\partial t^2} \equiv m_i \ddot{\mathbf{r}}_i(t) \tag{44}$$

where $\mathbf{F}_i$ is the net force the i:th particle is subject to and $\ddot{\mathbf{r}}_i(t)$ is the acceleration of the i:th particle due to $\mathbf{F}_i$. This is Newton's second law of motion. In this inertial reference, frame Newton's third law, stating that two directly interacting particles exert equal and opposite forces on each other, will also hold. [4]

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji} \tag{45}$$

[4]Generally, because the force field propagate with a finite velocity (usually the speed of light), forces will not be opposite. Therefore, Newton's third law is only approximate. However, in this thesis, the magnitude of the particle velocities are taken to be sufficiently small when compared to the relative absolute distances between particles so that the approximation errors are negligible when the forces are taken as being instantaneous. [33, 34]

This allows us define the force as a function of the coordinates, and equation (44) will correctly describe the motion of a single particle. [33]

For the machine learning model, this means that the motion of each particle i can be modeled independently at every time step. Thus, only the forces $\mathbf{F}_{ij}$ due to particles $i \neq j$ need to be considered when modeling the trajectory of the i:th particle. Furthermore, only the change in position of the i:th particle relative the other particles affect the force $\mathbf{F}_{ij}$, making state pooling a reasonable method for describing a neighborhood with many particles. Therefore, solving Newtonian many-body problems is a suitable experiment for testing the machine learning model. By not programming any domain knowledge about the definition of the particle-particle interactions $\mathbf{F}_{ij}$ into the model, other than that required by state pooling, the machine learning algorithm is tasked with deriving some rules for the particle-particle interactions given the trajectories $\mathbf{r}_{ij}(t)$.

4.2 Data Acquisition Using Velocity Verlet Integration

The simulated many-body systems consist of N interacting particles. The N particles are initialized in states with random positions $\mathbf{r}_i$, random velocities $\dot{\mathbf{r}}_i \equiv \frac{\partial \mathbf{r}}{\partial t}$, random masses m_i and additional system-specific randomly initialized variables, like charge for example.

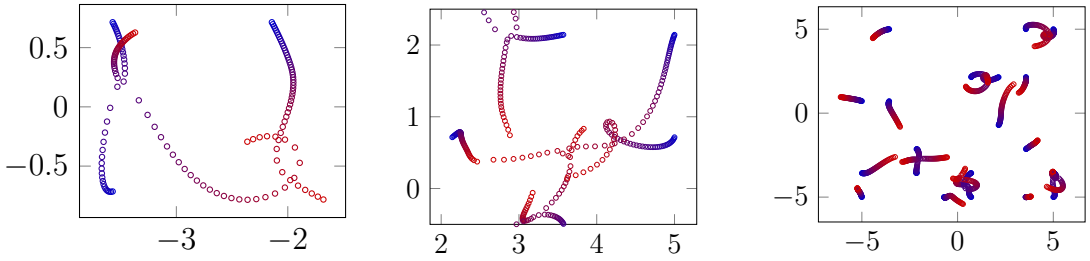
Simulating trajectories $\mathbf{r}_i(t)$ is equivalent to numerically solving the equation of motion given by (44). Given initial conditions $\mathbf{r}_i(t_0)$ and $\dot{\mathbf{r}}_i(t_0)$ for the position and velocity respectively, an approximate solution at time $t_1 = t_0 + \Delta t$ is obtained by assuming constant acceleration during the time interval Δt.

$$\mathbf{r}_i(t_1) \approx \mathbf{r}_i(t_0) + \dot{\mathbf{r}}_i(t_0)\Delta t + \frac{1}{2}\ddot{\mathbf{r}}_i(t_0)\Delta t^2 \tag{46}$$

An approximate solution to the position at some arbitrary number of time steps Δt into the future is obtained by the following algorithm.

1. Calculate the next position $\mathbf{r}_i(t_{n+1}) \approx \mathbf{r}_i(t_n) + \dot{\mathbf{r}}_i(t_n)\Delta t + \frac{1}{2}\ddot{\mathbf{r}}_i(t_n)\Delta t^2$
2. Calculate the next acceleration $\ddot{\mathbf{r}}_i(t_{n+1}) \approx \frac{\mathbf{F}_i(\mathbf{r}_i(t_{n+1}))}{t_{n+1}}$
3. Calculate the next velocity $\dot{\mathbf{r}}_i(t_{n+1}) \approx \dot{\mathbf{r}}_i(t_n) + \frac{1}{2}(\ddot{\mathbf{r}}_i(t_n) + \ddot{\mathbf{r}}_i(t_{n+1}))\Delta t$
4. Repeat from 1

This is the velocity Verlet method of numerically integrating (44) in order to solve for $r_i(t)$. Examples of approximate solutions obtained by velocity Verlet integration are shown in figure 22. Solutions like these are used as training data in the experiments.



(a) (b) (c)

Figure 22: Examples of particle trajectories used as training data. The trajectories are obtained by solving Newton's equations of motion using velocity Verlet integration. The particles interact through a force that is inversely proportional to the square of the relative vector distance between the particles.

4.3 Data Preprocessing

The simulated trajectories are saved as NumPy arrays in `npy` files. A single simulation produces an array of size $T_s \times N \times F$, where T_s is the simulation length in number of time steps, $N = 25$ is the number of particles and $F = 4$ is the number of features. The simulations were run for 1 000 000 iterations with a time step size of $\Delta t = 10^{-7}$ time units. The magnitude of 1 unit of time is arbitrary. Of the 1 000 000 time steps only every 1000:th step was kept, since the step size used in the numerical integration is very small (in order to give accurate solutions). The resulting sequences consist of 1000 time steps with the coordinates, masses and charges of 25 particles at each

time step.

Similar to the preprocessing described in section 3.2, an efficient input pipeline that creates pairs of shorter input and target sequences, and creates augmented copies by random rotations is used. Additionally, at the beginning of the input pipeline only every 20:th time step of a sequence is kept, thereby further reducing the sampling frequency so that a complete simulation is reduced from 1000 steps to 50 steps. As before a sliding window is used for sampling the input and target sequences from the complete simulation sequence. The sliding window is shifted by 5 time steps between each sampling to ensure that the samples are not too similar. Input sequence lengths $T = 5$ and target sequence lengths $T' = 30$ are used. From each training sample (sequence), 10 augmented copies were made. A total of 100 simulations make up the training set and 20 simulations make up the validation set. Since modeling decisions were made based on the result on the validation set, an additional 20 files were set aside as an unbiased test set. This results in the training set consisting of 20 000 samples, and the validation and test sets of 4000 samples each.

4.4 Model

The model predicts one time step at a time. At each time step the input is the coordinates, charges and masses for all particles. The output is a set of Gaussian probability density functions (see section 2.5) describing the predicted coordinates one time step into the future. Each particle is described by a separate LSTM (state) but all LSTMs share the same weights. Stacked state pooling is used with embedded memory states, followed by a small CNN, followed by a single-layer perceptron (SLP). The perceptron output is concatenated with a single particle's embedded features and passed as input to the LSTM. The LSTM's updated memory state is transformed by a small SLP into parameters for a Gaussian probability density. An overview of the model is shown in figure 23, and a conceptual description of the state pooling module in figure 24.

The model is iterated over an input sequence $\mathcal{X}_{\text{in}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ in order to pre-

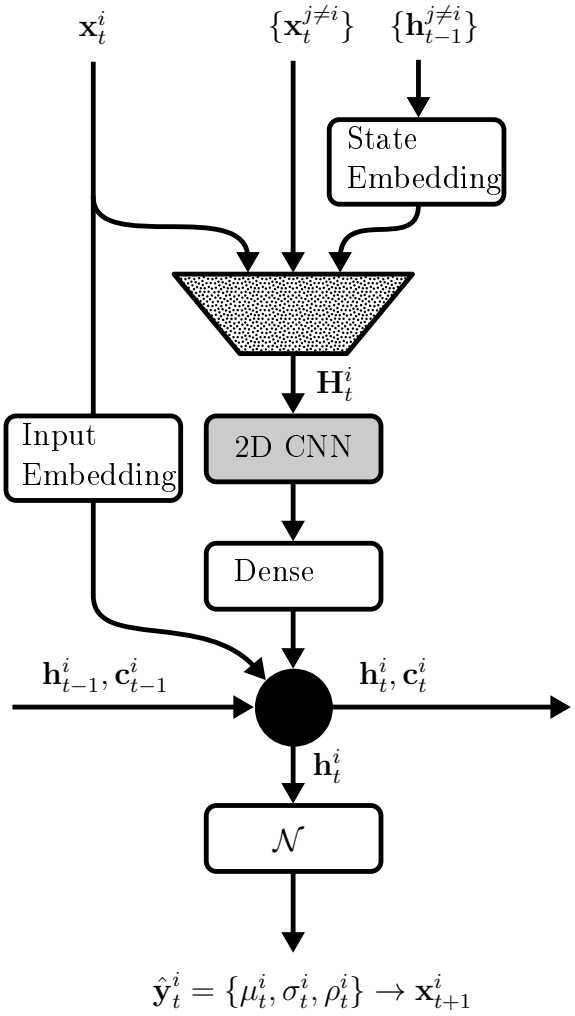


Figure 23: A schematic description of how a single-particle prediction is made at a single time step. The pooling tensor $\mathbf{H}_t^i$ is constructed from the previous memory states and the current features (position) $\mathcal{X}_t \equiv \mathcal{Y}_{t-1}$. The LSTM (black disk) takes as input its particle's features $\mathbf{x}_t^i$, the result of the pooling procedure, and the previous states.

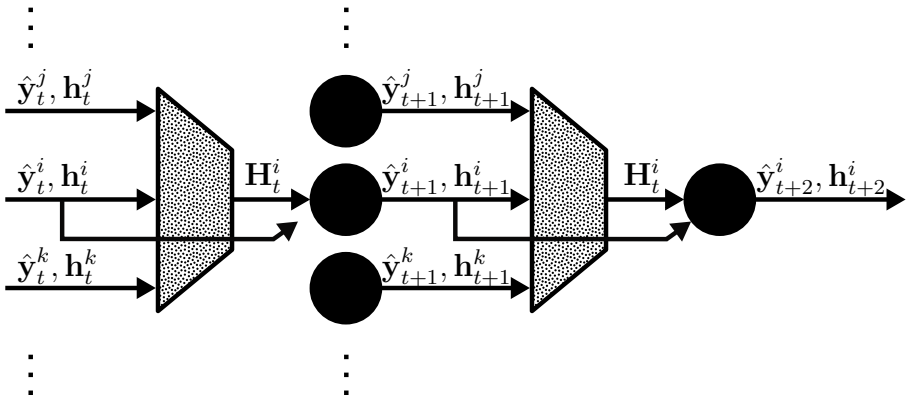


Figure 24: A simplified description of the model, showing how predictions are made for a single particle i, two time steps into the future. The predictions for the other particles are done similarly. At each time step the layers (black disks) can share weights. For simplicity, no LSTM cell states $\mathbf{c}_t^i$, only memory states $\mathbf{h}_t^i$, are shown.

pare the LSTM states (see section 2.5.2), after which an output sequence $\mathcal{Y} = \{\mathbf{y}_T, \ldots, \mathbf{y}_{T+1}\}$ is produced in an iterative manner, as shown in figure 25. The outputs are predictions of the particle positions $\mathcal{X}_{\text{target}} = \{\mathbf{x}_{T+1}, \ldots, \mathbf{x}_{T+T'}\}$ and are compared to these using the negative logarithmic likelihood as the error function (see section 2.5).

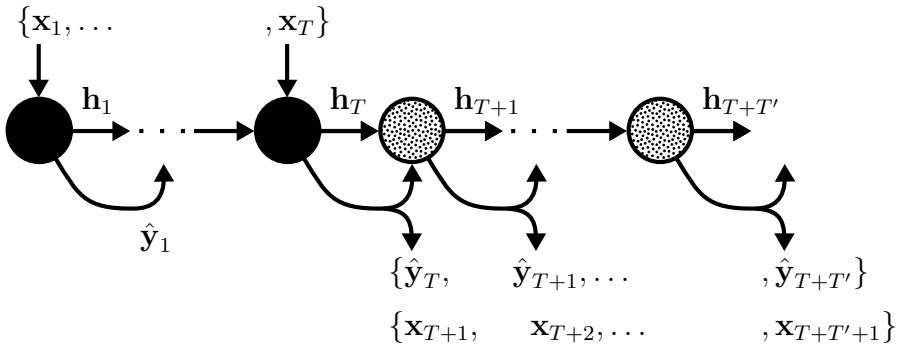


Figure 25: During the initialization phaze, the LSTM memory state $\mathbf{h}_t^i$ and cell state $\mathbf{c}_t^i$ are accumulating information for T iterations, after which the first prediction $\mathbf{y}_T$

The layer that embeds the memory states (state embedding) before the state pooling, and the layer that embeds the i:th particle's features (input embedding) before they

are passed to the i:th LSTM, are simple linear transformations

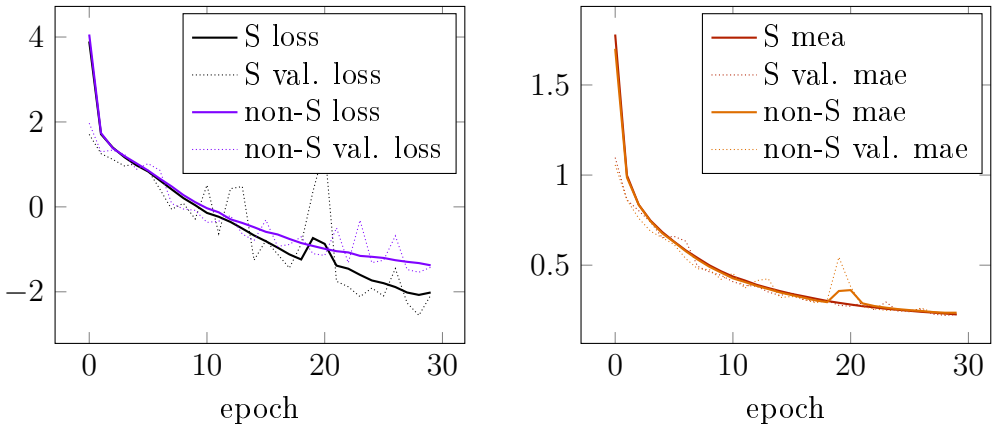
$$\boldsymbol{\alpha}_t^i = \mathbf{W}_\alpha \mathbf{x}_t^i \tag{47}$$

$$\boldsymbol{\beta}_t^{j\neq i} = \mathbf{W}_\beta \mathbf{h}_t^{j\neq i} \tag{48}$$

with an embedding dimension of 32. The stacked state pooling is implemented according to section 2.3 using 4 separate pooling tensors that are summed together. The pooling grids all have an identical 32×32 shape, but each correspond to a different physical size, with the relative side lengths 1, $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{8}$ respectively. Specifically, he largest pooling grid is chosen to have a side length of 11 length units, but this value is specific to the data set and must be chosen on a case by case basis. The value of 11 length units is approximately the same distance as the largest distances between any two particles seen in the data set, and it is equivalent with the smallest pooling grid having a spatial resolution of approximately 0.04 length units, which is of the same order of magnitude as some of the smallest inter-particle distances in the data set. Using stacked state pooling reduces the required amount of computer memory needed for training and also speeds up the calculations. As a comparison, training the model using 4 stacked pooling tensors (multi-grid model) of shape 32×32 took only 10 hours compared to the 17.5 hours needed for training the same model with only a single pooling grid (mono-grid model) of shape 64×64. [5] The errors were similar between the two models, as shown in figure 26, with the multi-grid model having a more stable loss on the validation data set than the mono-grid model. Furthermore, the mono-grid size was limited by the GPU memory, whereas the multi-grids are only a $\frac{1}{4}$ the size of the mono-grid. The multi-grid model is chosen because of the performance and stability benefits, especially because of the memory benefits.

The two-dimensional CNN has 3 consecutive layers, each with a ReLU activation function, and kernels with 16, 16, and 32 units respectively. Filters sizes of both 2×2 and 4×4 were tested on a limited data set. The performance of both were identical, as shown figure 27. The larger filters were chosen in order to increase the

[5] Note that the performance benefits of a smaller state pooling tensor does not only come from the pooling calculations but also from calculations in the layer(s) following the state pooling.



(a) Negative logarithmic likelihood loss during training.

(b) Mean absolute error during training.

Figure 26: Comparison of using stacked (S) state pooling with four pooling grids of size 32×32, and a single pooling grid (non-S) of size 64×64. The multi-grid model uses less memory and is trained approximately twice as fast as the mono-grid model.

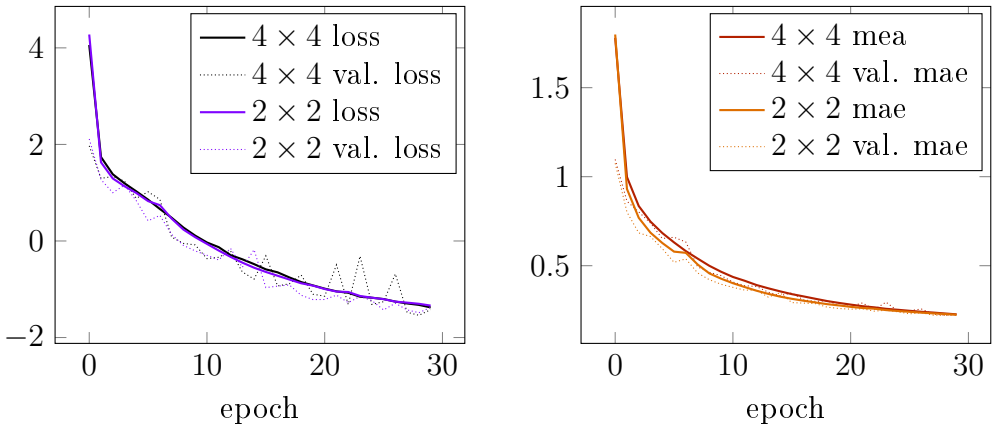
"field of view" of the kernels.

The SLP that embeds the CNN output $\mathbf{g}_t^i$ before it is passed to the LSTM is a simple non-linear transformation, with an embedding dimension of 32, having a ReLU activation function.

$$\boldsymbol{\gamma}_t^i = \text{relu}(\mathbf{W}_\gamma \mathbf{g}_t^i) \tag{49}$$

The the output layer predicts parameters for a bivariate Gaussian distribution and is thus a 5 unit SLP with several activation functions given by equations (21), (22) and (23), as described in section 2.5. In the case of the mixture density model, 5 mixture components without correlation are used, see equation (16). For each mixture a mixture (weighing) coefficient as (31) must be predicted. Thus, the output layer of the mixture density model has 25 units.

Because of the way the model is built to calculate single particle positions, and because all particles should abide the same laws of motion, all layers share their weights between all particle instances (in the same way as the LSTMs share their weights).



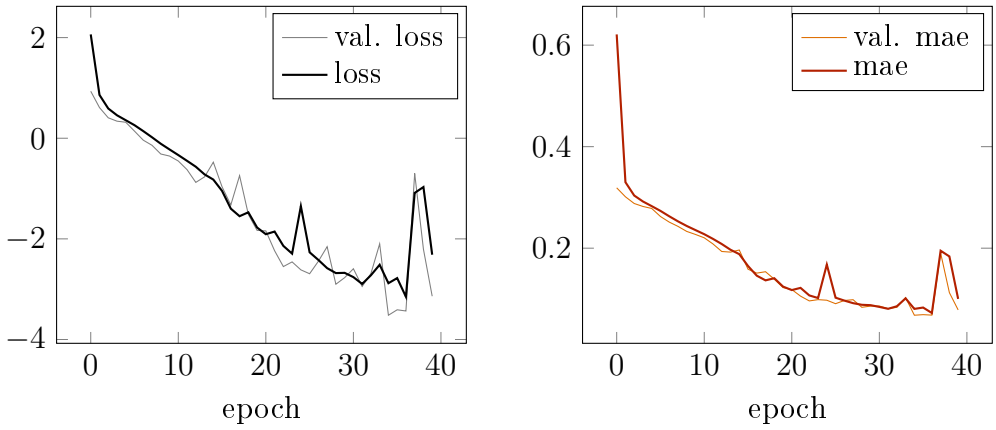
(a) Negative logarithmic likelihood loss during training.

(b) Mean absolute error during training.

Figure 27: Comparison of using different sized filters in the CNN layers. There is no significant difference in the errors.

4.5 Result

The model was trained for a maximum of 50 epochs with early stopping to reduce overfitting. The input sequences were of length $T = 5$ steps and the target sequences of length $T' = 30$ steps. Because of the shared layers, the model had merely 140 768 trainable parameters. The prediction error was minimized by stochastic gradient descent using the Nesterov-accelerated adaptive moment estimation (Nadam) algorithm with a learning rate $\eta = 0.001$, the decay rates $\mu = 0.9$ and $\nu = 0.999$, and the small constant $\varepsilon = 10^{-7}$ that is only used in divisions for numerical stability. [35] With the single-density model an error, as given by equation (33), of -3.4332 and a mean absolute error of 0.0681 were reached on the validation set. The training loop was ended after 40 epochs when the smallest error had not improved in four epochs. The errors are plotted in figure 28. The mixture density model, suffered from several bugs that hindered it to be trained properly. It is therefore left as future work to compare the performance of the mixture density model to the single-density model. Theoretically, a mixture density model with a large number of components should be able to model the trajectories better than a single-density model, as a mixture density is a universal approximator [25, 26]. Though, how well the models (with a limited number of mixture components) perform in practice on a specific and finite data set must be measured in experiments.



(a) Negative logarithmic likelihood. (b) Mean absolute error.

Figure 28: Early stopping terminated the training loop after 40 epochs.

Analyzing the paths that are predicted by the single-density model reveals how well-behaved the model is. Looking at the less successful predictions indicates where there is room for improvement. In this analysis more emphasis will therefore be set on analyzing why the less successful predictions fail. The following analysis is done using the unbiased data set that was set aside in section 4.3.

In figures 29 - 30, predictions on the unbiased test set are compared to the corresponding target trajectories. An overview of three different regions in the many-body systems are shown in figure 29. A visual inspection of the regions confirms that the mean absolute error of approximately 0.01 correspond to overall good predictions. In figure 30, three (mainly) long-long range interactions are presented, long-range being of the order of magnitude 1. The long-range interactions cause slow changes in the direction of the particles velocity, meaning the trajectories make wide curves. These interactions the model predicts well. Many times, the short-range interactions, as shown in figure 31, look physically realistic as if there only was a stronger attractive force than what was used in the MD simulations, leading to steeper approaches in the predicted trajectories. A closer inspection of more samples of close approaches reveals another behavior. Particles that are attracted by each other may approach too slowly or they may turn away from each other before they meet, see figure 32. The latter would be realistic if there was an attractive force proportional to r^{-1} an

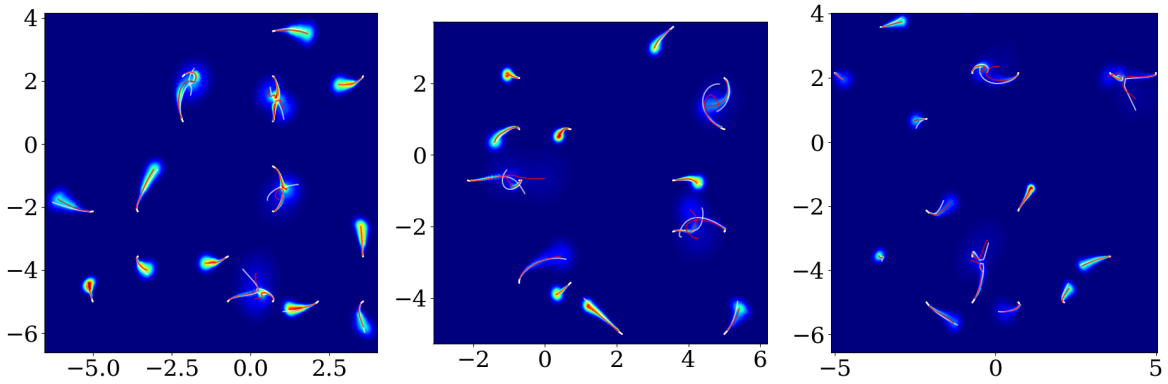
a repulsive force proportional to r^{-2}, leading to the net force being attractive over large distances and repulsive over small distances. However, this is not the case. The simulated systems have a Coulomb force and a gravitational force, both which are proportional to r^{-2}. Thus, there must be something else causing these effects.

In figure 33 a force driving the particles away from each other seem to exist although no such force is present. [6] This is due to subsampling of the true trajectories as part of the data input pipeline. As the particles' trajectories never appear to cross each other, another level of difficulty is added to the modeling task. In the simulations the particles' velocities have changed direction because the particles have "passed by" each other when making a partial orbit. As this pass-by is not apparent in the training samples, due to subsampling, the model must learn to correctly add a correction that accounts for this missing information, which adds another level of difficulty. Based on the samples in figures 32c, 33, 34b and 34c, this correction must be such that when attractive particles approach each other, instead of making a partial orbit as real (simulated) particles would, the particles make an early steeper curve, so that they never meet, and so that the loss is minimized. This way the model would learn to always make slightly steeper curves than in reality (e.g. by adding what may be interpreted as a force component perpendicular to the velocity in the direction of the focus of the current trajectory), and thus also leading to predictions like those seen in figure 33.

Despite the too sparse subsampling, the overall modeling accuracy is acceptable. More frequent subsampling of the time series should be used when building the data sets. In all examples given in figures 29-34 the predicted likelihoods are significantly more localized (smaller standard deviations) before a close approach than after. This pattern is present throughout all predictions. This could in part be a realization of the limitations described in section 2.3.4 and in part a consequence of too sparse subsampling. It could likely be improved by a combination of more frequent subsampling and adjusting the relative sizes and resolutions of the state pooling grids.

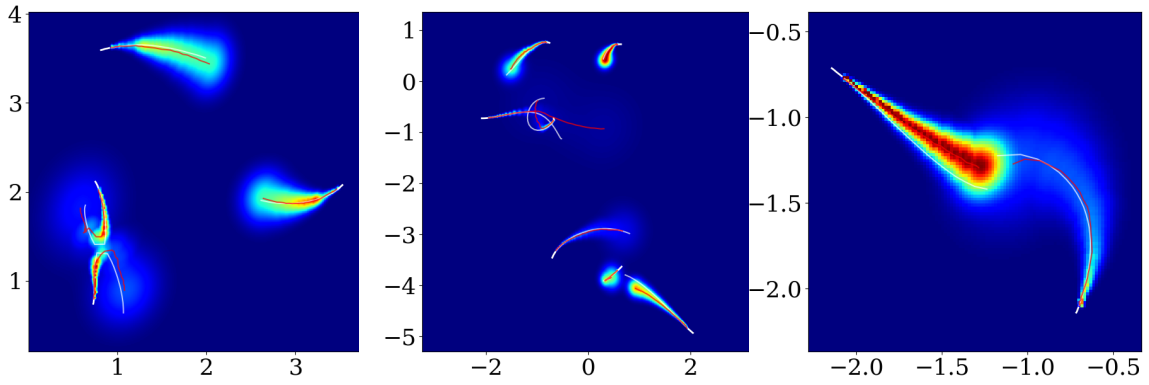
[6] A repulsive force only in the direction of the line connecting the particles would not cause these trajectories though.

Additionally, proper hyperparameter tuning of the model should be performed in order to quantify which hyperparameters are the best [36].



(a) (b) (c)

Figure 29: An overview of predictions in three different regions over the course of 30 steps. Target trajectories are white, predicted trajectories are red, and predicted probability densities are represented with a color gradient.

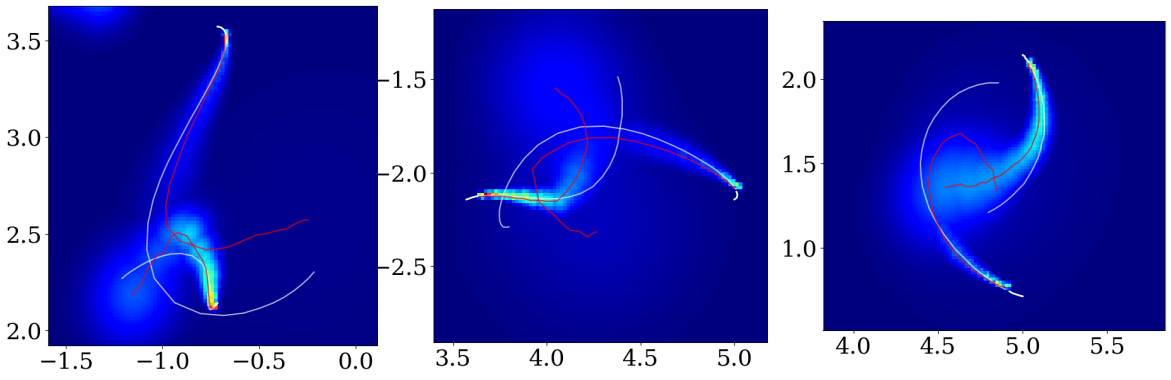


(a) Long range interaction (upper right) and short-range interaction (lower left)

(b)

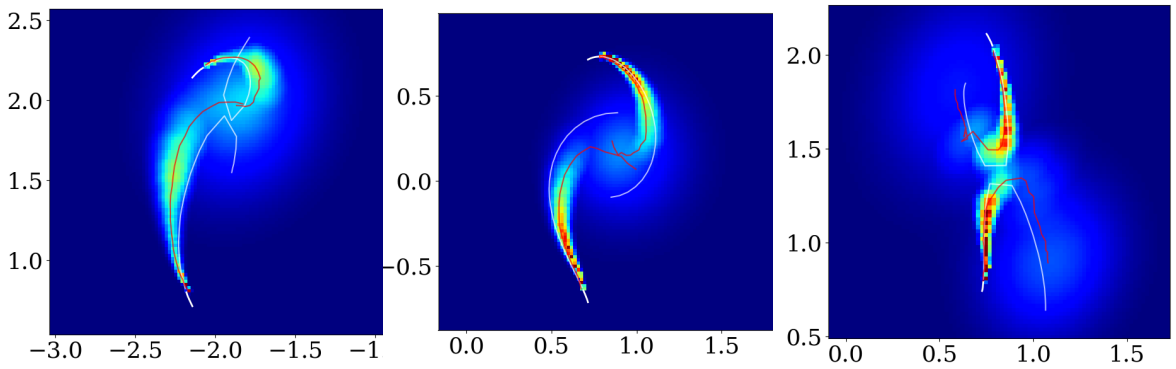
(c) Particles of different masses interacting. The lower right particle is the lighter of the two.

Figure 30: Some typical long-range interactions. 28b



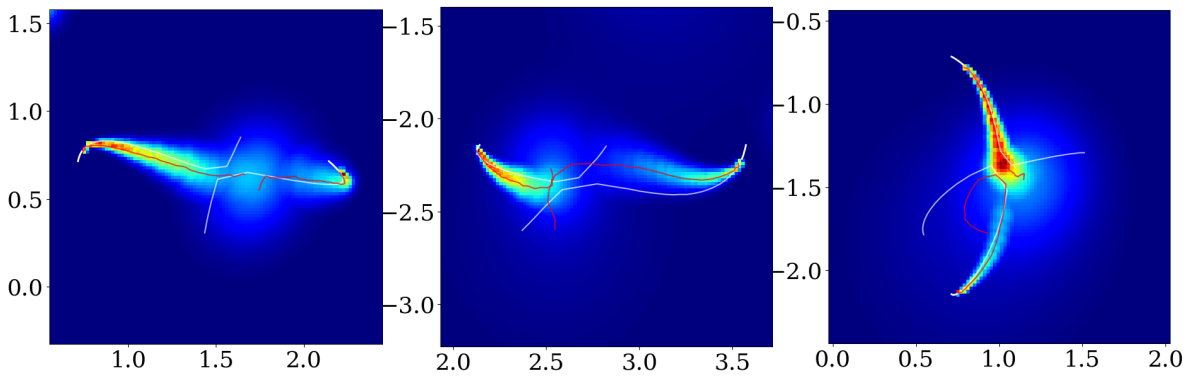
(a) (b) (c)

Figure 31: As particles get close to each other, interactions that appear as if the strength of the attractive force is overestimated are typically observed.



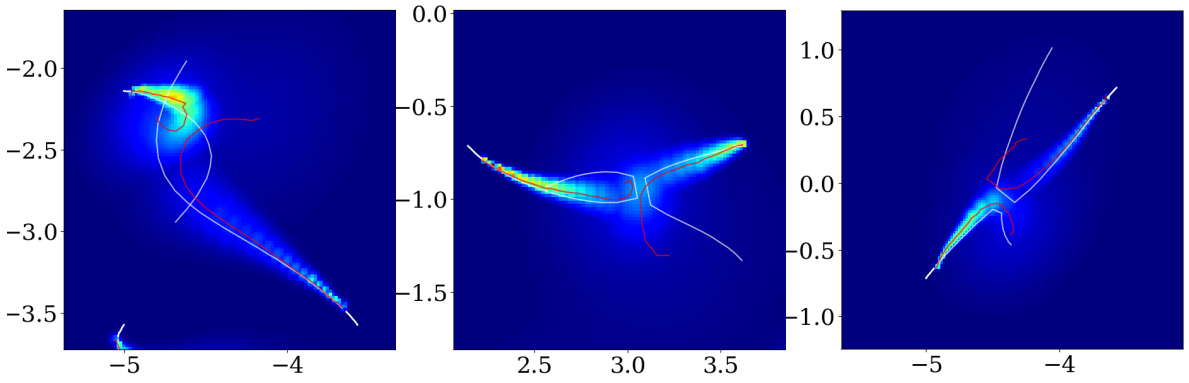
(a) (b) (c)

Figure 32: Predictions may have too slow velocity such that the travelled distance is too short, as in 32a, or they may make premature changes in the direction of travel.



(a) (b) (c)

Figure 33: Too infrequent subsampling make it seem as if the true particle trajectories (white) never cross, even though they in reality do.



(a) (b) (c)

Figure 34: Because of sub-optimal training data, the model has been optimized to make incorrect trajectory corrections causing the particles to never meet.

5 Conclusions

The aim of this work was to explore if an LSTM-based machine learning model can predict the surface properties of a drug carrying nanoparticle by modeling the dynamics of the drug carrier's surface molecules. An initial attempt was made at modeling the trajectories of the most important surface molecules. This attempt failed due to the following possible reasons.

1. The approximation of large non-symmetric molecules as point-like particles and limiting the observations to only two types of molecules obscures too many important features.

2. There is no simple pattern to learn.

3. The model architecture is not suitable for the task and does not have the capability of being trained to express the motion of the surface molecules.

A second approach was developed to address the shortcomings of the first. In this approach the dynamics of a system of classical particles are modeled by explicitly modeling the particle-particle interactions. For this, LSTM state pooling similar to that used by Alahi et. al. [2] but with some key differences is used. Specifically the following techniques are introduced to the state pooling.

1. Distance based filtering. The pooling grid is made effectively circular by filtering the LSTM states (the particles) by their radial distance from the center of the pooling grid. See section 2.3.3 and figure 6.

2. Stacked state pooling. Multiple pooling tenors, all of the same $N \times N$ shape but each corresponding to a different physical size, are combined using a summarizing statistic. In this thesis, that summarizing statistic is a simple sum. See section 2.3.5 and figure 8

3. A CNN is used for processing the pooled states (i.e. the pooling tensor $\mathbf{H}_t^i$) before using them as input to the LSTM.

Stacked state pooling showed that multiple small pooling grids give similar results as using a single large pooling grid. Furthermore, using stacked state pooling, instead

of a single large pooling grid, requires less GPU memory and reduced the training time from 17.5 hours to 10 hours. See section 4.4

In order to test the model from the ground up, beginning from the lowest order approximation of the charged particles, as described by section 2.6, the model is tasked with solving the equations of motion for a many-body problem consisting of gravitationally- and electrostatically-interacting monopolies . Overall the model performs well and demonstrates the proof of concept the experiment was set up for. Artifacts in the position predictions of closely approaching particles is observed. The artifacts show patterns that indicate that the they are likely caused by too sparse subsampling of the simulated particle trajectories when building the training data set.

Additionally, the predicted likelihood estimates for the future positions are significantly less localized after close approaches than before. The conclusion is drawn that this is likely due to a combination of finite spatial resolution of the state pooling (see section 2.3.4), which is used for modeling the particle-particle interactions, and too sparse subsampling as described before.

5.1 Suggestions for Future Work

The following tasks are suggested, in no particular order, as future work.

1. Proper methodological hyperparameter optimization should be performed in order to analyze the full potential of the model.

2. The Gaussian mixture model should be evaluated and compared to the results of the non-mixture model presented in section 4.5.

3. The state pooling (section 2.3) should be expanded to tree spatial dimensions.

4. Experiments should be performed to evaluate the the model on tasks where the monopoles are replaced by dipoles. The model must predict both positions and orientations of the dipoles, thereby adding another level of difficulty.

5. Lastly, if 4 is successful, the next step would be to model quadrupoles, before moving to the original problem of modeling the surface molecules in a drug carrying nanoparticle. The model presented in this thesis has no mechanism specifically designed to consider inter-molecular bonds, such as those between the ligands and drug molecules, that restrict the motion of a molecule. A straightforward approach would be to use a simple hybrid model where the ligands and the bulk of the nanoparticle are replaced by simple strands to which the surface molecules are attached, as visualized in figure 35. The machine learning model could be used to model the interaction between surface molecules (and the solvent) and the strands could restrict the molecules radial distance from the drug carrier's core by a simple damped restoring force, much like a simple spring.

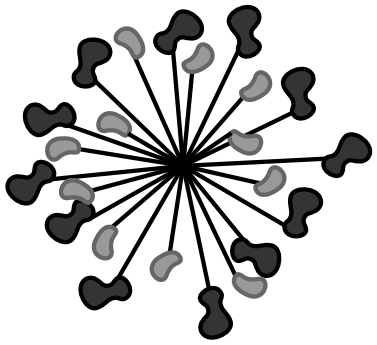


Figure 35: A simplified drug carrier where the motion of the surface molecules are restricted by simple damped restoring forces in the directions of a lines connecting the molecules to the drug carrier's center point.

6 List of Abbreviations

CNN, convolutional neural network

GPU, graphics processing units

LSTM, long short-term memory

MD, molecular dynamics

MLP, multilayer perceptron

ReLU, rectified linear unit

RNN, recurrent neural network

SASA, solvent accessible surface area

SLP, sinlge-layer perceptron

2D, two-dimensional

3D, three-dimensional

References

[1] Alex Graves. Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs]*, June 2014. arXiv: 1308.0850.

[2] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, Las Vegas, NV, USA, June 2016. IEEE.

[3] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. Deepmd-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications*, 228:178–184, 2018.

[4] Antti Pihlajamäki, Joonas Hämäläinen, Joakim Linja, Paavo Nieminen, Sami Malola, Tommi Kärkkäinen, and Hannu Häkkinen. Monte carlo simulations of au38(sch3)24 nanocluster using distance-based machine learning methods. *The Journal of Physical Chemistry A*, 124(23):4827–4836, 2020.

[5] Nils Mönning and Suresh Manandhar. Evaluation of complex-valued neural networks on real-valued classification tasks. *arXiv preprint arXiv:1811.12351*, 2018.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 5. MIT Press, 2016. http://www.deeplearningbook.org, accessed: 01-04-2019.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[8] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[10] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[11] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[12] Parth Kothari, Sven Kreiss, and Alexandre Alahi. Human trajectory forecasting in crowds: A deep learning perspective. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–15, 2021.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *ICML*, 2011.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 10. MIT Press, 2016. http://www.deeplearningbook.org, accessed: 01-04-2019.

[16] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 9. MIT Press, 2016. http://www.deeplearningbook.org, accessed: 01-04-2019.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[20] Coenraad Mouton, Johannes C. Myburgh, and Marelie H. Davel. Stride and translation invariance in cnns. In *Artificial Intelligence Research*, pages 267–281, Cham, 2020. Springer International Publishing.

[21] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv preprint arXiv:1805.12177*, 2018.

[22] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR, 2019.

[23] V Dumoulin, F Visin, and GEP Box. A guide to convolution arithmetic for deep learning. arxiv prepr. *arXiv preprint arXiv:1603.07285*, 2018.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 19.4. MIT Press, 2016. http://www.deeplearningbook.org, accessed: 01-04-2019.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 3. MIT Press, 2016. http://www.deeplearningbook.org, accessed: 01-04-2019.

[26] Christopher Bishop. Mixture density networks. Technical Report NCRG/94/004, January 1994.

[27] John R. Reitz, Frederick J. Milford, and Robert W. Christy. *Foundations of Electromagnetic Theory (4th Edition)*. Addison-Wesley Publishing Company, USA, 4 edition, 2008.

[28] J. J. Sakurai and Jim Napolitano. *Modern Quantum Mechanics*. Cambridge University Press, 3 edition, 2020.

[29] Marina Kovacevic, Igor Balaz, Domenico Marson, Erik Laurini, and Branislav Jovic. Mixed-monolayer functionalized gold nanoparticles for cancer treatment: Atomistic molecular dynamics simulations study. *Biosystems*, 202:104354, 2021.

[30] D.A. Case, K. Belfon, I.Y. Ben-Shalom, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, G. Giambasu, M.K.

Gilson, H. Gohlke, A.W. Goetz, R. Harris, S. Izadi, S.A. Izmailov, K. Kasavajhala, A. Kovalenko, R. Krasny, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, V. Man, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, A. Onufriev, F.Pan, S. Pantano, R. Qi, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. Simmerling, N.R.Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, L. Wilson, R.M. Wolf, X. Wu, Y. Xiong, Y. Xue, D.M. York and P.A. Kollman (2020). Amber 2020, university of california, san francisco.

[31] Eric Pettersen Tom Goddard Greg Couch Elaine Meng Scooter Morris Thomas Ferrin, Conrad Huang. Ucsf chimerax, developed by the resource for biocomputing, visualization, and informatics at the university of california, san francisco, with support from national institutes of health r01-gm129325 and the office of cyber infrastructure and computational biology, national institute of allergy and infectious diseases.

[32] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation, 2011.

[33] G.R Fowles. *Analytical Mechanics, Third Edition*, chapter 2. Holt, Rinehart and Wilson, 1977.

[34] J. J. Sakurai and Jim Napolitano. *Modern Quantum Mechanics*, chapter Quantum Fields, Relativistic Quantum Mechanics. Cambridge University Press, 3 edition, 2020.

[35] Timothy Dozat. Incorporating nesterov momentum into adam. 2016. https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ, accessed 2021-06-06.

[36] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings*

of the 30th International Conference on Machine Learning, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.