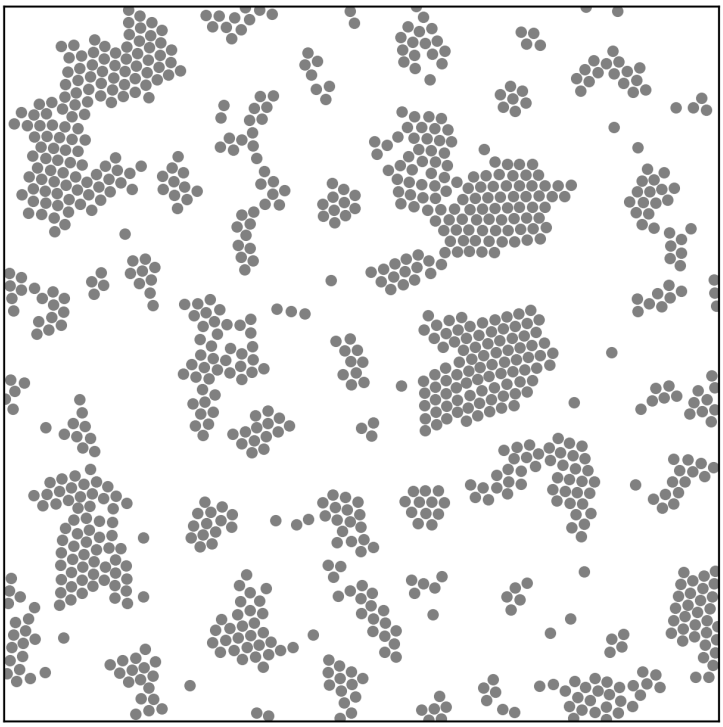


Characterizing the Dynamics of Active Matter using Graph Neural Networks

Master's thesis in Complex Adaptive Systems

CHRISTIAN RUTGERSSON

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Characterizing the Dynamics of Active Matter using Graph Neural Networks

CHRISTIAN RUTGERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Characterizing the Dynamics of Active Matter using Graph Neural Networks
CHRISTIAN RUTGERSSON

Supervisors:
Miguel Ruiz Garcia, Facultad Ciencias Físicas, Universidad Complutense de Madrid,
Carlo Manzo, Facultat de Ci'encies i Tecnologia, Universitat de Vic – Universitat Central de Catalunya,
Jesús Pineda, Department of Physics, University of Gothenburg,
Giovanni Volpe, Department of Physics, University of Gothenburg

Examiner: Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simulation of self-propelling, interacting particles forming clusters in a way that is commonly seen in active matter.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Characterizing the Dynamics of Active Matter using Graph Neural Networks
CHRISTIAN RUTGERSSON
Department of Physics
Chalmers University of Technology

Abstract

Biological systems often have self-organizing properties. On the microscopic scale, the self-organization may be driven by constituents that generate their own motility by expending energy. Systems made up of such constituents, that self-propel, are termed active matter. Via this definition, it is clear that it is important to understand the rules that govern the microscopic constituents of an active matter in order to understand the active matter itself. Even though active matter is a topic of interest today, finding good methods of qualitatively and quantitatively characterizing the constituents of active matter remains an issue. Coincidentally, different types of artificial neural networks (ANN) have, in recent years, been used increasingly in research settings with great success. One such network is called the graph neural network (GNN). As the name suggests, this network is specifically designed to work with graphs as input data. Graphs can act as a useful representation of a system of particles, including active matter systems. Therefore, this project aims to characterize the forces that underpin an active matter system consisting of interacting particles that also have an active component, using a special type of GNN called message passing network (MPN). This was done by creating a rudimentary simulation of this type of active matter, and training an MPN on it using standard machine learning algorithms. In the end, the simulations were found to give rise to characteristic active matter phenomena, and the MPN was able to correctly predict the force dynamics of a particle in the given active matter system.

Keywords: active matter, particle system simulation, machine learning, neural networks, graph neural networks, message passing network, message passing.

Acknowledgements

I would like to extend my deepest gratitude and appreciation to Miguel Ruiz Garcia, who was present with strong encouragement, wise advise, and valuable feedback all along the way of this project. I also want to thank Carlo Manzo, who contributed with valuable insight that were very helpful when obstacles arose. Additionally, I want to thank Jesús Pineda, whose invested support at a critical junction of the project was essential in its completion. Last but not least, many thanks to Giovanni Volpe, for inviting me into Soft Matter Lab, and encouraging me to attend group meetings, which offered me good insights into new, interesting fields of study.

Christian Rutgersson, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Artificial Neural Network
GNN	Graph Neural Network
MPN	Message Passing Network
CGNN	Convolutional Graph Neural Network
GCN	Graph Convolutional Network
MSD	Mean Squared Displacement
LJ	Lennard Jones
TSLJ	Truncated and Shifted Lennard Jones

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i,j	Indices for nodes in a graph
t	Index for time steps

Sets

V	Set of nodes.
E	Set of edges.
$\mathcal{N}(v_i)$	Set of nodes in the neighborhood of node v_i.

Constants

k_B	Boltzmann constant.

Parameters

f_a	Constant magnitude of active force in a system of interacting particles.
ϵ	Energy coefficient of Lennard-Jones potential.
σ	Length scale coefficient of Lennard-Jones potential. Also defined as the diameter of a particle.
d_{cut}	Distance at which to truncate Lennard-Jones potential.

d_{max}	Distance below which edges are drawn between a pair of nodes in the graph generation done post-simulation.
D_t	Translational diffusion coefficient.
D_r	Rotational diffusion coefficient.
T	Temperature.
ϕ_i	Bias of node i in a neural network.
w_{ij}	Weight parameter for connection between nodes i and j in a neural network.
ρ	Density parameter of particle system simulation.

Variables

$\mathbf{c}$	Position vector of a particle.
c^x	Position vector x-axis component.
c^y	Position vector y-axis component.
θ	Orientation of a particle (is 0 when particle is directed in the positive direction along the x-axis).
d	Distance between two particles.
$V_{\mathrm{P}}(d)$	Value of a potential with respect to distance.
$V_{LJ}(d)$	Lennard-Jones potential.
$V_{TSLJ}(d)$	Truncated and shifted Lennard-Jones potential.
$\boldsymbol{\xi}_{i,t}$	Two-dimensional random variable for simulating noise in the translation of particle i.
$\xi_{i,r}$	Random variable for simulating the Brownian fluctuation of the orientation of particle i.
ξ_i	Gaussian distributed random variable at time-step i.
$\hat{\xi}_i$	ξ_i divided by the square-root of the time-step length.
$\mathbf{n}_i$	Direction vector of particle i.
v_i	Node i.
e_{ij}	Edge connecting nodes i and j. Alternatively referring to a corresponding one-dimensional edge feature vector.
$\mathbf{v}_i$	Feature vector of node i.
$\mathbf{e}_{ij}$	Feature vector of the edge connecting nodes i and j.

Functions

$\mathbf{f}_{\mathrm{a}}(\theta)$	Active force vector.
$\mathbf{f}_{\mathrm{p}}(d)$	Passive force vector.
g	Activation function.
X	Dense neural network responsible for the first feature vector update in the MPN.
Ω	Dense neural network responsible for the last feature vector update in the MPN.
ζ	Edge function in ActiveNet.
ψ	Node function in ActiveNet.

Contents

List of Acronyms **ix**

Nomenclature **xi**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**
1.1 Background . . . 1
1.1.1 Active matter . . . 2
1.1.2 ActiveNet . . . 2
1.1.3 DeepTrack and MAGIK . . . 3
1.2 Purpose . . . 3

2 Theory **5**
2.1 Simulating active matter . . . 5
2.1.1 The dynamics of an overdamped active matter system . . . 5
2.1.2 Equations of motion . . . 7
2.1.3 Discretizing the equations of motion . . . 7
2.1.3.1 Brownian motion in simulations . . . 8
2.1.3.2 Discrete equation of motion . . . 8
2.1.3.3 Reduced LJ units . . . 9
2.2 Artificial neural networks . . . 9
2.3 Dense neural network . . . 9
2.4 Backpropagation and gradient descent . . . 10
2.5 Graph theory and definitions . . . 11
2.6 Graph neural networks . . . 11
2.7 Message passing neural networks . . . 12
2.8 ActiveNet's architecture . . . 13

3 Methods **15**
3.1 Dataset creation . . . 15
3.1.1 Simulation of active matter . . . 15
3.1.2 Graph generation . . . 17
3.1.3 Simulation specifications . . . 18
3.2 Implementing an MPN to learn the dynamics of a system of particles 19

3.2.1 MPN model parameters 19
3.2.2 Model training 19
3.2.3 Qualitative validation 20
3.2.4 Extracting the forces 20

4 Results **21**
4.1 Simulation results 21
4.2 MPN performance 21
4.2.1 Active force 22
4.2.2 Passive force 23

5 Discussion **29**
5.1 MIPS formation indicated in histograms 29
5.2 Choice of validation set 29
5.3 Evaluation of the choice of targets 29
5.3.1 Training time 30
5.4 Comparison to ActiveNet 30
5.5 Future work 30

6 Conclusion **33**

Bibliography **35**

List of Figures

2.1 Figures illustrating the TSLJ potential and its resulting force. 6

2.2 Schematic of an MLP, where the black lines represent the weighted connections between nodes. Note that only adjacent layers are connected. The red, unlabeled nodes represent the hidden layer. 10

3.1 An illustrative overview of the particle simulations. In 3.1a, the box with simulated particles with associated forces is shown, and in 3.1b, the properties of an individual particle are illustrated. 16

(a) Subfigure 1 list of figures text 16

(b) Subfigure 1 list of figures text 16

3.2 A graph representation of the snapshot in figure 3.1a, where $\mathbf{v}_i$ is the node feature vector, $\mathbf{t}_i$ is is the vector of labels for the node, and e_{ij} is the edge feature. The black lines between nodes represent graph edges. Edges are only generated below a certain distance threshold $d_{\max}$, and therefore, some nodes are not connected. Note that the bottom two nodes are connected because of their proximity via the periodic boundary condition. The distance encoded in the edge feature is the distance between the particles when the periodic boundary condition is taken into account. Note also that the label vector could either include both components of the active and passive force, or the two components of the sum of all forces acting on the particle (which would result in a two-dimensional vector). 18

4.1 Two snapshots of simulation 5 (as described in section 3.1). Figure 4.1a shows more clustering of particles. The axes are scaled such that one unit length corresponds to the diameter σ, of a particle. Here, a 22

4.2 A histogram of the pair-wise distances between particles over all of the sampled frames of the simulation shown in figure 4.1. All possible pair-wise distances are grouped into 50 bins. The total count for each bin is represented by a point. 23

4.3 The pair-wise distance histograms for simulations 1-4. All possible pair-wise distances are grouped into 50 bins. The total count for each bin is represented by a point. 24

4.4 The training loss and validation loss during the training of the MPN on datasets from four different simulations. 25

4.5 The MPN active force prediction results for simulations 1-4. Note that for simulation 1 and 2, the method of acquiring the active force was different from the method for 3 and 4, as described in section 3.2.4. 26
4.6 The MPN passive force prediction results for simulations 1-4. Note that for simulation 1 and 2, the method of acquiring the active force was different from the method for 3 and 4, as described in section 3.2.4. Also, the closest distance that particles where placed from each other in simulations 1 and 3 was 1.1σ. 27
4.7 A zoom-in on the bottom left part of figure 4.6b, highlighting a range of distances where the force predictions were incorrect. 28

List of Tables

3.1 Simulation specifications for simulations used in this thesis. 19
3.2 The parameters that were used for the MPN. 19

4.1 Mean absolute error (MAE) with respect to the simulation used. Note that the passive force MAE is only calculated for the distances 1.1 - 3.0, and refers to the discrepancy between the model's prediction and the true force value w.r.t. orientation or distance when using the method described in section 3.2.4 to extract the forces. 21

1
Introduction

Systems characterized by many, densely packed interacting particles are present everywhere in nature. One salient example is that of interacting, living cells. When analyzing microscope images of such an environment, it is often useful to process the images in some way, in order to gain information about hidden properties. This can be done with algorithms of varying complexity, one example being the thresholding method, where a gray-scale image is transformed to a binary image based on a threshold, resulting in a rudimentary classification of the objects in the image (cells versus not cells in this example). In recent years, machine learning approaches for extracting useful information from experimental data have become more prevalent. For example, convolutional neural networks (CNN) are famously adapted to analyzing images, and extracting useful statistical information from them [1].

Depending on the type of input used, and the problem to be solved, the optimal neural network type varies. The input of interest in this thesis is the graph, consisting of a number of nodes and edges connecting them (a more detailed description can be found in section 2.5). Such a datatype is well suited to modelling systems of interacting cells and particles. In this case, the CNN is not easily applicable, because the input size of a graph will vary depending on the number of nodes (cells, particles) present in a given snapshot, and CNNs generally have a rigid input data size based on the resolution of the input image [1]. Instead, one may use a graph neural network (GNN), which is a neural network purposefully constructed to work on graph input data. In this thesis, the distinctive qualities of GNNs have been used to characterize the forces governing a system of interacting particles, in which the particles themselves are affected by self-propelling forces.

1.1 Background

The approach of using GNNs to characterize a the forces in a system of particles has already seen a some degree of interest and progress [2]. As the name suggests, graph neural networks exploit graph representations of input data. In many cases, this representation limits the output space of the network, and thereby simplifies the task for the neural network.

Although it would be highly useful to have a machine learning method which can deduce the physical laws at play in any type of particle system, this is a difficult task to tackle immediately. Therefore, the scope of this project is limited to one branch of systems of particles, namely active matter.

1.1.1 Active matter

Active matter is characterized by the ability of its constituents to generate mechanical force through energy dissipation. On a microscopic scale, a clear example of active matter are some bacteria (such as E-coli), which are able to propel themselves through a medium, albeit with some amount of stochasticity in the direction of travel [3] [4]. The resulting diffusion of these bacteria becomes very different compared to particles only undergoing thermal diffusion in the same medium. This is an example of how active matter tends to display emergent macroscopic properties thanks to the active motion of the microscopic constituents. Other examples of active matter include flocks of birds - where each bird adapting its direction to the rest of the population yields the characteristic movement of the flock as a whole, and the cytoskeleton of the human cell - an important tissue which stretches throughout the cell, and restructures itself to change the functionality of the cells, which in turn affects larger scale processes in the body [3]. One might also view humans themselves as the constituents of an active matter (our social structure) which reorganizes and reacts in complex ways.

By discovering the rules which govern the constituents of active matter, the same principles can be applied on synthetic or numerically simulated particles to recreate or model the type emergent behavior found in active matter in nature.

1.1.2 ActiveNet

In a recent work, [5], it was emphasized that a problem in the field of active matter study is how to effectively determine the correct expressions of forces which act on the interacting particles in various active matter experiments. As the authors of the work described it, the current method is to make an educated guess on the correct model for the system under consideration, make experiments, compare, and iterate the process. This process may be impractical in many situations, and the authors therefore opted to explore the potential of machine learning as an alternative. They constructed a specific type of graph neural network which they named ActiveNet. A full description of it's architecture and how it functions can be found in section 2.8. When trained, ActiveNet could be used to extract both the potential-based two-body force between two particles in an interacting system of particles and the active, self-propelling force acting on each particle. However, for this graph neural network to be applicable, certain assumptions were made for the particle system under consideration. At the time of this project's start, ActiveNet was designed to only work on a system where the velocity of each particle was proportional to the total force acting on it. This was a way to model an overdamped regime where inertia can be ignored. Another way to word it is that the particle configurations of preceding time-steps have no effect on the forces which act on the particles in the current time-step. Thus, ActiveNet, in this iteration, is designed and optimized to evaluate the instantaneous forces acting on particles based on a snapshot in time. Therefore, the forces in a particle system with pronounced inertial properties would not be predicted well by ActiveNet.

1.1.3 DeepTrack and MAGIK

Coincidentally, an advanced graph neural network model called Motion Analysis through GNN Inductive Knowledge (MAGIK) has recently been developed [6]. MAGIK is part of a greater machine learning package called DeepTrack [7]. The model is based on a more fundamental and commonly used graph neural network model called message passing network (MPN) which will be described in detail in section 2.7. MAGIK's output can be tailored to the needs of the user. For example, MAGIK can be used to estimate the trajectories of particles in a sequence of microscopy snapshots, but it can also directly estimate the spatial distribution of diffusivity in a medium without explicitly establishing any trajectory relationship between particles in adjacent time-frames beforehand. One important detail in MAGIK's architecture is that it is designed to consider changes in graphs over time. With that in mind, one might ask whether MAGIK is able to perform the same task as ActiveNet, even while dropping the constraint that no inertia is present in the particle system.

1.2 Purpose

The ultimate goal of this project is to use a GNN to model an interacting system of particles using simply a sequence of images containing their positions and implied velocities. This would prove that such an application of GNNs can act as a more convenient method for characterizing the forces on the particle level than the ones previously used. As mentioned, ActiveNet has limitations on the breadth of problems it can be applied to, since the implementation rests on, among others, the assumption that history from previous time-steps does not affect the state of particles in the current time-step. Therefore, it is of interest to determine if another type of GNN such as MAGIK could be capable of performing the same tasks as ActiveNet while placing fewer assumptions on the system of particles.
In practice, the main goal of this thesis is to determine how well one can use an MPN to complete the same task as ActiveNet. Simultaneously, the project becomes a staging point for applying MAGIK on more advanced problems, since MAGIK is a more advanced version of an MPN.
In order to train the neural networks, training data is needed. Also, with full insight into the details of the training data itself, one gains more agency to modify it to fit the project's needs. Therefore another goal is to create a Python code which roughly emulates the simulated training data used in [5].

2
Theory

Here follows a detailed description of simulations used to emulate a type of active matter, followed by surface level theory in artificial neural networks and graph neural networks, and a description of neural network architectures that are referred to in this thesis.

2.1 Simulating active matter

The simulations of active matter used in this thesis are based heavily on those made for training ActiveNet in [5]. Here follows a review of relevant theory for the simulation of an interacting system of particles as an active matter model.

2.1.1 The dynamics of an overdamped active matter system

In an interacting system of particles, each particle may have unique properties such as for example a coordinate and orientation (other realistic examples of particle properties would be area and mass). Take a system in which N particles are characterized by the coordinate and orientation properties. The coordinates for each particle $i = 1, 2 \ldots, N$ are given by $\mathbf{c}_i = (c_i^x, c_i^y)$ and the orientation by θ_i. Each particle is influenced by an *active force* and a *passive force*. The active force is computed as

$$\mathbf{f}_{\mathrm{a}}(\theta) = (\cos\theta, \sin\theta) \cdot f_a, \tag{2.1}$$

where f_a is a chosen parameter that is constant. The passive force is given by

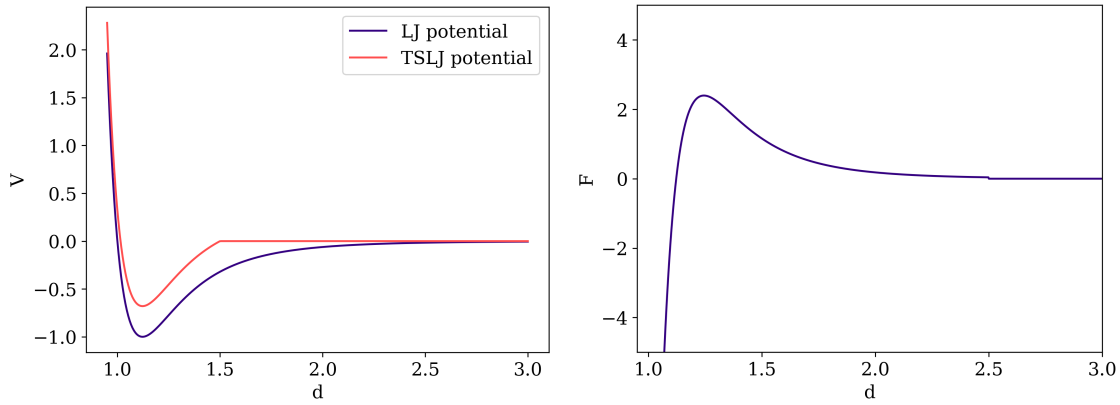
$$\mathbf{f}_{\mathrm{p}}(d) = -\nabla V_{\mathrm{P}}(d), \tag{2.2}$$

where d is the distance between two particles and $V_{\mathrm{P}}(d)$ is a potential w.r.t. distance. The active force is a self-propelling force, pushing the particle in the direction of its orientation, that acts independently of the local environment, while the passive force is conservative, two-body force which acts on two particles with equal magnitude according to a potential.
The type of potential which is used in this project is a truncated and shifted Lennard-Jones potential, $V_{\mathrm{TSLJ}}(d)$, given by

$$V_{\mathrm{LJ}}(d) = 4\epsilon \left[\left(\frac{\sigma}{d}\right)^{12} - \left(\frac{\sigma}{d}\right)^{6} \right], \tag{2.3}$$

$$V_{\mathrm{TSLJ}} = \begin{cases} V_{\mathrm{LJ}}(d) - V_{\mathrm{LJ}}(d_{\mathrm{cut}}) & \text{if } d < d_{\mathrm{cut}} \\ 0 & \text{if } d > d_{\mathrm{cut}} \end{cases} \tag{2.4}$$

(as described in [5]), where ϵ is named the energy parameter and σ the length scale parameter. The non-shifted and truncated Lennard-Jones potential is given by $V_{\mathrm{LJ}}(d)$. Figure 2.1 illustrates these two potentials side by side. The Lennard-Jones potential models the repelling and attracting van der Waal interactions of neutral atoms and molecules [8]. Thus the modeled active matter system in question could be viewed as one concerning neutral particles of a small size that actively propel due to some externally applied field.



(a) LJ potential versus a truncated and shifted LJ potential, where $d_{cut} = 1.5$.

(b) Illustration of the force that is gained from a TSLJ potential with $d_{cut} = 2.5$.

Figure 2.1: Figures illustrating the TSLJ potential and its resulting force.

The self-propulsion of the particles in this model results in drastically different macroscopic properties for the system, compared to when no active motion is applied. If only the passive force was active, the particles would attract each other and aggregate into stationary clusters and eventually find an equilibrium configuration. In contrast, when the active force is applied to the system, a characteristic so-called motility induced phase separation (MIPS) occurs, where one observes areas of varying particle density (phases). One way to explain the formation of MIPS is by observing a 2D system in which particles move at a certain speed, u_1, in some areas, and move at another speed, $u_2 < u_1$, in the rest of the 2D space. Even while assuming that the two types of areas occupy the same fraction of the total, the particle will stay longer in the area where it moves slower. Thus, over time, one will observe a difference in the densities, ρ_1, ρ_2, of the two area classes. Likewise, for the case with the interacting system of particles, the areas where collisions occur between particles can be likened to the low-speed areas. In areas of collisions or high proximity, particles will stay longer, resulting in these areas with higher density. It is worth noting that the heterogenous distibution of particle speeds, and subsequently the appearance of MIPS, is an intrinsic property found in non-equilibrium systems such as active matter. Systems in thermal equilibrium generally do not display spatial dependence of particle speeds, since that would violate the thermal equilibrium constraint (due to the speed being proportional to the homogeneously distributed temperature) [9].
For an overdamped system, viscous forces ensure that the effect of inertia is nullified and negligible. In this regime, the total force acting on a particle is sometimes

modeled as proportional to its speed, $\dot{\mathbf{c}}_i \propto \mathbf{f}_i$ [5]. The argument for this is as follows: take a particle with velocity $\mathbf{u}$ which is affected by some external force $\mathbf{F}$ in a viscous medium. Thus there is a drag force proportional to the speed of the particle, $\mathbf{F}_\mathrm{D} = -b\mathbf{u}$. Newton's second law gives us the particle's equation of motion as

$$m\dot{\mathbf{u}} = -b\mathbf{u} + \mathbf{F}. \tag{2.5}$$

It is clear that the particle will accelerate until the drag force and the constant force $\mathbf{F}$ are equal. When this happens, the velocity will be $\mathbf{u}^* = \mathbf{F}/b$. The time τ that it takes to reach this equilibrium can be calculated via integration. It is proportional to m/b. The system being in the overdamped regime implies that b takes on a very large value (the drag force is large), and therefore τ will tend to zero. In effect, this means that the time of acceleration for a particle is negligible, since the particle reaches the equilibrium velocity almost instantly, and the left-hand-side of equation 2.5 can be approximated to zero. The resulting equation is one where the velocity of a particle is proportional to the total external force.
In order to simulate the thermal fluctuations and collisions with smaller particles, Brownian motion is also included in the simulation, meaning that there are random fluctuations in the velocity of each particle. The magnitude of these fluctuations are set by diffusion parameters.

2.1.2 Equations of motion

In the overdamped regime, with active and passive forces with Brownian motion as described in the previous section, the equation of motion is given by

$$\dot{\mathbf{c}}_i = \frac{D_t}{k_B T}\left(\sum_{v_j \in \mathcal{N}(v_i)} \mathbf{f}_\mathrm{p}(d_{ij}) + f_a \mathbf{n}_i\right) + \sqrt{2D_t}\,\boldsymbol{\xi}_{i,t}, \tag{2.6}$$

$$\dot{\theta}_i = \sqrt{2D_r}\,\xi_{i,r}, \tag{2.7}$$

where $\mathbf{n}_i$ is the direction vector of particle i, k_B is the Boltzmann constant, T is the temperature, D_t and D_r – respectively – are the translational and rotational diffusion constants, while $\boldsymbol{\xi}_{i,t} \in \mathbb{R}^2$ and $\xi_{i,r} \in \mathbb{R}$ are uniformly distributed random variables of element-wise mean and variance of zero and one with no correlation between particles [5].

2.1.3 Discretizing the equations of motion

One way of creating simulations based on equations 2.6 and 2.7 is to discretize them (and subsequently use Euler's method). This means that the particle states (in this case, the coordinates and orientations) are evaluated at discrete time-steps. A simple way of achieving this is to add an increment $\Delta\mathbf{c}_i$ to the coordinates at each time-step, where the increment is determined as $\Delta\mathbf{c}_i = \dot{\mathbf{c}}_i \cdot \mathrm{d}t$ (analogously for the orientations). However, when involving Brownian motion in the simulation, some additional considerations are required to construct a valid simulation.

2.1.3.1 Brownian motion in simulations

For a simulation that emulates a physical system, it is desirable that the system behaves similarly even when the time-step size is varied, as long as it is short enough to not skip past relevant events. In [10], a simple model of a particle undergoing Brownian motion in one dimension is presented as

$$x_{i+1} = x_i + \xi_i \, \mathrm{d}t, \tag{2.8}$$

where the position of the particle at time-step i is given by x_i, the time-step length is $\mathrm{d}t$, and ξ_i is a Gaussian random variable of mean zero and variance σ_ξ^2. It is clear that the mean of a system with many such particles will be the starting point of the particle no matter which time-step size is chosen, since there is no bias in the direction of travel. However, when computing the mean squared displacement (MSD) (under the assumption that $x_0 = 0$ and $\langle x_i \rangle = 0$):

$$\left\langle x_n^2 \right\rangle = \cdots = \left\langle x_0^2 \right\rangle + \sum_{i=1}^{n} \left(\langle 2 \, \xi_{i-1} \, \mathrm{d}t \, x_{i-1} \rangle + \left\langle \xi_{i-1}^2 \, \mathrm{d}t^2 \right\rangle \right) = n \, \sigma_\xi^2 \, \mathrm{d}t^2 = \sigma_\xi^2 \, \mathrm{d}t \, t, \tag{2.9}$$

one finds that the MSD will be different for the same point in time t if $\mathrm{d}t$ is varied ($t = n \, \mathrm{d}t$). Therefore, this model isn't physically valid (since the result of an emulation of reality shouldn't be based on arbitrary parameters), unless a modification is made [10]. The solution should make it so that the choice of $\mathrm{d}t$ doesn't affect the simulation result. To this end, one may scale the random variable variance based on the square root of the time-step length, $\hat{\xi}_i = \frac{\xi_i}{\sqrt{\mathrm{d}t}}$. Replacing ξ_i with $\hat{\xi}_i$ in equation 2.9,

$$\left\langle x_n^2 \right\rangle = n \, \sigma_\xi^2 \, \mathrm{d}t = \sigma_\xi^2 \, t. \tag{2.10}$$

Thus, the MSD is no longer dependent on the time-step size. In summary, the time-step dependent MSD is replaced by a time-step dependent variance for the random variable $\hat{\xi}_i$, which is not something non-physical, since it is merely one arbitrary parameter affecting the other, in such a way that the simulation result remains the same. When replacing the random variable in equation 2.8 with $\hat{\xi}_i$, the result becomes

$$x_{i+1} = x_i + \hat{\xi}_i \, \mathrm{d}t = x_i + \xi_i \sqrt{\mathrm{d}t}. \tag{2.11}$$

Put simply, the time-increment should be replaced by its square root for the Brownian motion term.

2.1.3.2 Discrete equation of motion

In the previous section, it is shown that the MSD in simulations of Brownian motion will vary depending on $\mathrm{d}t$ unless the random variable is divided with the square-root of the time-step. When discretizing equations 2.6 and 2.7, the same principle should be used to achieve a time-step independent MSD for the diffusion terms. With this in mind, the increments for the coordinates and orientations becomes

$$\Delta \mathbf{c}_i = \mathrm{d}t \cdot \frac{D_t}{k_B T} \left(\sum_{v_j \in \mathcal{N}(v_i)} \mathbf{f}_{\mathrm{p}}(d_{ij}) + f_a \mathbf{n}_i \right) + \sqrt{2 D_t \mathrm{d}t}\, \boldsymbol{\xi}_{i,t}, \tag{2.12}$$

$$\Delta \theta_i = \sqrt{2 D_r \mathrm{d}t}\, \xi_{i,r}. \tag{2.13}$$

2.1.3.3 Reduced LJ units

Using dimensionless units can help simplify simulations of particle dynamics. When using reduced Lennard-Jones units, one sets σ, ϵ, and the particle mass $m = 1$. Meanwhile, $d^* = d\sigma$, $F^* = \sigma F/\epsilon$, $V^* = V/\epsilon$, and $k_B = 1$, $t^* = t\sqrt{\epsilon/m\sigma^2}$ [5], [11]. In the rest of this text, the asterisks will be dropped for the sake of simplicity.

2.2 Artificial neural networks

Artificial neural networks (ANN) are one of the most common models which are used within machine learning. Generally, an ANN is composed of a number of neurons/nodes. Each node connects to a subset of the other nodes. Data is propagated through the network via these connections. Each node value, x_i, is computed in the following way:

$$x_i = g(\sum_{j}^{N} w_{ij} x_j - \phi_i), \tag{2.14}$$

where N is the number of nodes connected to the current node i, w_{ij} is a weight for the connection strength between the current node and node j, ϕ is a single, trainable value often called the bias. The encapsulating function g is called the *activation function*. Depending on where the node is located, and the application of the neural network, different activation functions are used. Common activation functions are ReLU, linear, signum, and sigmoid. Often, a set of nodes are identified as the input nodes and output nodes, where the input nodes are given external values. From the input nodes, data is propagated through the network by sequentially updating the value of each node using equation 2.14. If the weights and biases stay constant, and the input value doesn't change, the output remains the same. Since the goal of neural network-based machine learning is to optimize the response of the output neurons in some way, a method is required to change the weights and biases based on input data. [12]

2.3 Dense neural network

A type of neural network which is often seen as the standard in supervised machine learning is the dense neural network. It consists of an arbitrary number of nodes arranged in sequential layers, where each node is connected to every other node in the adjacent layers, as shown in figure 2.2. The layers that are neither output nor input layers are called hidden layers.

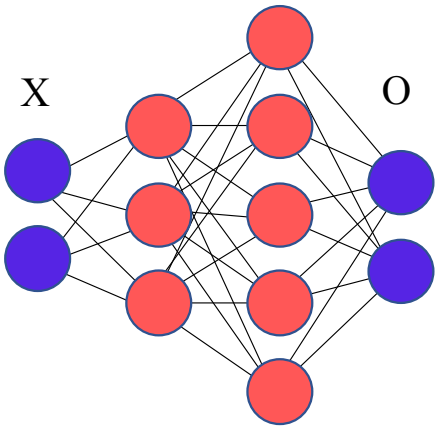


Figure 2.2: Schematic of an MLP, where the black lines represent the weighted connections between nodes. Note that only adjacent layers are connected. The red, unlabeled nodes represent the hidden layer.

2.4 Backpropagation and gradient descent

Assume that, for a certain input $\mathbf{X} = (x_0, x_1, ..., x_N)$, one receives an output $\mathbf{O} = (o_0, o_1, ..., o_M)$. In supervised machine learning, the correct output based on the given input is called the target, represented here by $\hat{\mathbf{O}} = (\hat{o}_0, \hat{o}_1, ..., \hat{o}_M)$. One can calculate the error of the network's output via a suitable *loss function* f_{loss}. A common loss function is the *mean square error*:

$$H = f_{\text{loss}}(\mathbf{O}, \hat{\mathbf{O}}) = \left\langle \sum_i (o_i - \hat{o}_i)^2 \right\rangle_\mu , \tag{2.15}$$

where one averages the squared error of each output node O_i compared to the target output $\hat{o}_i$ (also called label), over the different possible inputs in a set of data, denoted by μ. Thus if the error in an output node is higher, the loss function outputs a higher error value, H, for that node. More generally, to train a neural network via a supervised learning scheme, one defines a loss function which is defined on the space of all possible weights and biases, and which has minima for the weights and biases which result in $\mathbf{o} = \hat{\mathbf{o}}$ for all available inputs [12]. The objective of the network is to minimize the error of the output nodes, which correlates to finding the minima of the loss function with respect to the weights and biases. This is done by using *backpropagation* and *gradient descent*.
With gradient descent one iteratively modifies the weights based on the loss function's gradients with respect to the weights. Take an arbitrary weight w_{ij}^l within a dense neural network with L layers. Here, w_{ij}^l refers to the weight between the node in layer l indexed by i, and the node indexed by j in layer $l + 1$. The value of this weight is one coordinate in the space of all weights and biases. As mentioned, the loss function varies within this room, and one strives to find the minima. A numerical way of finding a minimum is to iteratively step in the opposite direction of the gradient of the loss function. The gradient is a vector made up of the partial derivatives of the loss function with respect to the weights and biases:

$$\nabla f_{\text{loss}} = \left(... \frac{\partial f_{\text{loss}}}{\partial w_{ij}^l}, ..., \frac{\partial f_{\text{loss}}}{\partial \phi_i^l}, ... \right) \forall i, j, l. \tag{2.16}$$

To compute this gradient, the method of backpropagation is used. Put shortly, computations of derivatives are done sequentially from ”right" (the later layers) to "left" (the earlier layers), since the derivatives are dependent on an error factor that propagates from right to left [12]. Going into further detail isn't essential for understanding the content of the thesis, but for the interested reader, [12] gives a thorough introduction to machine learning concepts.

2.5 Graph theory and definitions

In this thesis, a graph is defined as the set $G = (V, E)$, where V represents the set of nodes, and E the set of edges. The ith element in the node set is represented by $v_i \in V$, while the edge between node i and j is represented by $e_{ij} \in E$. Depending on the application, the nodes and edges can each be associated with a *feature vector*. For nodes, this feature vector could for example include the mass of a particle, its area, and/or its coordinates in euclidean space, while the edge features could represent the pair-wise distance between particles, or the type of chemical bonding between two atoms. In this thesis, $\mathbf{v}_i$ will refer to the i:th node's feature vector, while $\mathbf{e}_{ij}$ will refer to the feature vector of the edge between the i:th and j:th node. The neighborhood of node i, $\mathcal{N}(i)$ is defined as the set of nodes that share an edge with this node.
The adjacency matrix, $\mathbf{A}$ is a commonly used entity that describes the connectivity in a graph. If there are n nodes in a graph, it is an $n \times n$ matrix where the elements, a_{ij}, are equal to the number of edges from node i to j. In this thesis, no index pair ij will have more edges than one, so $e_{ij} \in E \iff a_{ij} = 1$.
A graph can either be *directed* or *undirected*. If it is directed, that means that e_{ij} connects v_i to v_j, but there is no connection in the opposite direction. For an undirected graph, $e_{ij} \in E \iff e_{ji} \in E$. By definition, an undirected graph will imply a symmetric adjacency matrix. [13]

2.6 Graph neural networks

Early outlines of graph neural networks (GNN) can be traced back to 2005. [1] Since then, many different types GNNs have emerged, continuously building on the ideas of predecessors. One large category of GNNs is called convolutional graph neural networks (ConvGNNs). Here, the idea is to aggregate information in each node from their respective neighborhood nodes a set number of times, and in this way gain new, updated node features. [1] A simple example of ConvGNNs are Graph Convolutional Networks (GCNs). For a graph $G = (V, E)$, each node feature vector $\mathbf{v}_i^t \in V$ (time step t) is updated as

$$\mathbf{v}_i^{t+1} = g(\sum_{v_j \in \mathcal{N}(v_i)} a_{ij} W \mathbf{v}_j), \tag{2.17}$$

where $\mathbf{v}_i^{t+1}$ is the updated node feature vector, g is an activation function, W is a trainable weight matrix which is applied to each node in the neighborhood, and $a_{ij} = \frac{1}{\sqrt{d(v_i)d(v_j)}}$, where $d(v_i)$ is the degree of v_i (number of nodes in $\mathcal{N}(v_i)$). A limitation with GCNs are that they do not make use of edge features, which could contain useful information, such as euclidean distance between particles. [1] [14]

2.7 Message passing neural networks

In contrast to GCNs, a GNN that does make use of edge features is the Message Passing Network (MPN). It shares many similarities with GCNs. Keeping the same notation as above, each node feature vector is first passed through a dense neural network, X, with trainable weights (training is done via gradient descent as described in section 2.4). After all of the node feature vectors have been updated to a new representation, $\mathbf{v}_i'$, messages $\mathbf{m}_{ij}$ are aggregated from each node's neighborhood in the following way. For each connected node pair, a message function, Ψ, is applied on the concatenation of their node features and the corresponding edge's features,

$$\mathbf{m}_{ij} = \Psi(\mathbf{v}_i', \mathbf{v}_j', \mathbf{e}_{ij}), \tag{2.18}$$

assuming that $v_j \in \mathcal{N}(v_i)$. The message function Ψ often has trainable parameters, but can also be set to just 1, leaving the concatenation of the node features and edge features unchanged. In each node neighborhood, the messages are aggregated (as a sum in this project) and passed through another trainable function Ω together with the previously updated node feature vector $\mathbf{v}_i'$, resulting in the second and final node update

$$\mathbf{v}_i'' = \Omega\left(\mathbf{v}_i', \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{m}_{ij}\right). \tag{2.19}$$

Summarizing the whole algorithm concisely,

$$\mathbf{v}_i'' = \Omega\left(\mathbf{v}_i', \sum_{v_j \in \mathcal{N}(v_i)} M(\mathbf{v}_i', \mathbf{v}_j', \mathbf{e}_{ij})\right), \tag{2.20}$$

where

$$\mathbf{v}_i' = X(\mathbf{v}_i). \tag{2.21}$$

Once $\mathbf{v}_i''$ has been computed for all nodes, the same operation can be applied again, resulting in information being passed further away from the nodes' local neighborhoods. [1]

After all of the node features have been passed through the MPN, and updated, the resulting feature vectors can be compared to target feature vectors. As described in section 2.4 this comparison results in an error, which is used (via gradient descent) to incrementally change the parameters of X and Ω to better fit the targets in the next iteration.

2.8 ActiveNet's architecture

Central to the design of ActiveNet is the assumption that the only types forces present in the experiment are two-body forces or one-body forces, which will henceforth be referred to as the passive and active forces respectively. Concisely, ActiveNet can be described as the sequential application of two functions, the *node function* ψ, and the *edge function* χ:

$$\mathbf{v}_i^{\mathrm{v}} = \psi\left(\mathbf{v}_i, \sum_{v_j \in \mathcal{N}(v_i)} \chi(\mathbf{v}_i, \mathbf{v}_j)\right), \tag{2.22}$$

where, in this case, the node feature vectors only include the particle coordinates and the particle's orientation: $\mathbf{v}_i = (c_i^x, c_i^y, \theta_i)$, and the neighborhood $\mathcal{N}(v_i)$ is defined as all nodes below a threshold distance Γ from v_i. Every node is transformed via this operation, and the resulting node feature vector is the predicted velocity of the particle ($\mathbf{v}_i^{\mathrm{v}} \in \mathbb{R}^2$). In the model's implementation, the functions ψ, and χ are dense neural networks, and thus are trainable in the same way that an MLP would be. [5] In the paper ([5]), ActiveNet was trained on a simulated interacting system of particles, which was allowed to run for ten million time-steps with dt = 1e−5, and a total number of 2500 particles per frame. Out of the whole simulation, 380 frames were sampled. If this sampling was not done, one would have a large amount of data, but with many frames being almost duplicates due to the small time-step needed for a valid simulation.
When the training is done, the usefulness of the ActiveNet architecture is demonstrated in how easily the relevant forces can be extracted. Because the interacting two-body force tends to zero as particles are at a very far distance away from each other, the network necessarily learns this for the loss to decrease. According to equation 2.22, the velocity of a particle v_i which is so far away from all other particles such that $v_i \notin \mathcal{N}(v_j) \forall j$ is given by

$$\mathbf{v}_i^{\mathrm{a}} = \psi\left(\mathbf{v}_i, \mathbf{0}\right), \tag{2.23}$$

where $\mathbf{0}$ is the zero vector. The sole source of this velocity can only be the active force. Therefore, $\psi\left(\mathbf{v}_i, \mathbf{0}\right)$ extracts the part of the velocity that is proportional to the active force, named $\mathbf{v}_i^{\mathrm{a}}$ here. If the network is trained correctly, the total velocity of two nodes v_i & v_j with any arbitrary distance d_{ij} between them is computed by equation 2.22. Therefore, since the total velocity is the sum of the velocity vectors resulting from the passive and active forces, the velocity resulting from the passive force, $\mathbf{v}_i^{\mathrm{p}}$, can be extracted as

$$\mathbf{v}_i^{\mathrm{p}} = \psi\left(\mathbf{v}_i, \chi(\mathbf{v}_i, \mathbf{v}_j)\right) - \psi\left(\mathbf{v}_i, \mathbf{0}\right). \tag{2.24}$$

3
Methods

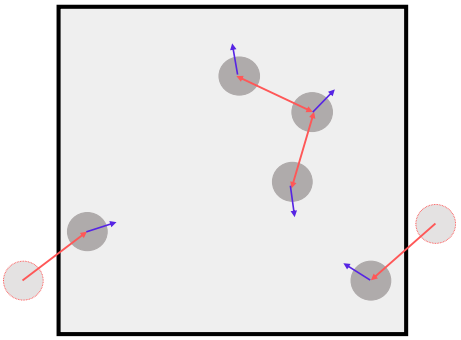
In this section, the approach used for generating training data, training the MPN, and the post-processing of the MPN's output is described, along with specifications of relevant parameters.

3.1 Dataset creation

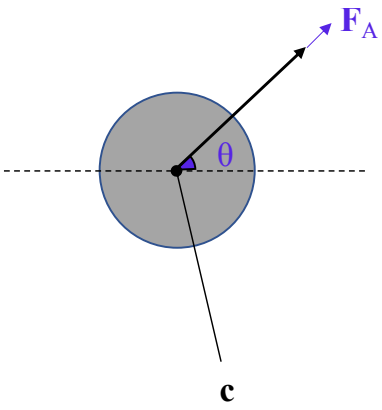
As stated, the objective was to train a graph neural network to learn the characteristics of the forces that were present in a system of interacting particles. Therefore, datasets containing graph representations of such systems were created. Here, nodes represented particles, while edges connected particles that had a pair-wise distance below a specified threshold. The nodes carried three node-features each: a horizontal position coordinate, a vertical position coordinate, and an orientation. The edges carried one feature: the pair-wise euclidean distance between the connected nodes. The process for generating this data can be categorized into two main parts: the simulation part and the graph generation part. For the simulation part, a code was written in which particle positions and orientations were generated and updated according to discretized equations of motion. Snapshots from the simulation were then processed into graph representations.

3.1.1 Simulation of active matter

This code was written in Python, and it utilized a high degree of matrix operations, leading to a faster computation time compared to the alternative of iterating through lists with for-loops. Additionally, the Python package *CuPy* [15] was used to speed up the computations by allowing matrix operations to be done on a GPU.
For the particle system simulations, reduced LJ units were used, which are detailed in section 2.1.3.3. All particles were inside a square box with a specified side length L and periodic boundary conditions such that a particle might exit the box on the right side and reappear on the left side, as seen in figure 3.1. The particles in question had three properties each: an x-coordinate, a y-coordinate, and an orientation.
The operation of the simulation was as follows: The initial values of the particle properties were set to uniformly distributed random values. If two particles were found to be within a distance of 1.1 (in reduced LJ units) from each other, one of the particles was replaced. By iterating this process, it was made sure that no particles were closer to each other than would be physically likely, given the steep increasing potential energy at close distances. Thereafter the coordinates and orientations



(a) Illustration of a simulation where particles are used. The red arrows represent the two-body passive forces, while the blue arrows represent the self-propelling active force in the direction of the particle's orientation. The transparent particles with red borders outside of the box show how the particle interaction occurs across the box boundaries via the periodic boundary condition.



(b) A zoomed in version of one of the particles in (a) for further clarity, where $\mathbf{c}$ and θ are the coordinates and orientation of the particle, and F_A is the active force. As can be seen, the orientation is measured from the horizontal axis.

Figure 3.1: An illustrative overview of the particle simulations. In 3.1a, the box with simulated particles with associated forces is shown, and in 3.1b, the properties of an individual particle are illustrated.

were updated according to equations 2.12 and 2.13 frame by frame. For this, the distance had to be computed for each particle pair. In the distance computation, the smallest possible distance between two particles – with the periodic boundary condition taken into account – was always chosen. Thus, two particles might interact with each other on two different sides of the box. Interaction across the boundaries was essential, since if it had not been present, particles crossing from one side to the other could end up very close to another particle, with an extremely high and physically unlikely repelling force, leading to an unstable simulation.
A parameter, $\rho = \frac{A_p}{L^2}$, was set to relate the total area of particles, $A_p = N \cdot (\sigma/2)^2\pi$, to the area of the simulation box (given N particles in total). In this way, by setting ρ to a constant, the number of particles could be varied while retaining the same particle density in the simulation.
The time-step of the simulation was small in order to ensure a physically valid simulation, but that meant that the properties of the particles varied slowly. Thus, adjacent frames included almost the same configuration of particles. Training on larger amounts of data tends to be slower, and therefore, the data size becomes a constraint, and reducing the amount of redundant data in the training data is important. To ensure that the final data from the simulations included a wide variety of particle configurations, the simulations were sampled at select time-steps. Each sample included the current properties (coordinates and orientations) of all particles, and their forces. By aggregating the samples, a trajectory with a wider time-step than the simulation itself was built up. After all the samples had been collected, the result was plotted to ensure that the particles had moved enough for each frame.
For all simulations the following parameters were set to constant values: $\{D_t = k_\mathrm{B}T = 0.01,\ D_r = 1,\ \mathrm{d}t = 1e-5,\ \sigma = 1,\ d_\mathrm{cut} = 2.5\}$.

3.1.2 Graph generation

As a final step in preparing the data to be used in the GNN training, the simulation output data was converted into a graph format, as shown in figure 3.2. Firstly, the particles were represented as nodes with the three node features x-coordinate, y-coordinate, and orientation. Secondly, while taking periodic boundary conditions into account, edges were drawn between nodes of the same time-step whose pair-wise distance was below a specified threshold $d_\mathrm{max} = 3\sigma$. In this way, the connectivity of the graph serves as an extra addition of information for the graph neural network during training. The pair-wise distance was also included as an edge feature. Finally, for each node, a label describing the forces acting on it was attached. Specifically, the label was a vector including some number of components of the instantaneous velocity of the particle when disregarding Brownian motion. The vector was either two-dimensional or four-dimensional. When two-dimensional, the label vector included the x and y components of the velocity vector with the Brownian motion term removed. When four-dimensional, the label vector would include the x and y components of the velocity resulting from the active force and the x and y components of the velocity resulting from the passive force. The dimensionality of this label was varied between two and four in order to test how well the MPN would

work with different degrees of information. In summary, a graph was generated for each simulation sample. Each such graph was independent, without any connections to any of the other graphs.

With the parameters used in this project, the part of the velocity resulting from the active force is equal to the active force, and likewise for the passive force. Therefore, these velocity components will henceforth simply be referred to as the passive and active forces.

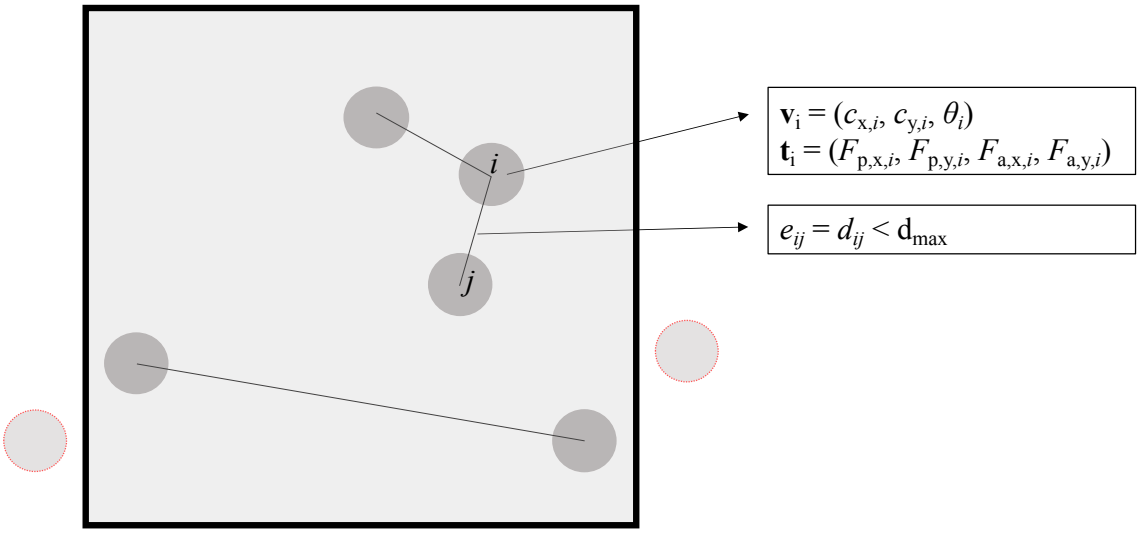


Figure 3.2: A graph representation of the snapshot in figure 3.1a, where $\mathbf{v}_i$ is the node feature vector, $\mathbf{t}_i$ is is the vector of labels for the node, and e_{ij} is the edge feature. The black lines between nodes represent graph edges. Edges are only generated below a certain distance threshold d_{max}, and therefore, some nodes are not connected. Note that the bottom two nodes are connected because of their proximity via the periodic boundary condition. The distance encoded in the edge feature is the distance between the particles when the periodic boundary condition is taken into account. Note also that the label vector could either include both components of the active and passive force, or the two components of the sum of all forces acting on the particle (which would result in a two-dimensional vector).

3.1.3 Simulation specifications

Table 3.1 shows the specifications of all of the simulations that are referred to in this thesis. In two of the simulations, the particles configuration was reset with the same method as setting the initial values (resulting in a set of particles with random positions and orientations). This was a way to gain data with well mixed particle configurations, in order to investigate whether the slow-down of the dynamics due to MIPS formation of the fully simulated data would lead to less useful training data for the GNN.

Table 3.1: Simulation specifications for simulations used in this thesis.

Simulation index	N	ρ	Label dim.	f_a	Iterations	Samples	Resets
1	50	0.2	4	60	10^3	10^3	10^3
2	50	0.2	4	60	10^6	10^3	0
3	50	0.2	2	60	10^3	10^3	10^3
4	50	0.2	2	60	10^6	10^3	0
5	1000	0.2	2	3	10^6	10^3	0

3.2 Implementing an MPN to learn the dynamics of a system of particles

Before inputting the graph data into the MPN for training. Some pre-processing steps were taken. Firstly, all of the coordinates of the particles were scaled such that the side-length of the simulation box became one, and the orientations were scaled by dividing with 2π. This was in order to ensure that the scale of the input data (in terms of the elements' magnitudes) was the same for different simulation setups. This was needed since, as mentioned in section 3.1.1, the box side length varied when changing the number of particles, in order to retain a constant particle density. Aside from scaling, a portion (the initial frames) of the graphs were also portioned off to be used as validation data, in order to be able to test the GNNs force predictions on unseen data, and to detect overfitting. This portion was always some number of frames from the start of the simulation.
Since the simulations used in this project included a multitude of parameters, some way of checking the validity of the data was needed. Aside from plotting a video of the particle movements, histograms of the pair-wise distance of the particles in the simulations were also made. These histograms could give fast insight into the validity of the simulation, since clustering was expected, which would lead to more particles at close distances from each other.

3.2.1 MPN model parameters

As detailed in section 2.7, the MPN essentially consists of three functions, X, Ψ, and Ω applied at different stages. For the MPN used in this project, $\Psi = 1$, the two other functions were dense neural networks. Their dimensions are listed in table 3.2.

Table 3.2: The parameters that were used for the MPN.

Neural Network	Number of hidden layers	Nodes per hidden layer
X	3	$10, 40, 120$
Ω	1	120

3.2.2 Model training

When training the MPN, the Keras fit function was used [16]. The model was compiled with the Adam optimizer, with a learning rate of 0.001. The loss function used

was the mean absolute error (MAE) of the model's prediction of the node labels. This loss was monitored during training. The MAE for the model's predictions on the validation dataset was also calculated in order to check for overfitting during training (when the model fails to learn the overarching properties of the data distribution that the training set originates from, and instead learns properties inherent only to the training dataset).

3.2.3 Qualitative validation

Once the training had been finished, or paused, some method was needed for qualitatively analyzing the type of prediction mistakes that the model was making. The method used for this was to simply plot and compare the model's force predictions for a chosen force component to the targets for an arbitrary choice of particles. This was done with both the training and validation splits. In this way, it could be seen if the model was using a strategy of predicting all of the forces to be the mean (predicting always 0, since the force's direction is uniformly distributed), and to see if a certain type of force was easier to predict than another one.

3.2.4 Extracting the forces

If the qualitative validation showed promise, the final analysis could be made. As mentioned, the goal of the project was to characterize the forces at play in the system. The training input for the model were instantaneous particle coordinates, orientations, and their separation distance. The method used for extracting the force predictions from the network was to use a new simulation with only two particles in various configurations. To extract the active force, the simulation code was re-used to simulate two particles that were allowed to rotate while being placed far enough away from each other to have no inter-particle interaction. If the active and passive forces were correctly learned, the predicted active force should be the only one which is non-zero, and therefore, if the output was two-dimensional, the predicted total force, $\mathbf{F}$, would be equal to the instantaneous active force (in the four-dimensional case, the active force is already separated). Since the particle was allowed to rotate, the active force with respect to orientation $\mathbf{F}_a(\theta)$ could thus be extracted, by simply sampling the full force prediction of one of the particles in different time-steps.
To extract the passive force, a similar method was used. Here, two particles were allowed, with constant orientations θ_1, θ_2, to travel towards each other. The passive force for particle at some time-step t as

$$\mathbf{F}_p(t) = \mathbf{F}(t) - \mathbf{F}_a(\theta_1). \tag{3.1}$$

in this way, and by sampling the passive for different time-steps, the passive force with respect to distance is gained.
It should be noted that using the simulation code to generate the artificial input data for these force extractions was a convenient, but not essential method. Another way of generating the data would be to simply construct artificial simulation results by hand, and generating graphs based on it. The result would have been the same.

4
Results

In this section the main results are presented. First, the quality and characteristics of the simulations is examined, followed by results from the training process, and the final force estimations.

4.1 Simulation results

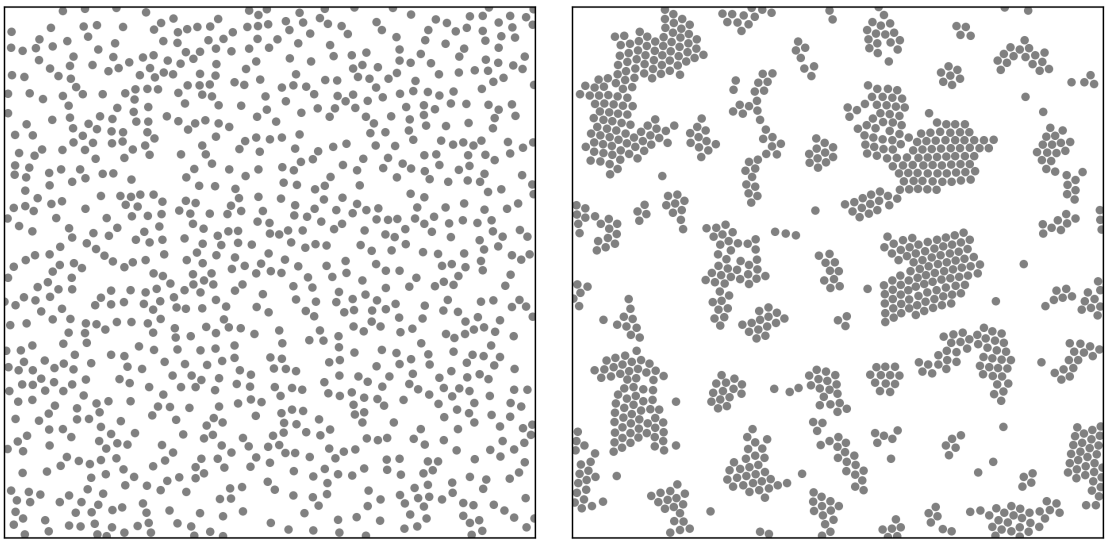
In figure 4.1, a visualization of a typical simulation result is shown. As can be seen, the clustering formation predicted in section 2.1.1 is achieved, which adds validity to the simulations. Additionally, in figure 4.2, the collisions and clustering of the particles is observed as a spike for the shortest possible distances. The potential wall for distances below 1 result in a high repulsive force which make it unlikely to find particles at such a distance. In figure 4.3, the pair-wise distance histograms of simulations 1-4 is are shown. Here, particle collisions can again be observed as a spike for the smaller distances. In comparison to simulation 5, simulations 2 and 4 likely had a lesser degree of MIPS formations due to the high active force allowing particles to escape clusters more easily (observed by viewing animations of the particle simulations).

Table 4.1: Mean absolute error (MAE) with respect to the simulation used. Note that the passive force MAE is only calculated for the distances 1.1 - 3.0, and refers to the discrepancy between the model's prediction and the true force value w.r.t. orientation or distance when using the method described in section 3.2.4 to extract the forces.

Simulation index	Active force MAE	Passive force MAE [1.1 - 3.0]
1	1.22	0.45
2	0.62	0.21
3	1.05	0.21
4	0.37	0.09

4.2 MPN performance

With simulations 1-4 (detailed in section ref 3.1) as training data, the MPN was able to estimate the active force and passive force to such a degree that the essential



(a) Snapshot from the start of a simulation.

(b) Snapshot from the end of a simulation.

Figure 4.1: Two snapshots of simulation 5 (as described in section 3.1). Figure 4.1a shows more clustering of particles. The axes are scaled such that one unit length corresponds to the diameter σ, of a particle. Here, a

features of the functions were clearly visible. In some cases, the estimation was almost exact.

Table 4.1 lists the mean absolute error of the force predictions for the four simulations that were used. In this table it can be observed that the fourth simulation gave the best results for extracting the correct force from the system of particles. Furthermore, figure 4.4 shows the evolution of training loss and validation loss during the training. For this figure, it should be noted that the active force, with its large magnitude resulted in a large error in the beginning of the training. The fast error decrease in the first epochs corresponds with the MPN learning the active force (confirmed by stopping training early and investigating the network's qualitative performance on the two different forces). Simultaneously it could be observed that the network favored finding the mean of the passive force (zero) early on, and always predicting the passive force to be its mean. The network began learning the variations in the passive force only after a relatively long training time.

4.2.1 Active force

Predicting the active force was a simple task for the network. When comparing the results in figure 4.5, the fourth simulation generates the most accurate prediction. Overall, the greatest prediction error tends to be found close to the limits 0 and 2π radians. When qualitatively comparing the mixed simulations to the non-mixed ones, the the fully simulated, non-mixed simulations appear to deviate less from the correct active force function. This agrees with the quantitative comparison in table 4.1.

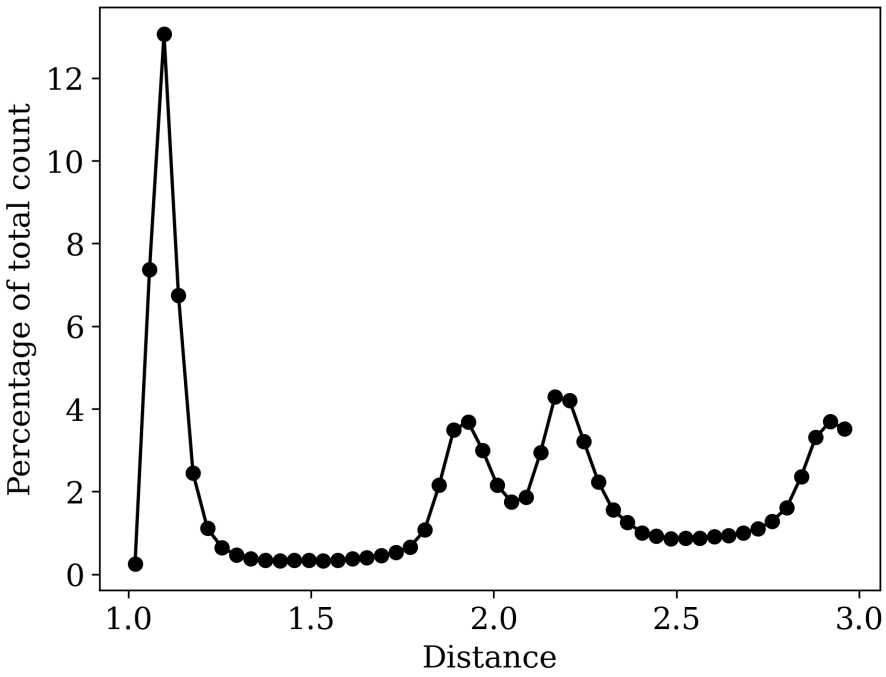
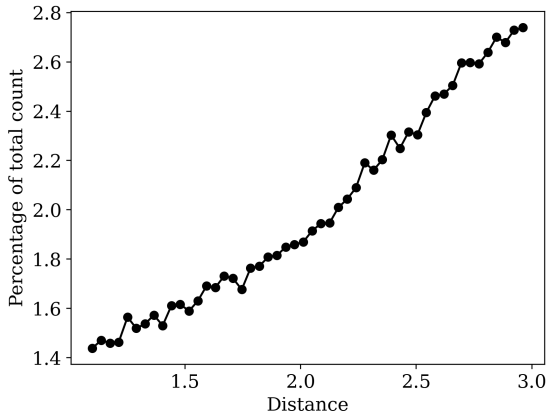


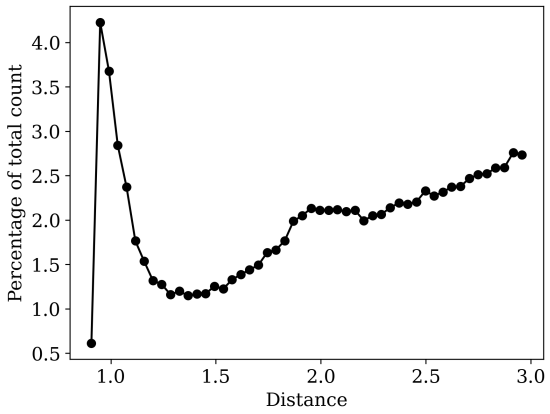
Figure 4.2: A histogram of the pair-wise distances between particles over all of the sampled frames of the simulation shown in figure 4.1. All possible pair-wise distances are grouped into 50 bins. The total count for each bin is represented by a point.

4.2.2 Passive force

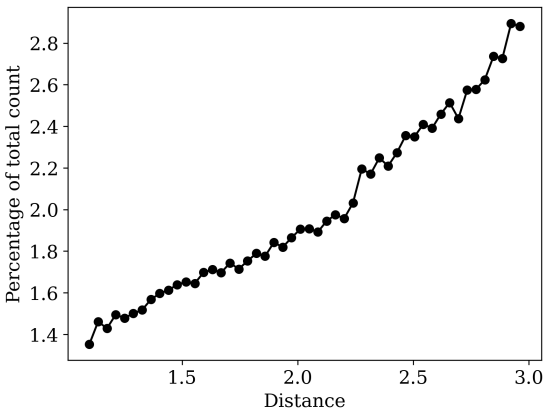
Despite the post-processing method for extracting the passive force for simulations 3 and 4 was more complex than for the other two, the results show that little if any error could be derived from this. In figure 4.6, the results from the model's passive force estimation is shown for the four simulations. As stated in section 3.1, all particles in simulations 1 and 3 are randomly placed for each frame with a maximum distance of 1.1. Therefore, the model's predictions for distances shorter than 1.1 are irrelevant for figures 4.6a and 4.6c. With this taken into account, a qualitative analysis of the figures still show that the models that were trained on fully simulated data more closely estimated the correct force. It can also be observed that out of the two fully simulated datasets, only simulation 4 resulted in a model that correctly estimates the passive force resulting from the TSLJ potential wall at small distances. The incorrect force estimate for simulation 2 is zoomed in on in figure 4.7.



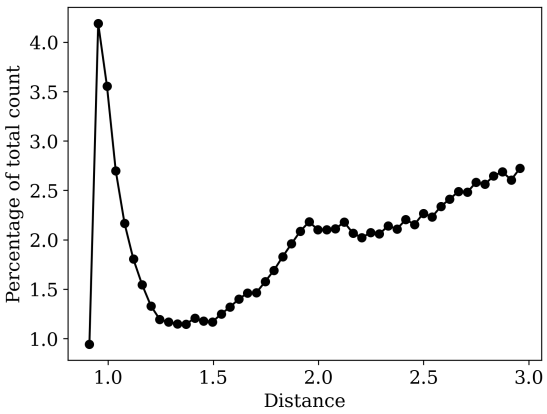
(a) The pair-wise distance histogram for simulation 1.



(b) The pair-wise distance histogram for simulation 2.

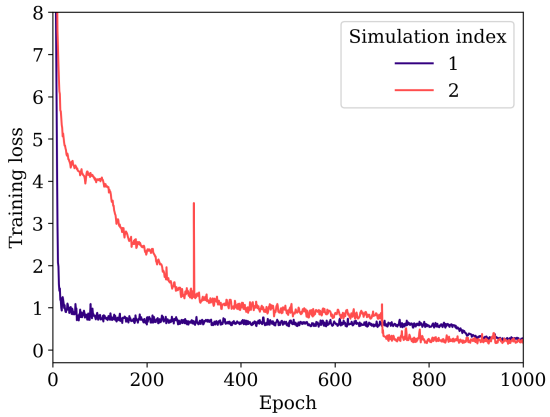


(c) The pair-wise distance histogram for simulation 3.

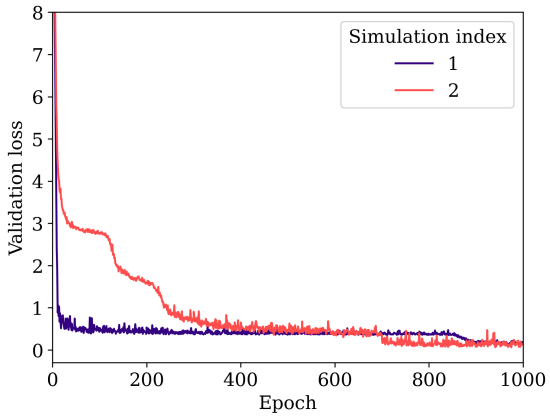


(d) The pair-wise distance histogram for simulation 4.

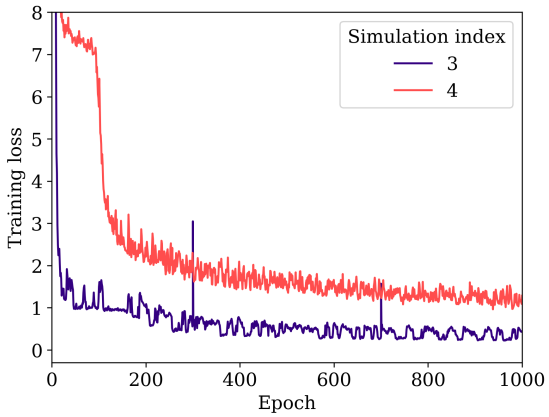
Figure 4.3: The pair-wise distance histograms for simulations 1-4. All possible pair-wise distances are grouped into 50 bins. The total count for each bin is represented by a point.



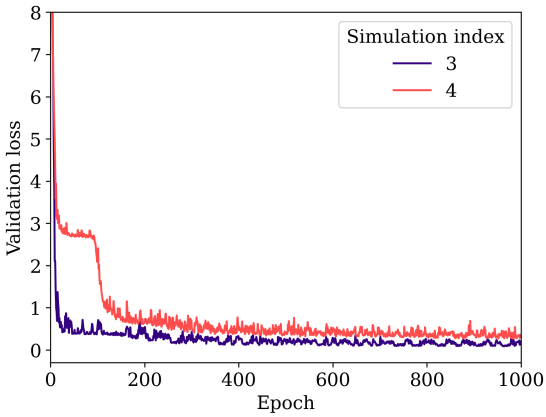
(a) The training loss history of simulations 1 and 2.



(b) The validation loss history of simulations 1 and 2.

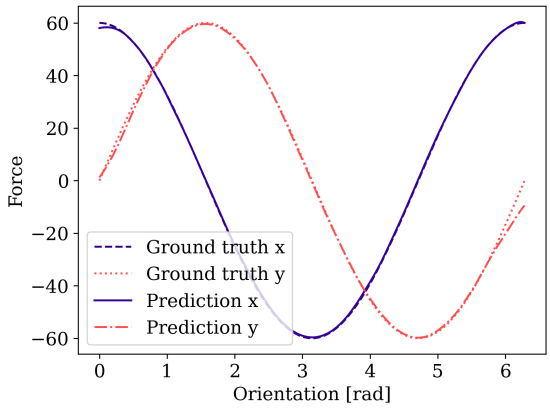


(c) The training loss history of simulations 3 and 4.

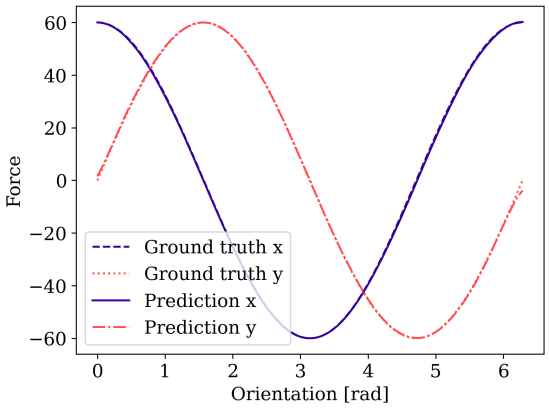


(d) The training loss history of simulations 3 and 4.

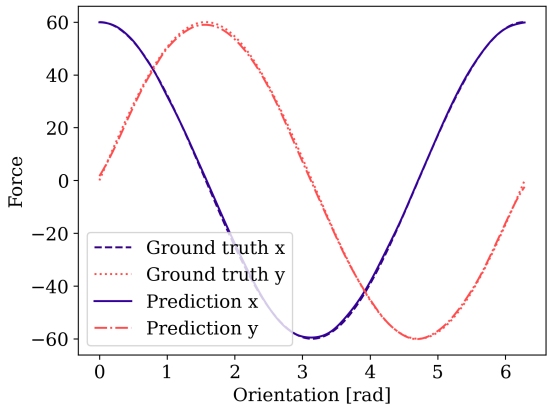
Figure 4.4: The training loss and validation loss during the training of the MPN on datasets from four different simulations.



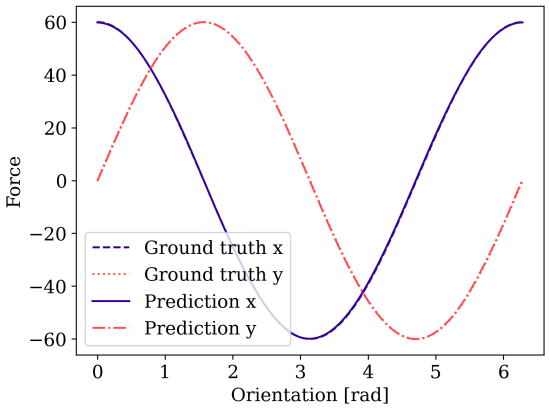
(a) The MPN predictions of the active force using training data from simulation 1.



(b) The MPN predictions of the active force using training data from simulation 2.

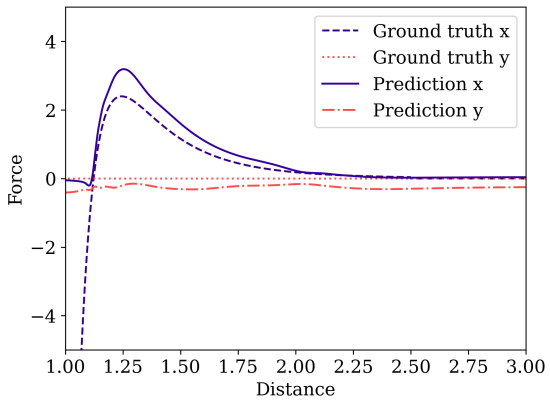


(c) The MPN predictions of the active force using training data from simulation 3.

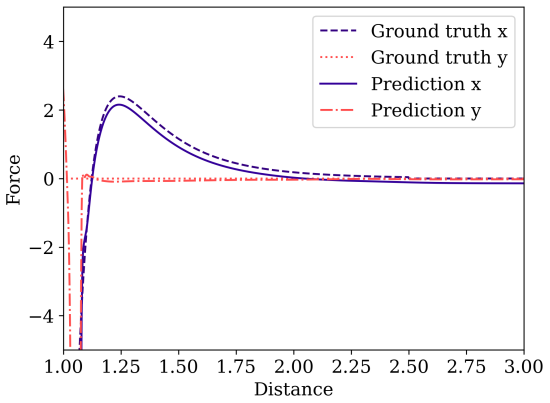


(d) The MPN predictions of the active force using training data from simulation 4.

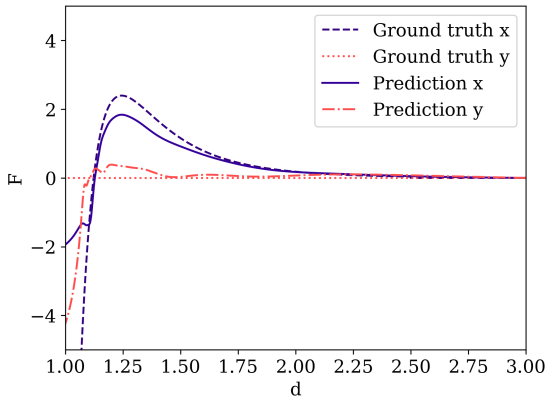
Figure 4.5: The MPN active force prediction results for simulations 1-4. Note that for simulation 1 and 2, the method of acquiring the active force was different from the method for 3 and 4, as described in section 3.2.4.



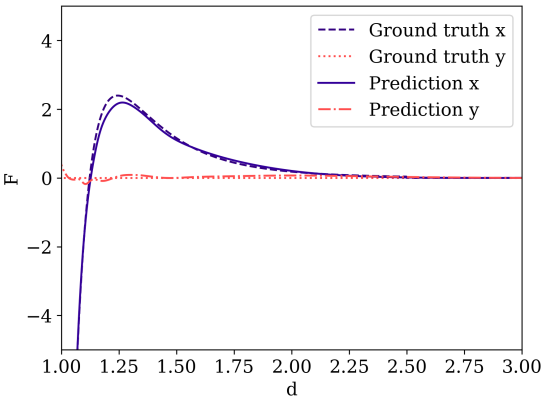
(a) The MPN predictions of the passive force using training data from simulation 1.



(b) The MPN predictions of the passive force using training data from simulation 2.



(c) The MPN predictions of the passive force using training data from simulation 3.



(d) The MPN predictions of the passive force using training data from simulation 4.

Figure 4.6: The MPN passive force prediction results for simulations 1-4. Note that for simulation 1 and 2, the method of acquiring the active force was different from the method for 3 and 4, as described in section 3.2.4. Also, the closest distance that particles where placed from each other in simulations 1 and 3 was 1.1σ.

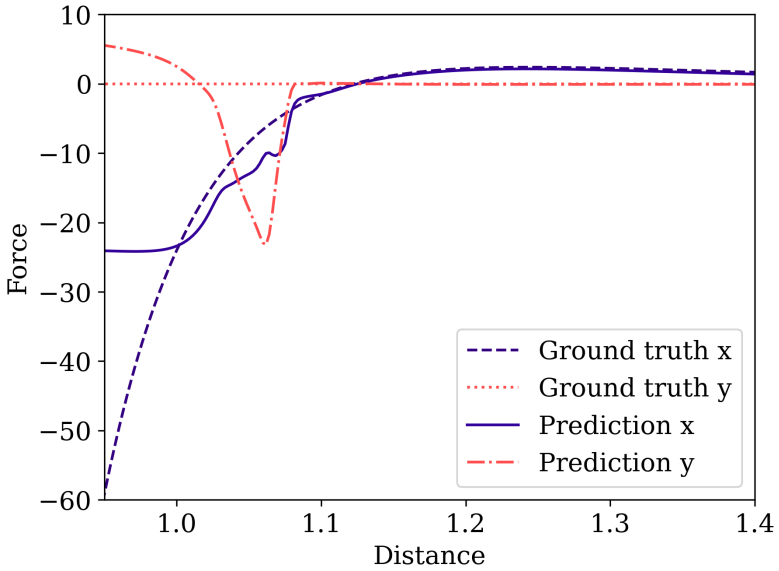


Figure 4.7: A zoom-in on the bottom left part of figure 4.6b, highlighting a range of distances where the force predictions were incorrect.

5
Discussion

Here follows some interpretations from the results, some additional details, and an outlook on the next steps of the project.

5.1 MIPS formation indicated in histograms

When comparing figures 4.2, 4.3b, and 4.3d, and taking into account that there was more MIPS formation for simulation 5, one can argue that the double spike around $d = 2$ in figure 4.2 is indicative of MIPS formation. In a cluster, many particles will be directly adjacent to each other, but if the cluster is large enough, particles would also have neighbors removed by roughly two particle diameters. Inspecting figure 4.1b, one can see that there are many large clusters in which many particles are at a distance of roughly 2 or 3, explaining the occurrence of the peaks around 2 and 3 in the histogram. In the histograms for simulation 2 and 4, the small bump around a distance of 2 likely indicates that there likely is some MIPS formation, but also that it's not strong.

5.2 Choice of validation set

The validation set that was used to measure the MPNs performance during training was picked from the first frames of the simulation. This likely results in the validation set being somewhat different, with more dispersion, in comparison to the training set. If the project were to have continued, this would have been changed such that the validation set is picked from random frames in the simulation data.

5.3 Evaluation of the choice of targets

As can be seen in figure 4.4, the loss decrease had stagnated by the end of the training, but there is still a chance that some changes would occur if training would have been continued. Thus, whether the fourth simulation can be expected to give the best result after another 1000 epochs of training is unknown.
It is surprising that the models trained on fully simulated data were more successful in estimating the correct passive force even for the longer distances. As seen in the histograms in figure 4.3, the non-simulated data contained a higher amount of particle pairs at long distances, and thus one would expect the long-range predictions

to be better in this case. Instead, the fully simulated data resulted in more accurate force estimates for any distance.
When using the 4-dimensional force labels, it was expected that the MPN would be assisted in the training process, since the separation of the forces' identity (active versus passive) had already been done for the network in a sense. The results did not follow these expectations, however, since both the qualitative analysis of the passive force predictions and table 4.1 show that the model achieved better predictions when trained on simulation 3 instead of 1, or 4 instead of 2. One possible explanation could be that the inductive bias in this case constrains the output dimension of the network. In turn this also constrains the architecture of the network, possibly forcing it to find a more complex solution to a problem that it could solve with a more simple representation had the output dimension been smaller.

5.3.1 Training time

With the setup used in this project, one epoch (which was composed of 32 frames being analyzed) took around twelve seconds to complete. Thus, 1000 epochs, which was the length of training for all simulation types, took around three hours and twenty minutes to complete.

5.4 Comparison to ActiveNet

In the same way as with the MPN of this project, ActiveNet was shown to correctly predict the magnitude of the force on a simulated system of particles based on the exact same dynamics as was used in this project. ActiveNet was also successfully tested on other types of potentials [5]. Most of the work for implementing these potentials into the simulation program of this project has already been done. However, the amount of results to analyze when using multiple potentials would be large, so for the sake of conciseness and efficiency, only the TSLJ potential was used.

5.5 Future work

A long term goal of the project was to use MAGIK to possibly remove the constraints that ActiveNet depended on, and to explore the more complex problems that could be solved with this more advanced GNN. The time constraints of the project coupled with unforeseen obstacles in the work process made this untenable. However, the architecture of MAGIK essentially includes and builds on an MPN architecture, meaning that the tasks solved by the MPN in this project are likely also solved with MAGIK architecture.
Another potential future addition to the project could be to use the same procedure again but on real, experimental data, where one has a video of particles moving. By using particle tracking software, such as DeepTrack, one could transform the videos into frames of nodes with encoded node features, and thereafter transform these frames into graphs in the same way as in this project. The labels would be constructed as the estimated instantaneous velocity of each particle. After training

an MPN or MAGIK on this data, forces could potentially be investigated and characterized by re-using the trained models on new data as in section 3.2.4. Of course, this specific method still works under the assumption of the overdamped regime, and it would be interesting to see if an analogous method could be devised for other cases.

6
Conclusion

In conclusion, it was proven that an MPN can be used to gain an approximation of the forces present in a system of interactive, active particles. Additionally, the active matter simulation setup described in this thesis was found to be likely valid, as it produced the MIPS phenomenon. Since the training of the MPN was done exclusively in DeepTrack, the MPN architecture can easily be substituted by other models such as MAGIK.

Bibliography

[1] Wu, Zonghan, et al. “A Comprehensive Survey on Graph Neural Networks.” IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, Jan. 2021, pp. 4–24. arXiv.org, https://doi.org/10.1109/TNNLS.2020.2978386.

[2] Cranmer, Miles, et al. Discovering Symbolic Models from Deep Learning with Inductive Biases. arXiv, 17 Nov. 2020. arXiv.org, https://doi.org/10.48550/arXiv.2006.11287.

[3] Das, Moumita,et al. ”Introduction to Active Matter”. Soft Matter, vol. 16, no 31, 2020, p. 7185–90. pubs.rsc.org, https://doi.org/10.1039/D0SM90137G.

[4] Tailleur, J., och M. E. Cates. ”Statistical Mechanics of Interacting Run-and-Tumble Bacteria”. Physical Review Letters, vol. 100, no 21, May 2008, p. 218103. arXiv.org, https://doi.org/10.1103/PhysRevLett.100.218103.

[5] Ruiz-Garcia, Miguel, et al. Discovering dynamic laws from observations: the case of self-propelled, interacting colloids. 2, arXiv, 21 April 2023. arXiv.org, https://doi.org/10.48550/arXiv.2203.14846.

[6] Pineda, Jesús, et al. Geometric Deep Learning Reveals the Spatiotemporal Fingerprint of Microscopic Motion. arXiv, 13 Feb. 2022. arXiv.org, https://doi.org/10.48550/arXiv.2202.06355.

[7] Midtvedt, Benjamin, et al. DeepTrack2. 2020. 19 June 2023. GitHub, https://github.com/softmatterlab/DeepTrack2.

[8] Lennard-Jones, J. E. “Cohesion.” Proceedings of the Physical Society, vol. 43, no. 5, Sept. 1931, p. 461. Institute of Physics, https://doi.org/10.1088/0959-5309/43/5/301.

[9] Stenhammar, Joakim. An Introduction to Motility-Induced Phase Separation. arXiv, 9 Dec. 2021. arXiv.org, https://doi.org/10.48550/arXiv.2112.05024.

[10] Argun, Aykut, m.fl. Simulation of Complex Systems. IOP Publishing, 2021. DOI.org (Crossref), https://doi.org/10.1088/978-0-7503-3843-1.

[11] Units Command — LAMMPS Documentation. https://docs.lammps.org/units.html. Accessed 23 May 2023.

[12] Mehlig, Bernhard. Machine Learning with Neural Networks: An Introduction for Scientists and Engineers. 1st ed., Cambridge University Press, 2021. DOI.org (Crossref), https://doi.org/10.1017/9781108860604.

[13] Rahman, Md. Saidur. Basic Graph Theory. Springer International Publishing, 2017. DOI.org (Crossref), https://doi.org/10.1007/978-3-319-49475-3.

[14] Karagiannakos, Sergios. “Best Graph Neural Network Architectures: GCN, GAT, MPNN and More.” AI Summer, 23 Sept. 2021, https://theaisummer.com/gnn-architectures/.

[15] “CuPy.” CuPy, https://cupy.dev/. Accessed 17 May 2023.

[16] K. Team, “Keras documentation: Model training APIs.” Available: https://keras.io/api/models/model_training_apis/. [Accessed: May 25, 2023]

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY