LIÈGE université
Sciences Appliquées

University of Liège
Faculty of Applied Sciences

Interpretable Graph Neural Networks for Cellular Systems

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in Data Science and Engineering

Author
Jérôme Pierre

Advisors
Gilles Louppe
Namid Stillmans

Academic year 2023-2024

Acknowledgements

First of all, I would like to warmly thank my supervisor, Gilles Louppe, and my co-supervisor, Namid Stillmans, for their expertise and availability throughout this work. I am truly grateful for the valuable time they dedicated to me.

I would also like to express my gratitude to the members of my jury for the time and attention they devoted to reading and evaluating this work.

I also want to thank my friends, especially Stéphane Clemens and Eliott Thommes, for their invaluable help and support throughout this marathon that is the writing of a final thesis. Your support has been invaluable.

Finally, I want to thank my family for their support, without which I would never have made it this far.

Abstract

Cell migration is a critical process that plays a fundamental role in a wide range of biological phenomena, including embryogenesis, wound healing, and immune responses. Despite its significance, accurately modelling the complex dynamics of cellular migration remains challenging.

This thesis investigates the use of graph neural networks (GNNs) to simulate the dynamics of cellular migration systems. In particular, this work explores the efficacy of several message-passing GNN architectures including a GaT and classical message-passing GNNs. The performances of these models are evaluated with a focus on their ability to replicate ground truth statistics.

Beyond simulation, a key objective of this thesis is to enhance the interpretability of the models by recovering the underlying mathematical expressions of the interactions governing cellular interactions. To achieve this, L1 sparsity regularization is applied to the messages of a single-layer GNN. This results in a dimension reduction that enables the use of genetic programming for symbolic regression.

Recognizing the challenges posed by symbolic regression in the context of highly complex analytic equations, this thesis introduces a methodology that decomposes the messages into a weighted sum of basis functions. Applying symbolic regression on the weights provides a more tractable means to recover the governing equations.

Contents

Figures

Chapter 1

Introduction

Physics is fundamentally concerned with developing models that predict natural phenomena. Two primary approaches are commonly employed to derive these models. The first one is a concept-based approach. Scientists propose hypotheses, such as energy or mass conservation, and use mathematical reasoning to derive governing equations. The second approach is data-oriented and consists in analyzing the correlation between different observed variables to develop empirical theories. A prominent example of this method is Kepler's discovery of laws of planetary motion. In this work, the data-oriented approach is further explored by developing a model that utilizes graph neural networks (GNN). Numerous similar physical models exist in the literature, with applications in weather forecasting [1], chemistry [2], structural mechanics [3], and more. The first step of this work consists of implementing various graph neural networks to predict the dynamics of a cellular system of active matter by modelling the force between cells. This is a fundamental domain of research as cell migration is a critical process that plays a fundamental role in a wide range of biological phenomena, including embryogenesis, wound healing, and immune responses. Despite its significance, accurately modelling the complex dynamics of cellular migration remains challenging. Previous work [4] tried to solve this system by using large Graph Neural Networks. This work continues this line of research by exploring new architectures and applying specific physical inductive biases based on Occam's razor and Newton's laws.

Current challenges with neural networks-based physical models include a lack of interpretability and poor generalization when dealing with data outside their training distribution. However, physics offers a language that appears to avoid these limitations: algebraic equations. This motivates the next step, which involves applying sparsity regularization and symbolic regression to derive analytic equations that describe the forces within the system. It is important to note that this method diverges from that of Kepler. While Kepler identified laws characterizing planetary motions, the objective of this work is to uncover the interactions driving movement without any a priori knowledge of them.

The structure of this master's thesis follows these two steps. The development and assessment of a neural network to predict cell trajectories comes first. Different models and training schemes are explored. Then, symbolic regression is used to express the black box neural network with mathematical equations.

The implementation mainly relies on `pytorch` [5] and `pytorch_geometric` [6] and is available on github[1].

1.1 Contributions

The contributions of this work are as follows:

- This work explores and evaluates various Graph Neural Network architectures to predict the dynamics of cellular systems.
- It also demonstrates that the interactions between cells can be accurately recovered by applying sparsity regularization to the GNN messages, even in a non-Newtonian framework.
- Additionally, this thesis introduces a method for obtaining interpretable interactions with complex analytic expressions by decomposing the GNN emssags into a weighted sum of basis functions.

[1] https://github.com/Jepi1202/master_thesis

Chapter 2

Cellular System of Active Matter

Active matter refers to systems composed of individual agents that contain their own energy source and consume it to move, exert mechanical forces or induce mechanical changes. Such systems are inherently far from equilibrium which prohibits from solely using thermodynamic theory to model them. Additionally, active matter systems spontaneously generate collective motion. Numerous examples can be observed in nature at various scales. Macroscopic examples include bird flocks [7], schools of fish [8] and ant group motions [9]. However, many microscopic systems, such as cell migration [10] and colloids [11] also fall into the category of active matter. A deeper understanding of active matter is essential, as these systems significantly influence biological processes such as embryonic development and cancer progression.

This work focuses on the 2D dynamics of microscopic systems of cells. Since the introduction of active Brownian particles and the Vicsek model, many models have been developed [12] [13]. However, these systems are not without limitations. Employing GNNs to learn the dynamics of these systems could lead to new models that more accurately reflect empirical observations.

2.1 Physical simulator

In line with previous research [14], the simulator is inspired by studies on the effects of cell division and apoptosis on the collective dynamics of epithelial tissues [1] [15]. Specifically, inspired by collective cell migration, the motion is considered to be over-damped:

$$\dot{\mathbf{r}}_i = v_0 \hat{\mathbf{n}}_i + \frac{1}{\zeta} \sum_{j \in \mathcal{N}(i)} \mathbf{F}_{ij} \tag{2.1}$$

This equation indicates that the velocity $\dot{\mathbf{r}}_i$ depends on both cell-specific factors and interactions with the other cells in its neighbourhood $\mathcal{N}(i)$. The term $v_0\hat{\mathbf{n}}_i$ represents the speed of the cell if there is no interaction with other particles. In this scenario, the cell moves in the direction $\hat{\mathbf{n}}_i$ with speed v_0 which is referred to as the active propulsion force of the particle. Concerning the other term, the total interaction is scaled by the friction of

[1] Examples of ground truth simulations can be seen by going to the URL https://youtube.com/playlist?list=PLzXmrzZW9EJ4S2wFZHaIhKHtkzi-CaUxG&si=Kz1SAv-5vAaapLyc

the system ζ and the interaction $\mathbf{F}_{ij}$ between a cell of radius R_i and another one of radius R_j is modelled as a piecewise linear function of the distance r between the two:

$$F_{ij}(r) = \begin{cases} k(r - b_{ij}) & \text{if } \frac{r}{b_{ij}} < 1 + \varepsilon \\ -k(r - b_{ij} - 2\varepsilon b_{ij}) & \text{if } 1 + \varepsilon < \frac{r}{b_{ij}} \leq 1 + 2\varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$\text{where } b_{ij} = R_i + R_j \tag{2.3}$$

This equation introduces two new parameters ϵ and k. The dimensionless parameter ϵ controls the strength of attraction and defines the distance threshold $(1 + 2\epsilon)b_{ij}$ within which the interaction force is calculated. The other parameter, k, represents the stiffness of the interaction and controls the slopes of the force function.

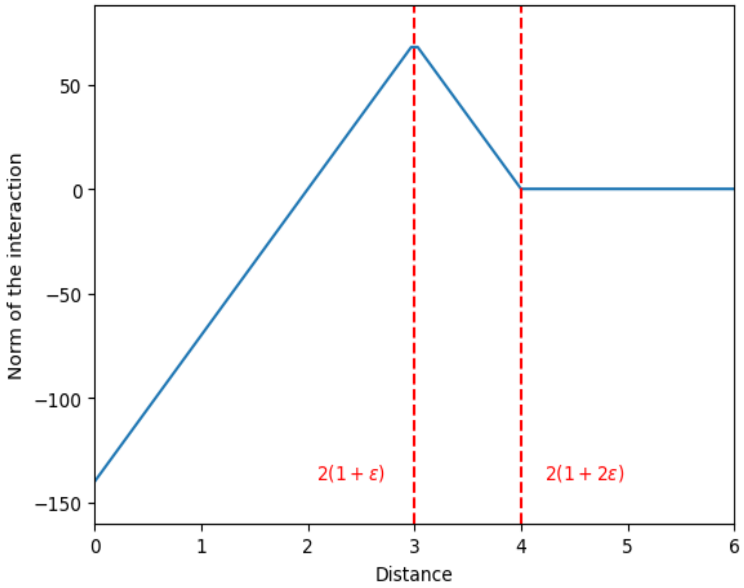


Figure 2.1: Illustration of the interaction with respect to the distance between cells.

Additionally, a noisy version of the simulator is considered by introducing Gaussian noise η to the angular variable θ of the cells at each timestep.

$$\dot{\theta}_i = \eta_i, \quad \langle \eta_i \rangle = 0, \quad \langle \eta_i(t)\eta_j(t') \rangle = \frac{1}{\tau}\delta_{ij}\delta(t - t') \tag{2.4}$$

Here, δ is the Kronecker delta and τ is the persistence timescale.

Cell trajectories are obtained by applying the Euler method to the equations detailed above with a time interval $\Delta t = 0.001$ hours. Moreover, the system is constrained within a square box of side length L, with boundary conditions that ensure cells experience elastic collisions with the walls.

It is important to note that the different variables in this project have arbitrary units. However, it is possible to rescale them to match empirical observations.

The initial positions of the cells also significantly impact the dynamics that are observed. In this work, the different cells are grouped by two and dispersed in a grid manner in the box. This choice was made to ease the training. The idea is based on the fact that learning a single force is more restrictive than learning the sum of forces. Therefore, having graphs composed of groups of cells affected by a single interaction is expected to improve the dataset quality.

Finally, unlike the initial model [15], neither cell death nor cell division is considered, which means that the number of cells N remains the same during the simulations.

Parameter	**Value**
nb steps	1000
Δt	0.001
N	$\mathcal{U}(100, 300)$
v_0	60
$\hat{\mathbf{n}}_i$	$\theta \sim \mathcal{U}(0, 2\pi)$
k	70
ϵ	0.5
ζ	1
τ	3.5
R	1
L	120

Table 2.1: Parameters of the physical simulator.

Part I

Predicting physical systems

Chapter 3

Graph Network-based Simulators

This chapter introduces the main notions that are needed to simulate particle systems with neural networks. It serves as both a literature review and a background section. It begins by introducing the concept of graph network-based simulations, followed by an overview of graph neural networks (GNNs) and some notions of physics-informed neural networks. A clear emphasis is put on message-passing graph neural networks which are the kind of neural network architecture explored in this work.

3.1 Introduction

A cellular system consists of N cells, which can be interpreted as the nodes $\{v\}_{i=1}^{N}$ of a graph $\mathcal{G} = (V, E)$, where $V = \{v\}_{i=1}^{N}$ is the set of nodes and $E = \{e_{i,j}\}$ is the set of edges between them. In practice, each cell influences each other in a pairwise manner, implying that the graph should be complete. However, similarly to other physical forces such as gravity or Coulomb's law, the influence between particles decreases with distance. Therefore, it is only necessary to maintain edges corresponding to forces with a magnitude above a certain threshold.

Graph Network-based Simulators (GNS) [16] involve designing a graph neural network that can predict the graph configuration of a system at the next timestep based on the current graph configuration. In this context, the task is to predict the next position of each cell at the node level.

By iteratively applying the GNS to its own previous output, it is possible to generate the trajectory of the cells over time. As illustrated in Figure 3.1, the goal is to develop a graph neural network GNN_θ with parameters θ that can predict the trajectory $\tau^{\mathcal{G}}$ of the graphs $\mathcal{G}_t$ for subsequent timesteps in a rollout manner. Denoting the different timesteps by t, this can be formally expressed as:

$$\tau^{\mathcal{G}} = \{(\mathcal{G}_t)\}_{t=0}^{T} \qquad \text{s.t.} \begin{cases} \text{GNN}_\theta(\hat{\mathcal{G}}_\text{t}) \Longrightarrow \hat{\mathcal{G}}_\text{t+1} \\ \hat{\mathcal{G}}_0 = \mathcal{G}_0 \end{cases} \tag{3.1}$$

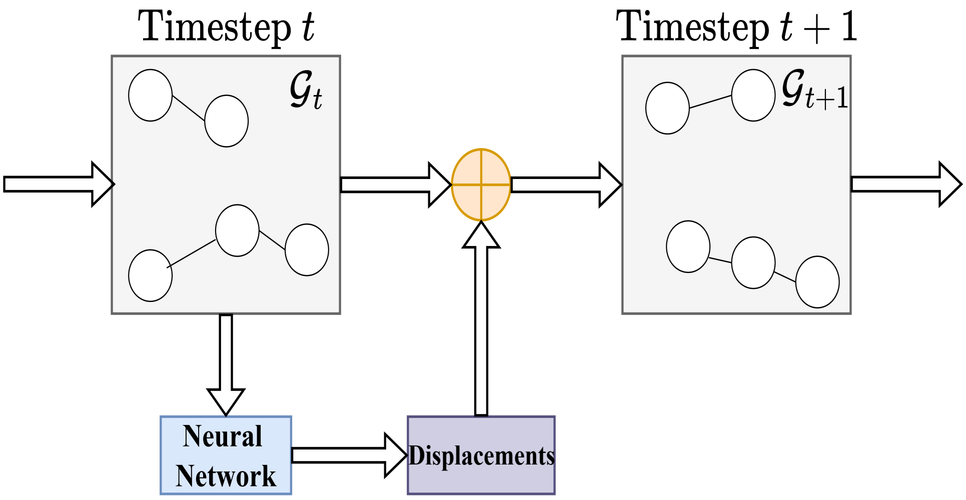


Figure 3.1: Illustration of the rollout simulation pipeline. Initially, a graph is fed into the neural network, which is a GNN in this work. The output generated by the neural network is then utilized to derive the next graph, which subsequently serves as the input for the next iteration of the neural network. Here it is done by computing the displacement of the cells.

The concept of GNS was formalized by Sanchez-Gonzales et al [16] for simulating water, sand and goop systems. It has since then been adapted in subsequent research for similar applications, such as granular flows [17] [18]. However, different papers were already trying to simulate physical systems with GNNs before this formalization [19] [20] [21] [22] [23]. The use of GNNs as simulators is particularly important because graphs are data representations that are highly suitable for a wide range of real-life applications beyond cellular systems. While it is possible to use classical Multi-Layer Perceptrons (MLPs) for graphs, they lack permutation equivariance which makes them require exponentially more data (O(n!)) compared to GNNs [24] [25]. Moreover, it was also shown that GNNs are generally more efficient than classical simulators when dealing with complex phenomena [26].

3.2 Graph Neural Networks

Graph Neural Networks are neural networks specifically designed for graphs and were first introduced in the mid to late 2000s [27]. Over the years, a variety of GNN architectures have been developed. Initially, most pioneer works relied on recurrent GNNs. Recurrent GNNs recursively apply layers with shared weights on a graph. Later, other architectures emerged, such as convolutional graph neural networks (ConvGNNs), graph auto-encoder [28], and spatial-temporal graph neural networks [29]. For a comprehensive overview, readers can refer to the review by Wu et al [30].

ConvGNNs are similar to Recurrent GNNs. The main difference is that they do not share the weights across the different layers. In this work, the focus is exclusively on message-passing GNNs [31]. Message-passing GNNs are a type of spatial ConvGNNs that can be viewed as a specific case of graph networks [24] without graph properties. It is also possible to give additional symmetries to the GNNs [32] [33] but this is not explored here.

Message-passing GNNs

This work relies on message-passing GNNs to predict cell dynamics. Message-passing GNNs consist of an edge model ϕ^e and a node update model ϕ^v. The edge model ϕ^e is used to model the interaction between the different nodes in the graph, while ϕ^v updates a given node based on the aggregated results of ϕ^e. In the notation used in this work, $\mathbf{v}_i$ denotes the features of the node i, $\mathbf{e}_{(i,j)}$ represents the features of the edge between nodes i and j and $\mathcal{N}_i$ is the set of nodes that have edges with node i. A GNN layer then produces the updated features $\mathbf{v}'_i$ for node i according to the equation illustrated in Figure 3.2.

$$\mathbf{v}'_i = \phi^v\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} \phi^e(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{(i,j)})\right) \tag{3.2}$$

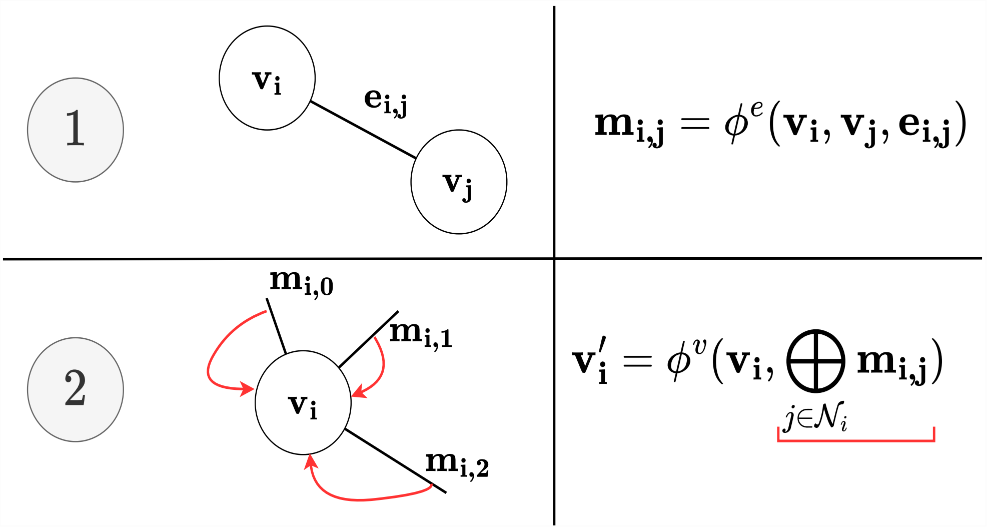

1
$\mathbf{v_i}$
$\mathbf{e_{i,j}}$
$\mathbf{v_j}$
$\mathbf{m_{i,j}} = \phi^e(\mathbf{v_i}, \mathbf{v_j}, \mathbf{e_{i,j}})$
2
$\mathbf{m_{i,0}}$
$\mathbf{m_{i,1}}$
$\mathbf{v_i}$
$\mathbf{m_{i,2}}$
$\mathbf{v'_i} = \phi^v(\mathbf{v_i}, \bigoplus_{j\in\mathcal{N}_i} \mathbf{m_{i,j}})$

Figure 3.2: Illustration of a message passing GNN. First, messages are computed using ϕ^e and then node features are updated by ϕ^v and the agglomerated messages.

In this paradigm, the nodes have access to the information of their direct neighbours. However, it is common practice in the literature to apply multiple message-passing layers consecutively, allowing nodes to access information from more distant neighbours [24]. While this can lead to better performances, it can also make the network less interpretable.

3.3 Physics-informed Neural Network

Given that we are dealing with physical systems, it is beneficial to incorporate our knowledge of physics into the neural network through various inductive biases. This can be done in multiple ways.

A popular approach is to ensure sparsity in the model, leveraging the principle of Occam's razor. Occam's razor is a philosophical principle that recommends using the simplest possible models to explain natural phenomena. For example, it was shown that messages containing the same number of elements as the number of spatial dimensions have better generalization performances [34]. Similarly, Cranmer et al [35] applied sparsity regularization on the messages in the edge model ϕ^e of a single-layer GNN to impart physical meaning. This work is greatly inspired by this paper.

Another potential strategy is to enforce the action-reaction principle. Since this study only considers closed systems, one approach could be to ensure that the sum of the interactions across all nodes equals zero. However, this approach is less restrictive than directly enforcing that opposite messages are opposite ($\phi^e(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{(i,j)}) = -\phi^e(\mathbf{v}_j, \mathbf{v}_i, \mathbf{e}_{(j,i)})$) [36].

There are still several ways to incorporate physical principles when working with GNNs. One method is to enforce physical symmetries through data augmentation or feature selection. For instance, it was shown that using relative positional information instead of absolute ones enhances performances [16]. Other notable examples consist in forcing the neural network to follow the Lagrangian or Hamiltonian formalism [37] [38].

Chapter 4

Architecture choices

This chapter presents the different neural network architectures employed in this work, along with the motivations behind the design of the features and the training procedures.

4.1 Neural networks

This work focuses exclusively on message-passing neural networks. Specifically, three distinct neural network architectures are examined: an Interaction Network [19] and two Encoder-Process-Decode (EPD) architectures with different GNN layers [16]. One of the EPD architectures employs Graph Attention Networks (GaT) [39], while the other relies on a GNN layer similar to the Interaction Network.

All of these neural networks adhere to the structure described in equation 3.2 where the aggregation operation ($\bigoplus$) of the messages is implemented as a summation ($\sum$). Moreover, the update model in these architectures processes a concatenation of the summed messages and the node features.

Interaction network and Baseline

The interaction network was the first GNS to scale up to real-world problems. It is designed as a single-layer message-passing GNN with no encoder nor decoder. In this implementation, the edge model ϕ^e and the update model ϕ^v are 3-layer MLPs with leaky ReLU activation functions. As recommended in the literature, the weights of both these MLPs are initialized following He initialization [40]. The specificity of this implementation is that a normalization layer is used right after the concatenation of the messages and the node features. In this case, by denoting the layer normalization with $\mathcal{LN}$ equation 3.2 becomes.

$$\mathbf{v}_i' = \mathrm{MLP}_\theta^1 \left[\mathcal{LN} \left\{ concat(\mathbf{x}_i, \sum_{j \in \mathcal{N}_i} \mathrm{MLP}_{\theta_2}^2(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{(i,j)})) \right\} \right] \tag{4.1}$$

By default, messages with 128 features are used in this project. However, inspired by previous work [34], which demonstrated improved generalization performance when using the same number of features as spatial dimensions, an alternative version of the

Interaction Network with messages of size 2 is also implemented. This variant is referred to as the baseline network.

Encode-Process-Decode GNNs (EPD)

Motivated by previous promising results [16], two different EPD architectures are considered. As depicted in Figure 4.1, EPD GNNs consist of a sequence that includes an encoder, multiple rounds of message-passing layers, and a decoder applied to a graph. The features obtained by the k^{th} GNN layer within the processor are passed as inputs to the subsequent GNN layers. A skip connection is also incorporated into the architecture to reach better gradient backpropagation. In this architecture, both node and edge features are encoded using 3-layer MLPs with leaky ReLU activation functions. As in the interaction network, He initialization is used to initialize the weights [40].

One of the investigated EPD architectures relies on the same architecture as the Interaction Network for the GNN layers of the processor. This architecture will be referred to as EPD-GNN.

The other network relies on attention. Since its introduction [41], attention has had large repercussions in many fields of deep learning. It is then only natural to assess how effective it is with graphs. The last architecture relies on graph attention networks (GaT) [39]. A GaT is a specific kind of message-passing GNN that employs the attention mechanism to weight the different edges connected to each node. The GaT layer used in this project are implemented in the following manner:

$$\mathbf{v}_i' = \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \cdot \mathbf{\Theta}_t \mathbf{v}_j \tag{4.2}$$

$$\text{with } \alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\Theta_s \mathbf{v}_i + \Theta_t \mathbf{v}_j + \Theta_e \mathbf{e}_{i,j}\right)\right)}{\sum\limits_{k \in \mathcal{N}(i)} \exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\Theta_s \mathbf{v}_i + \Theta_t \mathbf{v}_k + \Theta_e \mathbf{e}_{i,k}\right)\right)} \tag{4.3}$$

In this expression, Θ_s, Θ_t and Θ_e are matrices that represent the importance of the source node, the target node, and the edge, respectively. The term $\mathbf{a}$ denotes a linear layer.

Attention mechanisms can be particularly beneficial in this context. They can provide a means to access the weights of different edges, which should intuitively correlate with the strength of interactions between cells. Previous research has investigated the use of Graph Attention Networks in cellular migration models [4]. In this work, a similar architecture as the one developed in the previous research is developed to compare the performance of attention mechanisms against more traditional models. The only difference between this model and the one developed in [4] is that this one does not rely on transformers for the encoding and the decoding.

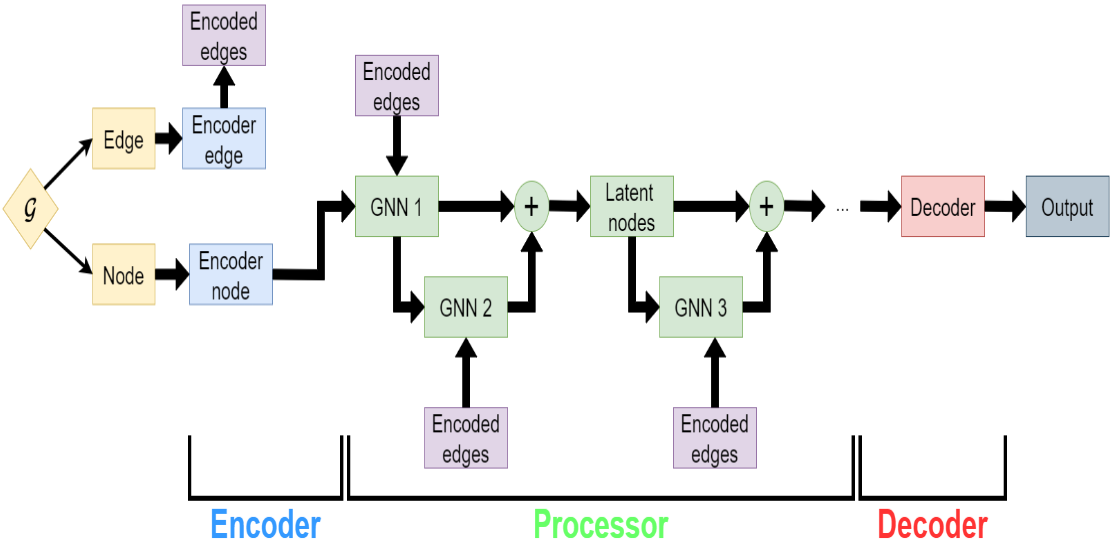


Figure 4.1: Architecture of an Encoder-Processor-Decoder GNN. First, the edges and the nodes are encoded into latent features by using two MLPs. Then, the newly obtained graph is sent in the processor. The latent nodes are then updated by a succession of GNN layers that also take as input the latent edges. Finally, the latent nodes are sent through another MLP that is used to decode them into displacements.

4.2 Features

The graph used as input by the different neural networks is constructed as follows. Nodes are obtained by concatenating the 4 previous velocities. Edges, on the other hand, are constructed by concatenating all the features that could influence physical interactions between cells such as the distance between the two cells, the cosine and the sine of the line linking both cell centres and the cell radius. The graphs that we are dealing with are not complete, edges are only created when the distance between two cells is smaller than six.

The rationale behind this choice of edge features is to eliminate the need for node features in the edge model ϕ^e. This approach significantly simplifies the architecture and the symbolic regression part of this project. This decision can also be intuitively motivated. Indeed, it is rare for interactions to depend on the speeds of the particles. Therefore, it should be fine to get rid of them. It should be noted that the cell radii are identical in our simulations. Initially, the goal was to start by developing a model that handles identical radii and then extend it to the general case. However, due to timing constraints, only identical radii are considered in this work.

It is important to note that the absolute positions of the cells are not represented in the graph. Previous research has demonstrated that preserving relative features can enhance the performance of the model by forcing the model to yield functions that automatically verify translation invariance [16]. Unfortunately, applying a similar strategy to enforce rotational invariance is more tricky. An alternative solution would be to apply some specific data augmentation but such techniques were not necessary in this work. The only data processing step applied during inference involves sorting the edges of the graph, which is done for algorithmic considerations.

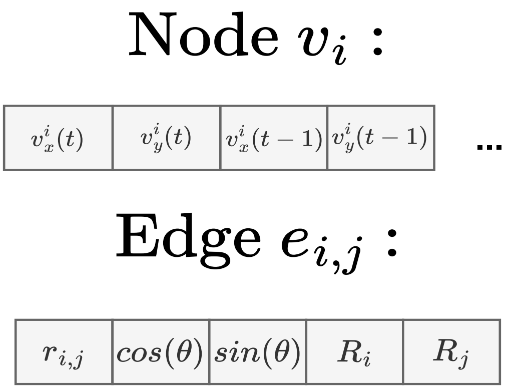


Figure 4.2: Illustration of the node features and the edge features.

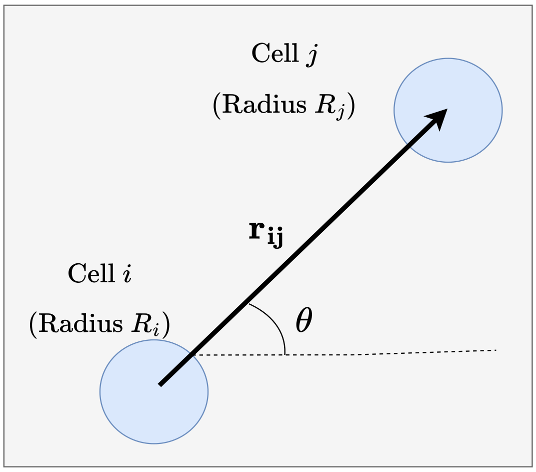


Figure 4.3: Illustration on the creation of the edge features.

4.3 Training:

Two main strategies can be used to train neural networks for rollout simulations: training on one-step transitions or directly optimizing the rollout simulation. Each approach has its own drawbacks.

Training on one-step transitions ensures that the neural network always receives exact ground truth inputs during training. However, during inference time, the network's predictions may deviate from the ground truth, causing errors that propagate and accumulate over time. This error accumulation significantly affects the predicted trajectories. A common solution to mitigate this issue is to add Gaussian noise to the network's features during training. Even though it is typical to apply a uniform standard deviation across all features, it was shown to lead to large relative errors in this case. Therefore, 0 mean Gaussian noise with different standard deviations for each feature is used in this work. More precisely, velocities are perturbed with Gaussian noise characterized by standard deviations of 1% of their absolute value, trigonometric features by standard deviations around 0.02 (corresponding to a 3-degree error 99% of the time) and distances by a standard deviation of 0.02.

The challenge with directly optimizing rollout simulations is that the neural network may learn the mean trajectory. To address this, one can either weight the transitions differently or incrementally add transitions while training. In this work, the latter was chosen. Specifically, the number of steps is increased by one every two epochs until a maximum sequence length of five transitions is reached.

The various architectures are trained using the Adam optimizer [42] to minimize the mean square error between the predicted speeds and the ground truth values. Additionally, a regularization loss $\mathcal{L}_{\text{reg}}$ can added to the prediction loss $\mathcal{L}_{\text{pred}}$ to enforce sparsity in the messages. The total loss $\mathcal{L}$ is then the linear combination of these two losses and the weight regularization loss $\mathcal{L}_w$.

$$\mathcal{L} = \alpha_0 \mathcal{L}_{\text{pred}} + \alpha_1 \mathcal{L}_{\text{reg}} + \alpha_2 \mathcal{L}_w \tag{4.4}$$

$$\text{with} \begin{cases} \mathcal{L}_{\text{pred}} = \frac{1}{N} \sum\limits_{i \in \{1:N_v\}} |\mathbf{v}'_i - \hat{\mathbf{v}}'_i| \\ \mathcal{L}_{reg} = \delta_{\text{regu}} \frac{1}{N_e} \sum\limits_{k \in \{1:N_e\}} |\mathbf{e}'_k| \\ \mathcal{L}_w = \sum\limits_{l = \{1:N_w\}} |w_l|^2 \end{cases} \tag{4.5}$$

$$\text{and } \alpha_0 = 100, \quad \alpha_1 = 0.001, \quad \alpha_2 = 0.00001 \tag{4.6}$$

where δ_{regu} controls when the L1 regularization is applied, and N_e, N_{layers} and N_w respectively denotes the number of edges, the number of layers in the GNN and the number of weights in the architecture.

Moreover, boundary conditions are manually enforced in such a way that the model can solely focus on physics. It is important to emphasize that the decision not to model the boundary is crucial for eliminating absolute positions from the network's features.

In this work, the one-step learning method is applied across all networks. Training is conducted over 50 epochs using transitions from 200 simulations of 1000 timesteps. The different methods are also trained by optimizing k-step trajectories. Different results are available in the annex such as an ablation study. The training process utilizes batches of 32 transitions and a learning rate of 0.001 with an exponential decay relying on a $\gamma = 0.9$. Training losses are available in the annex.

Chapter 5

Experiments

In this chapter, the different models are assessed. To do so, 20 rollout trajectories of 150 steps are generated with the differents models and are compared with the ones obtained with the ground truth simulator.

The different models have been chosen following the ablation study realized in the annexe. Specifically, no dropout was selected. It was also observed that using the inductive bias that forces opposite messages to be opposite was clearly useful. Therefore, this inductive bias will not be used here.

More specifically, it is observed in the ablation study in the annex that the results for the best models trained using one-step transitions are close to the ones corresponding to rollout training. In the following, only models obtained by rollout training are used. Also, no dropout is employed and EPD-Gat and EPD-GNN respectively have three and four GNN layers.

Analyzis

Examples of rollout simulations are available online[1]. It can be observed that the simulations get very different from the ground truth ones after a few iterations. Such a behaviour is expected and quantitative experiments are necessary to assess the quality of the simulation trajectories.

It is then necessary to compute quantitative measurements of the neural network performances to assess its quality. First, the quality of 1-step transitions is assessed. The results show that the methods are very accurate concerning one-step transitions. However, neural network simulators always produce errors that scale exponentially with time. That is why, it is also important to assess the quality of the rollout trajectories $\tau_{\mathcal{G}}$ with different statistics.

Important results and their standard deviations are available in Table 5.1. More precisely, this table contains the mean absolute error. This loss was further decomposed into norm and angle errors. The table also contains MAE averaged. While previous metrics dealt with one-step transitions, this one is more oriented with assessing trajectories. It

[1] To see the different rollout simulations, please refer to this URL https://youtube.com/playlist?list=PLzXmrzZW9EJ4S2wFZHaIhKHtkzi-CaUxG&si=KzlSAv-5vAaapLyc.

is obtained by computing the mean L1 errors between the ground truth trajectories and the simulated one. In this work, this metrics is computed by generating 15 steps.

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
Baseline	0.00718 ± 0.00649	0.305 ± 0.305	0.00796 ± 0.00831	0.134 ± 0.00479
IN	0.00216 ± 0.00242	0.0514 ± 0.0514	0.00254 ± 0.00299	0.0485 ± 0.00216
EPD-GNN	**0.000179 ± 0.00107**	**0.0338 ± 0.0338**	**0.0000563 ± 0.000960**	**0.0233 ± 0.00243**
EPD-GAT	0.000319 ± 0.00102	0.0528 ± 0.0528	0.000215 ± 0.000801	0.0280 ± 0.00214

Table 5.1: Evaluation of the different models for non-noisy data. It can be observed that EPD-GNN reaches the best performances compared to the other models.

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
Baseline	0.00665 ± 0.00673	0.253 ± 0.253	0.00668 ± 0.00862	0.0958 ± 0.00370
IN	0.00144 ± 0.00180	0.127 ± 0.127	0.00140 ± 0.00206	0.0361 ± 0.00262
EPD-GNN	0.000918 ± 0.00134	0.135 ± 0.135	**0.000369 ± 0.00127**	0.0320 ± 0.00266
EPD-GAT	**0.000848 ± 0.00115**	**0.112 ± 0.112**	0.000526 ± .000915	**0.0277 ± 0.00228**

Table 5.2: Evaluation of the different models for noisy data. It can be observed that EPD-GaT reaches the best performances compared to the other models.

Tables 5.1 and 5.2 show that smaller networks are outperformed by larger ones. More precisely, it is observed that the EDP-GNN has the best performance when dealing with non-noisy data and the EPD-GaT is the best for noisy datasets. It can also be observed that the baseline model performs significantly worse than the interaction network. This result will repeat itself with the other statistics: large models are better simulators than small ones.

It can also be observed that models trained on noisy datasets do not have significantly worse performances compared to the ones trained on non-noisy data. This may seem a bit counterintuitive. A plausible explanation is that, as shown in Figure 5.9 and Figure 5.10, the different models tend to focus on small speeds. This is possibly an effect of the initial conditions. Indeed, by forcing particles to be distributed following a grid in pairs, there is a large portion of the dataset that is characterized by cells having only a few numbers of interactions. This small number of interactions partly explains the small velocities in the dataset whose associated data distributions can be observed in Figures 5.3 - 5.4. Moreover, applying multiple rotations to the speed can make it stay closer to the initial positions which would then explain the better results with noisy data. Another element to this explanation lies in the choice of parameters of the simulations. Indeed, by selecting this set of parameters, the resulting noise is rather small with only rotations of a few degrees. Such a small amplitude of noise does not degrade the performances significantly.

The impact of the speed on the error can also be studied with Figures 5.1 - 5.2. These figures show that the neural network performs better when dealing with slow cells with relative MAEs smaller than 1%. This behaviour is expected considering that, as shown in Figures 5.3 and 5.4, there are significantly more data points in this range of values. However, it is still possible to observe a slight upwards tendency in the errors when the norm is smaller than 0.12.

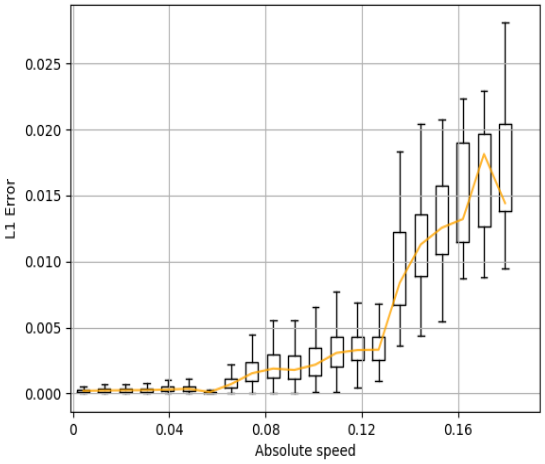


Figure 5.1: Evolution of the error with respect to the norm of the speed for the non-noisy data (Interaction Network).

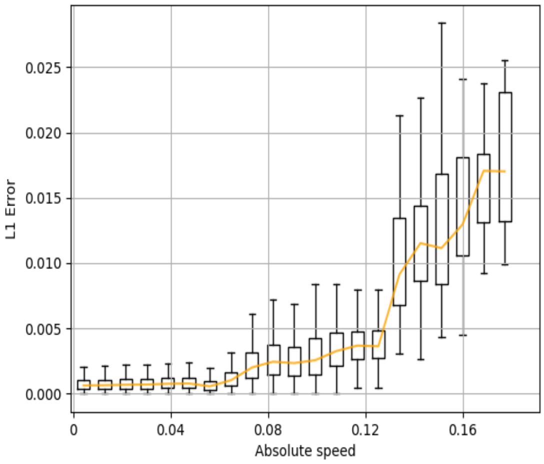


Figure 5.2: Evolution of the error with respect to the norm of the speed for the noisy data (Interaction Network).

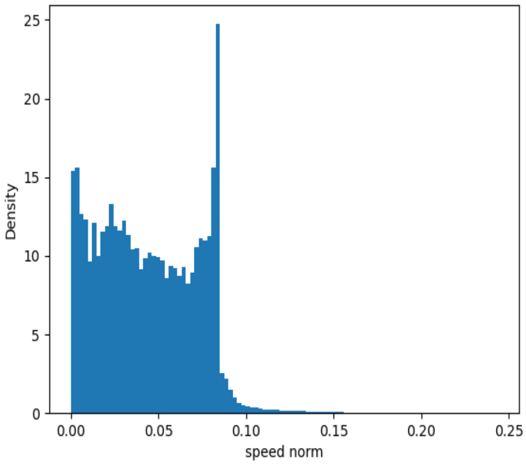


Figure 5.3: Speed norm distribution for non-noisy data.

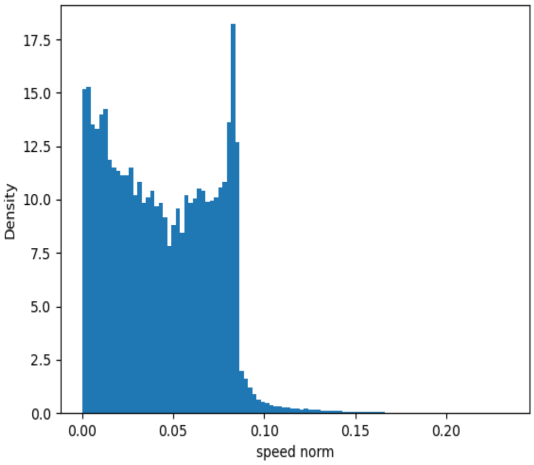


Figure 5.4: Speed norm distribution for noisy data.

While the behaviour of the neural networks with respect to the speed is independent of the architecture, the relation with the angles is a bit more complicated to analyse as it depends on the different architecture choices. Two examples are available below. While it is true that the qualitative behaviours are different, it is important to note that the values remain very small.

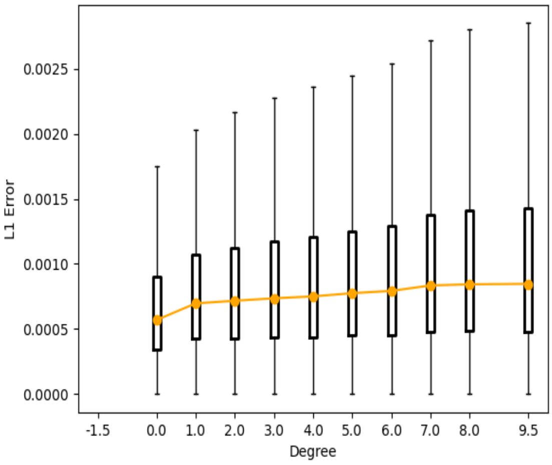


Figure 5.5: Evolution of the error with the degree of the cell. In this case (EPD-GaT), a slight increase can be observed.

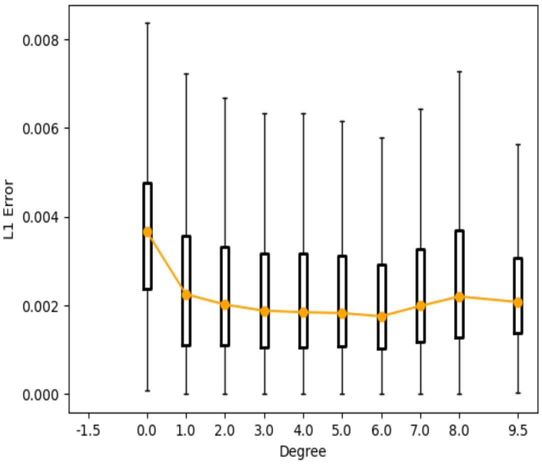


Figure 5.6: Evolution of the error with the degree of the cell. In this case (baseline), a decrease of the different values can be observed

MSE rollout

Various other statistics can be computed to further compare the models and the ground truth simulations. Figures 5.7 and 5.8 show the evolution of the mean square error between the rollout trajectories of the different models and the ground truth. The different results are aligned with the Tables 5.1 and 5.2. One interesting observation is to realize that the rollout MSE of the EPD-GNN increases faster than the one of the EPD-Gat which leads to the latter becoming better after around 80 steps for non-noisy data.

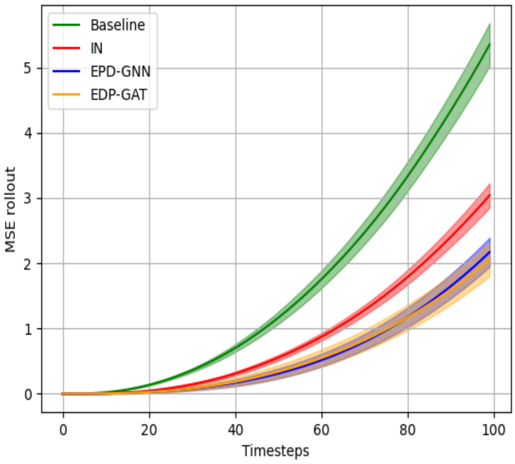


Figure 5.7: Rollout MSE when dealing with non-noisy data.

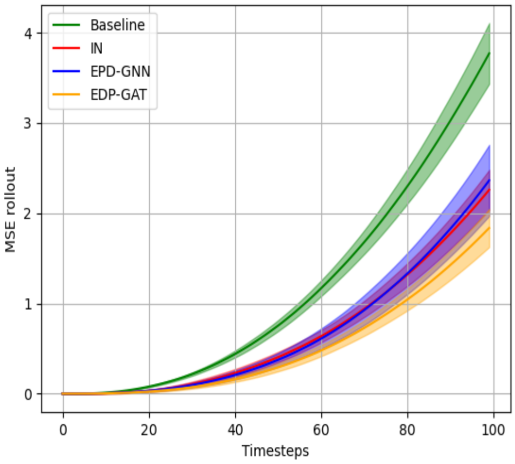


Figure 5.8: Rollout MSE when dealing with noisy data.

Velocities distribution

A simple way to assess if the simulations are close to the ground truth ones is to observe if the velocities are close to the ground truths. This can be done by computing histograms representing the relative speed s_{rel} of the different cells. In this context, the average speed is computed by applying the mean over all the timesteps of the speeds of a cell.

Denoting by S the number of simulations, T the number of timesteps within each of them and i the index given to each cell, the following summary statistics are introduced.

$$s_{rel} = \frac{|s|}{\langle s \rangle} \qquad\qquad \langle s \rangle = \sum_{t=1}^{T} |v^i(t)| \tag{5.1}$$

where $|.|$ denotes the norm of the speed vector.

It is then possible to plot the histograms of the magnitude s_{rel} and the projections of s_{rel} according to the x and y axes. Please refer to the annexe to observe the histograms of the projections.

These results show that the models tend to underestimate the speeds as there is a spike at $s_{rel} = -2$ and not a lot of values beyond two. It can also be observed that the interaction and the EPD-GaT are the best models here.

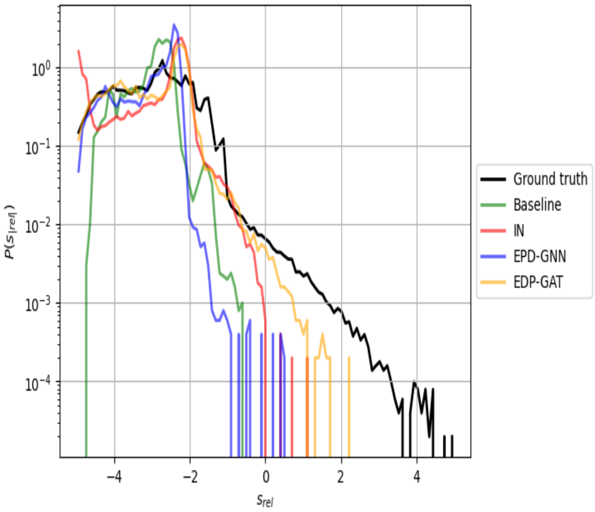


Figure 5.9: Speed norm distribution for non-noisy data.

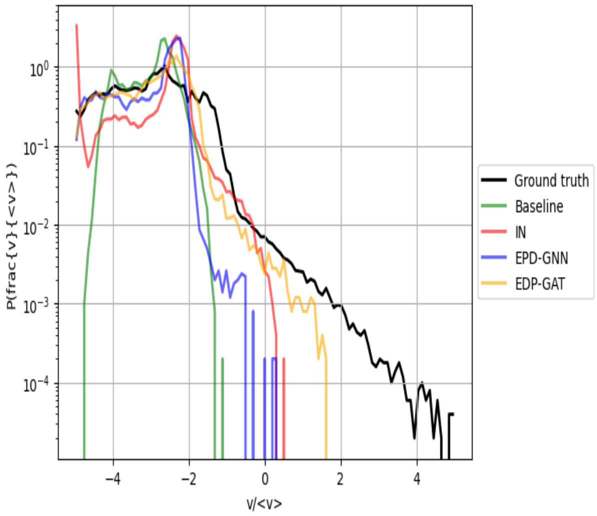


Figure 5.10: Speed norm histogram for noisy data.

Mean square displacement

Analyzing dynamics is often performed by computing the different moments. Moments provide a measure of how far particles have moved from their initial position after τ timesteps. The second moment which is referred to by mean square displacement (MSD) is commonly used in the literature [43]. It helps describe the kind of diffusion that is experienced. In normal diffusion, the MSD increases linearly with time, while in subdiffusive or superdiffusive processes, it respectively increases slower or faster.

$$MSD(\tau) = \frac{1}{S}\frac{1}{N}\frac{1}{T-\tau}\sum_{s=1}^{S}\sum_{t=1}^{T-\tau}\sum_{i=1}^{N}\left|\mathbf{x}_{t+\tau}^{(s,i)} - \mathbf{x}_{t}^{(s,i)}\right|^2 \tag{5.2}$$

Figure 5.11 shows the MSD for the different models. In this case, angle corrections are small enough not to affect the MSD, resulting in line graphs for all the models. Since it is complicated to assess the difference between the different models, it is necessary to compute the distance between the curves and the ground truth one. This is realized by computing the MAE between them. Nevertheless, it has to be highlighted that it is not always good practice to compute standard distance metrics when dealing with time series data as they only assess the difference between values and do not consider the trends. In this case, it is acceptable since the trends are qualitatively considered to be similar. This technique is repeated with the other statistics.

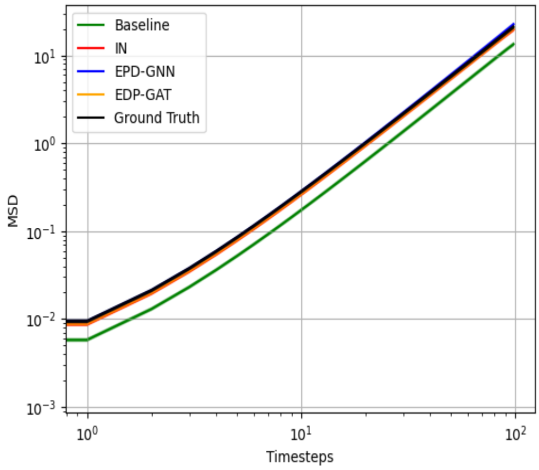


Figure 5.11: MSD for different trajectories generated by the different models for non-noisy data. Most statistics are close to the ground truth.

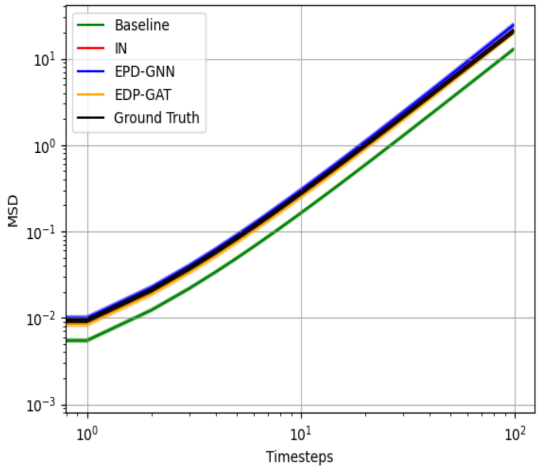


Figure 5.12: MSD for different trajectories generated by the different models for non-noisy data. Similar results are observed compared to non-noisy measurements as noise does not have such an impact.

Model	MSD (non-noisy)	MSD (noisy)
Baseline	12.28 ± 16.11	15.11 ± 19.37
IN	0.40 ± 0.57	0.00017 ± 0.00015
EPD-GNN	0.39 ± 0.61	1.64 ± 2.45
EPD-GAT	0.24 ± 0.31	0.25 ± 0.25

Table 5.3: MSD measurements of the different models (MAE from the ground truths). It can be observed that EPD-Gat has the best performances

Self-intermediate scattering function

The next statistic is the self-intermediate scattering function along the x dimension. The intermediate scattering function is defined as the Fourier transform of the Van Hove function and describes the interference between pairs of particles. The self-intermediate scattering function describes how the position of a cell at a timestep t correlates with its future position at timestep $t+\tau$. On top of particle simulations, this statistic is widely used to have a better understanding of liquid or glassy materials [44] [45].

$$SISF(\mathbf{q},\tau) = \frac{1}{S}\frac{1}{N}\frac{1}{T-\tau}\sum_{s=1}^{S}\sum_{t=1}^{T-\tau}\sum_{i=1}^{N}\exp\left[i\mathbf{q}\cdot\left(\mathbf{x}_{t+\tau}^{(s,i)} - \mathbf{x}_{t}^{(s,i)}\right)\right] \tag{5.3}$$

$$\text{with } \mathbf{q} = \frac{2\pi}{R}\cdot\begin{bmatrix}1\\0\end{bmatrix} \tag{5.4}$$

In the different results that are obtained, an oscillatory behaviour can be observed. This is partially explained by the initial conditions of the simulations that place the different cells in a grid-like manner. It can be observed from Figure 5.13 and 5.14, that the EPD-GAT and the EPD-GNN are closer to the ground truth which is another indicator that these are the best models.

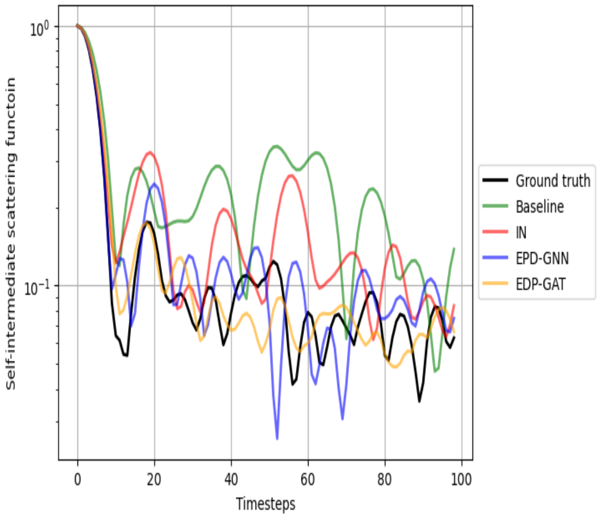


Figure 5.13: Intermediate self scattering function for non-noisy dataset.

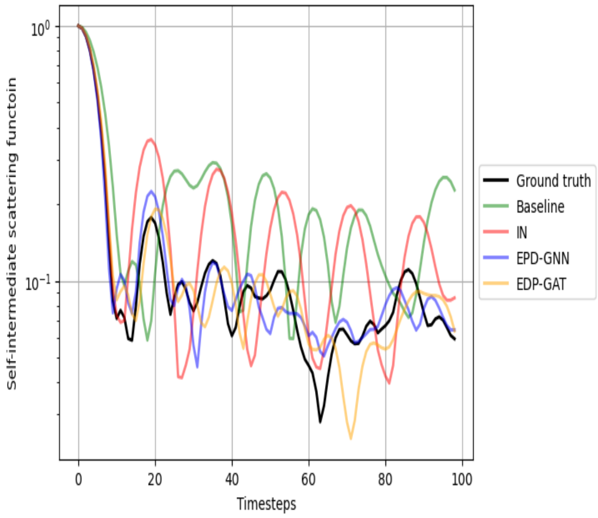


Figure 5.14: Intermediate self scattering function for noisy dataset.

Model	SISF (normal)	SISF (noisy)
Baseline	0.019 ± 0.020	0.0162 ± 0.0166
IN	0.0063 ± 0.0099	0.00795 ± 0.0105
EPD-GNN	0.0015 ± 0.0023	0.000915 ± 0.00141
EPD-GAT	0.00074 ± 0.0012	0.000902 ± 0.000981

Table 5.4: l1 Performance Metrics for Different Models (MAE from the ground truths). It can be observed that the EPD-Gat model is the best according to this statistics and this metric

Radial Distribution Function

Another interesting statistic is the radial distribution function RDF(x). This statistic describes the density of cells with respect to the distance from a reference one. More precisely, it is a histogram whose values are proportional to the number of cells that can be expected to be found at a distance r from another cell.

Figures 5.15 and 5.16 show that it is more likely to encounter a cell around a distance of 2 which makes sense as this is the position where the interaction is equivalent to zero.

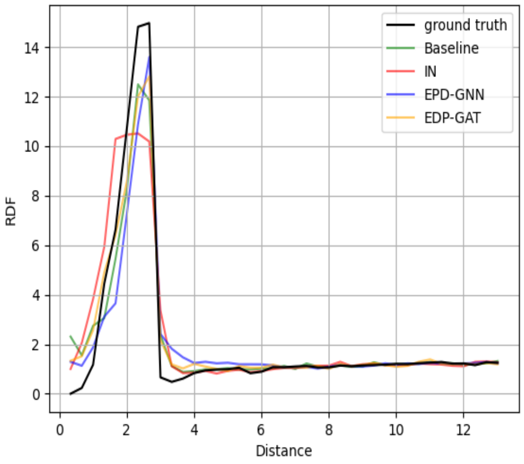


Figure 5.15: Radial distribution function of the simulations with respect to the ground truth one for non-noisy simulations.

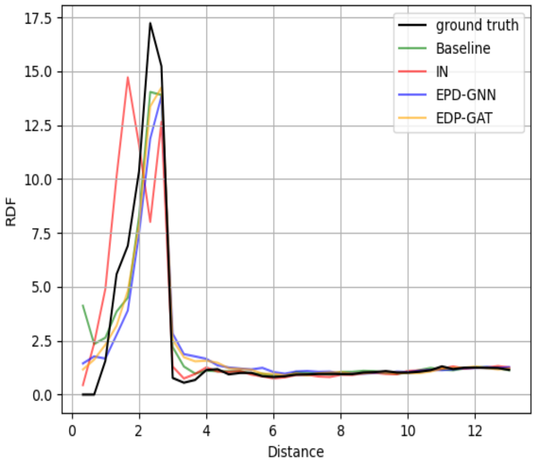


Figure 5.16: Radial distribution function of the simulations with respect to the ground truth one for noisy simulations.

Model	**RDF (normal)**	**RDF (noisy)**
Baseline	7.26 ± 22.2	7.97 ± 24.8
IN	8.30 ± 19.8	10.5 ± 25.4
EPD-GNN	6.81 ± 23.4	6.62 ± 21.5
EDP-GAT	7.94 ± 23.5	7.09 ± 22.0

Table 5.5: L1 Error Metrics for Different Models (MAE from the ground truths). EPD GNN is the best according to the metric and this statistics.

Part II

Model recovery using symbolic regression

Chapter 6

Symbolic regression

This chapter introduces different notions on symbolic regression. The different kinds of symbolic regression methods are briefly introduced. Since this work utilizes genetic algorithms, specifically PySR, to perform symbolic regression, a more detailed explanation of this algorithm is provided subsequently.

6.1 Introduction

In recent decades, the performances of neural networks have steadily improved with their size, reaching state-of-the-art results on a wide range of problems and even surpassing humans in certain tasks [46] [47]. Despite these impressive achievements, neural networks are not perfect. One significant drawback is that they are black-box models which means that it is complicated to understand how they relate inputs to outputs. This lack of interpretability is particularly concerning in fields where understanding the decision-making process is crucial. For example, in high-stakes areas such as medicine, interpretability is essential since errors can have severe consequences. As a result, neural networks that do not provide performance guarantees are unlikely to be adopted, regardless of their accuracy. Various techniques, such as saliency maps [48], have been developed to mitigate the black box issue.

This work demonstrates that symbolic regression can also be used to tackle this issue. Symbolic regression is a machine learning task that focuses on deriving symbolic mathematical expressions from data. The core idea is to leverage the performances of neural networks in high-dimensional spaces while extracting simple equations that best explain the behaviour of the network. Beyond enhancing interpretability, obtaining symbolic models has the potential to improve generalization performance [49] and could also uncover new insights that enable humans to develop a deeper understanding of complex phenomena.

In this work, symbolic regression is employed for model discovery. More precisely, symbolic regression is applied to the messages of the interaction network with l1 regularisation to extract the algebraic equations of the interactions.

6.2 Symbolic regression methods

Model rediscovery of physical systems using symbolic regression is a current task that was introduced in the 1970s [50]. Since then, symbolic regression has reached other fields and various algorithms have been introduced. It is possible to divide them into four categories: regression-based, expression-tree-based, physics-inspired and mathematics-inspired methods.

First, regression-based methods focus on optimizing a set of parameters that weight different predefined functions [51]. Expression-tree are very different from this first category. This kind of method relies on a tree structure to represent the equations. As illustrated in Figure 6.1, the nodes of the tree are either operators, variables or constants. These methods then make use of genetic algorithms [52], reinforcement learning [53] or other methods to explore the space of possible expression trees. Finally, physics-inspired and mathematics-inspired methods make use of specific physical or mathematical notions to simplify the symbolic regression issue [54]. A notable example of a physics-inspired method is AI-Feynman [55] [56] which utilizes concepts such as symmetry, dimensional analysis or Occam's razor to explore the hypothesis space.

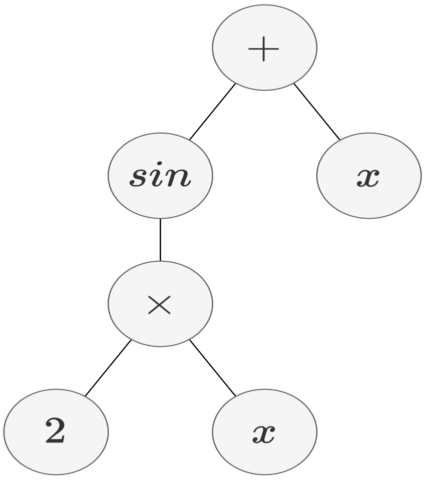


Figure 6.1: Example of expression tree "$x + sin(2 \times x)$". In this expression, "$sin()$" is an unary operator and $\{\times, +\}$ are binary operators. The complexity of this equation is 6 because 6 nodes are in the expression tree.

6.3 Genetic algorithms and PySr

Genetic algorithms have demonstrated strong performance compared to other methods [57] and are therefore employed in this part of the work. More precisely, the open-source and user-friendly package PySr [52] is utilized.

PySr employs an evolutionary-inspired tournament selection algorithm to perform symbolic regression. Different populations of expression trees evolve simultaneously. At each iteration, the best individuals are selected by computing the accuracy of the different equations in the population. Additionally, the algorithm aims to find the best equations at each level of complexity, where complexity is defined as the number of nodes in the expression tree. Moreover, the selection process is probabilistic. The best individuals are selected according to some probability p. If they are not selected, they are discarded and the process starts again with the remaining of the population. On top of these classic steps, PySr relies on simulated annealing to speed up the process.

Once the fittest individual is selected, a modified copy replaces the eldest individual in the population. These modifications are based on three processes that can be performed simultaneously: mutation, crossbreeding and migration. Mutation consists in replacing a node with another one. Crossbreeding swaps an entire subtree of one expression tree with a subtree from another equation. Finally, migration involves transferring the best equation from one population to another.

Moreover, PySr apply other improvements to the typical genetic tournament selection. PySr is based on an Evolve-Simplify-optimize loop. Once multiple iterations of evolution are performed, the algorithm simplifies the different equations by using a set of algebraic equivalences. It also tries to further optimize the constant by using multiple iterations of BFGS [58].

However, it is important to acknowledge that genetic algorithms like PySR are not without limitations. While they generally perform well in practice, the results can be difficult to reproduce. Indeed, these algorithms struggle when dealing with very large hypothesis spaces and are highly sensitive to randomness or hyperparameters [59].

PySR has been applied across various domains, including astrophysics [60], particle physics [61], and climate models [62]. Research using PySr to interpret neural networks often builds on the work of Cranmer et al [35]. In the paper, the authors decided to apply l1 regularisation to the messages. Then, they only keep the most important message features and discover that the messages are linear transformations of forces. This thesis is also greatly inspired by this paper. Some follow-up work has been realized. For instance, PySr was also used to recover the equation of gravity and the masses of the celestial bodies in a 31-body version of the solar system [36].

Chapter 7

Experiments

This chapter contains the experiments conducted related to symbolic regression. First, the different messages are interpreted. Next, symbolic regression is applied to derive the equations of the interactions. Finally, an extension of the current technique based on general additive models is proposed to obtain more easily interpretable models.

For these experiments, data is generated by simulating 200 randomly initialized rollout simulations of 30 steps with a GNS. This limited number of steps is chosen to ensure that the behaviour of the neural network remains consistent with the data it was trained on. Specifically, the interaction network implementation that does not use dropout and the one-step training procedure are used in this section. The different models are obtained by taking the ones that perform best in the 1-step transition after 20 epochs.

7.1 Interpreting messages

In previous sections, the Interaction Network was introduced with an l1 regularization on the messages. This regularization allows them to be sparse. It means that only a small part of the features of the messages should bear most of the information. As illustrated in Figure 7.1, it is possible to compute the standard deviation of the different features with respect to different inputs to uncover which ones are the most important. It is current to only have 2 features that change significantly more than the others. This result goes in the same direction as previous work [34] [35] that argued that the most optimal dimension to transmit the information is the same as the number of spatial dimensions. However, it is not clear in practice that this is always the case. Indeed, many counter-examples were found when training different neural networks. When related papers [35] tried to apply this method to dark matter, a counter-example occurred with a single feature that was significantly different from the others. It remains complicated to predict the number of elements that will bear most of the information in the messages of GNNs.

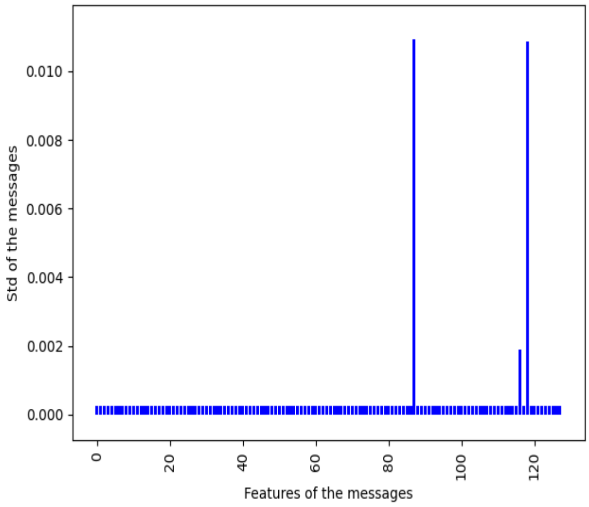


Figure 7.1: Standard deviation of the features in the message. It can be observed that 3 elements change significantly when encountering different inputs but that only 2 of them are significantly larger than the rest. As assumed in Cramner et al [35], it is current to reach a size that is the same as the spatial dimensionality.

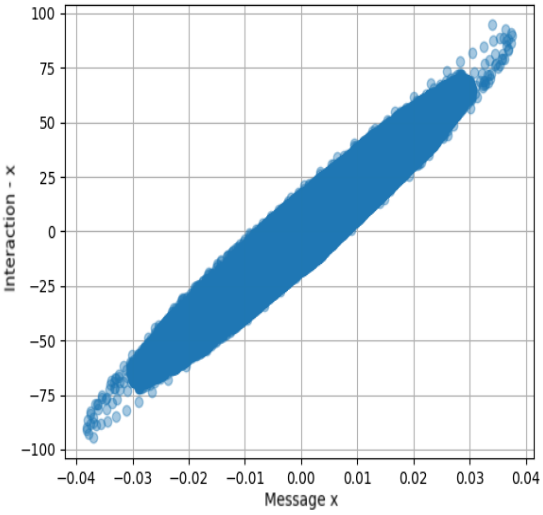


Figure 7.2: Scatter plot of the ground truth interaction and first message feature. It can be observed that the message features are linear transformations of ground truth interactions.

Various observations can be realized by observing the evolution of the messages with respect to the different inputs. Figure 7.3 shows that the different messages are very close to the ground truth interaction forces. Previous works [35] showed that the messages are linear transformations of the forces when dealing with systems that respect the second Newton's law. Moreover, this paper deals with a simple gravitational system where no active force components appear, which means that only their edge model ϕ^e contains semantic information. This seems to indicate that their task is simpler than the problem tackled here.

However, it is possible to show that the messages m_{ij} are still linear transformations of the ground truth in this case. The related mathematical demonstrations are available in the appendix. This is also empirically verified by obtaining the scatter plots of the main messages with respect to the ground truth interactions (Figure 7.2) and by computing the R^2 score of the corresponding linear regression, as realized in Table 7.1.

Another interesting observation from Figures 7.3 and 7.4 is that message features with a smaller standard deviation than the two larger ones still embody interesting information about the messages. For instance, it is possible to obtain other rotations of the forces or even the magnitude of the force. However, among the different training performed for this section, these kinds of messages would not always appear. Trying to use these additional message features or understanding the reasons behind the apparition of the interaction norm in the messages could be some further work.

Moreover, the obtained messages are compared with the ones from the baseline model. As expressed in Table 7.1, the baseline model has worse performances. This result is possibly linked to the winning lottery ticket hypothesis [63]. Having larger output spaces allows for more possible well-initialized weights that ease training. This might also be part of the explanation why dropout leads to worse results for this model. It can also be observed in Table 7.1 that using smaller l1 regularization leads to better messages.

Figure 7.4 shows that networks that are trained on the noisy dataset are still able to uncover linear transformation of the different messages. On the other hand, as soon as dropout is introduced in the model, a completely different behaviour is observed. As Figure B.9 indicates, the standard deviation of the message features gets very low when the distance between cells is fixed in that case. While the equations of the different message features could be obtained, the physical interpretation of these is still complicated. A possible explanation is that using dropout with l1 regularization leads to instabilities during training and degrades the results.

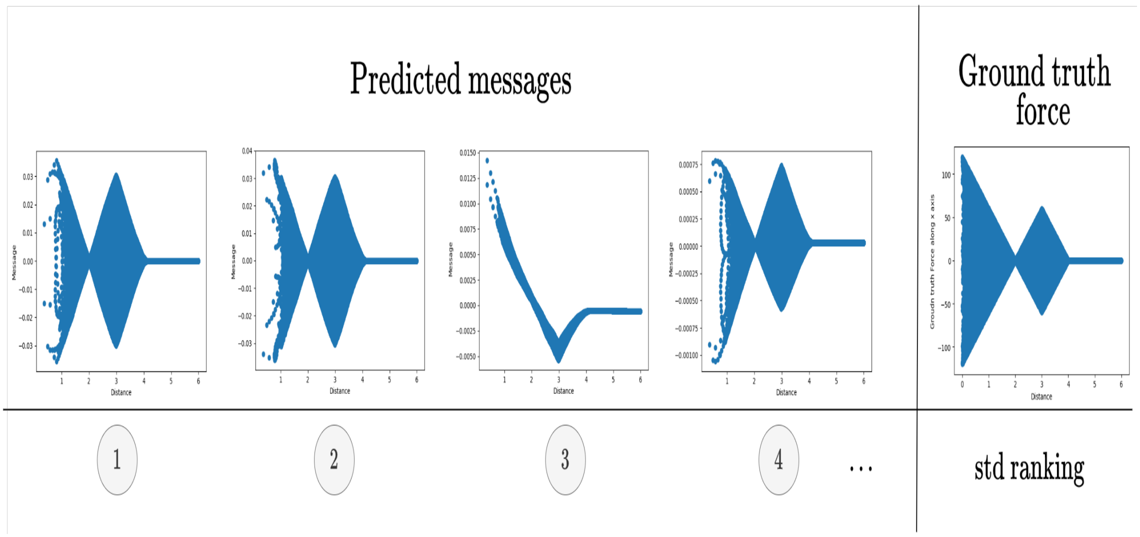


Figure 7.3: Impact of the distance on the most relevant features of the messages of a network trained on non-noisy data. It can be observed that some message feature with smaller standard deviations still yields important information. Some are close to rotations and others are similar to the norm of the interaction.

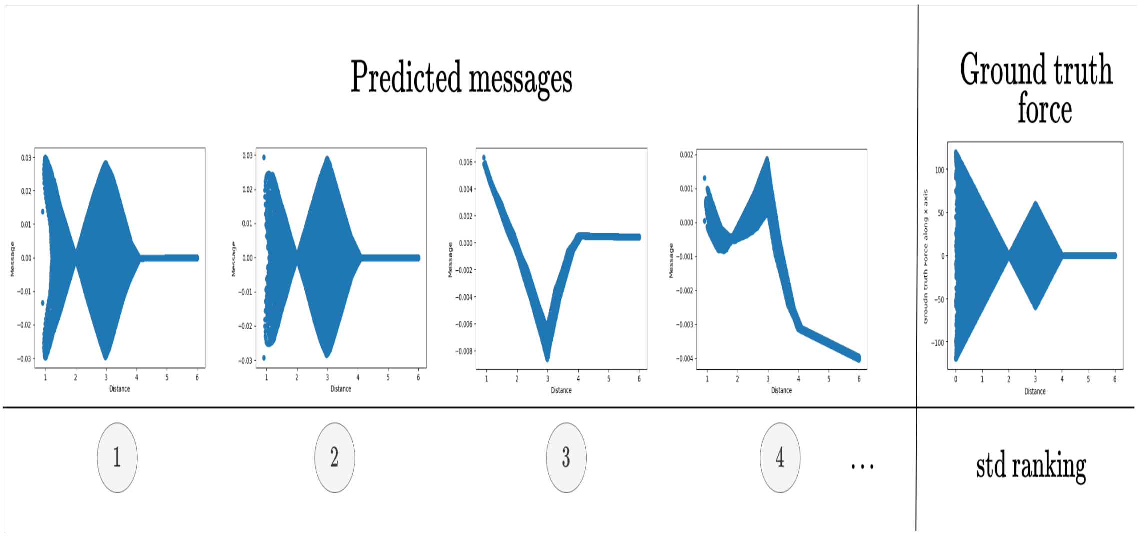


Figure 7.4: impact of the distance on the most relevant features of the messages of a network trained on noisy data. It can be observed that some message feature with smaller standard deviations still yields important information. Some are close to rotations and others are similar to the norm of the interaction. This example shows that it is also possible to encounter messages that are harder to directly interpret as the fourth message feature.

Model	**R2 score (non noisy)**	**R2 score (noisy)**
IN (l1 = 1)	0.781	0.972
IN (l1 = 0.01)	0.999	0.988
IN (l1 = 0.001)	0.999	0.995
IN (l1 = 0.0001)	0.999	0.999
Baseline	0.103	0.09651

Table 7.1: Table of R^2 score for different architecture. These results seem to indicate that l1 values have a small influence on the R^2 score. As soon as the values are not too large, the R^2 scores are similar. Compared to these results, the R^2 scores of the baseline model are significantly smaller than the other ones.

7.2 Symbolic regression on messages

PySr is now used to fit equations to the two most significant features of the messages. Genetic algorithm struggles when the hypothesis space of expression trees is too large. That is why different parameters such as the number of possible operators is kept as small as possible. Another parameter that should remain small is the complexity which is the number of nodes within the expression tree of an equation (cfr. Figure 6.1). Durign the different runs, the variables corresponding to $cos(\theta)$ and $sin(\theta)$ are associated with a complexity of 3 compared to other variables that have a complexity of 1. This is done because the cosine and sine are complex operators.

The algorithm is applied on the same edges $\{r\,,\,\cos(\theta)\,,\,\sin(\theta)\,,\,R_1\,,\,R_2\}$ as in the first part of this master thesis. It is important to note that the inputs that are chosen for symbolic regression are important as they will impact the complexity of the equation. For instance, taking the two components of the vector between the cells instead of the current configuration would have significantly increased the complexity of this task. Regarding its outputs, PySr tries to yield equations that are accurate while remaining as simple as possible. To do so, the final equation is selected by maximizing the *score* which is the drop of the loss, which is the mean square error (MSE) here, per increase in complexity. This idea of simplicity and accuracy once again is in echo with Occam's razor.

$$score = \frac{|\delta\ \log(MSE)|}{\delta\ complexity} \tag{7.1}$$

Unfortunately, since the interaction force between cells is a piecewise linear function, the total complexity is large enough to prevent retrieving the exact solution. Equations that are obtained in such a case can be observed below[1]. While the method found that the interactions were indeed piecewise linear functions, the results are far from the ground truths. It has to be noted that it is already possible to find some of the limits of the different segments of forces with this method.

$$\begin{aligned} m_0 =& \sin(\theta) \times 1.4444948 \times (0.31420892 + 0.9930598) \\ & \times \text{Piecewise}\begin{pmatrix} 0.0104460325, & \text{if } -0.3825599r - 0.3525599 + 1.5175834 > 0 \\ 0.0, & \text{otherwise} \end{pmatrix} \\ & \times (r + r - 2.1429582 - 2.1429582 + 0.31420892) \times 0.7894803 \end{aligned}$$

$$\begin{aligned} m_1 =& 0.011282312 \times 1.8461835 \times \cos(\theta) \\ & \times \text{Piecewise}\begin{pmatrix} r - 1.2624552 - 0.6207398, & \text{if } 0.6907398r - 2.1437254 < 0 \\ 0.0, & \text{otherwise} \end{pmatrix} \end{aligned}$$

To tackle this issue, it is possible to simplify the task and apply symbolic regression on each segment of the forces. Indeed, by inspecting the different messages in Figure 7.3,

[1] This equation is obtained by performing 10 000 steps with a maximal complexity of 42.

the different segments can be identified. The idea is then to apply symbolic regression on the parts where the behaviour of the envelop significantly changes. Hence, the limits are $\{r = 2, r = 3, r = 4\}$. It has to be noted that the point $r = 2$ results from the change of sign in the norm of the forces. Observing the higher-order messages could give some insights on the fact that 2 should not be used to define segments. To make sure that no data points outside the expected segment are considered, some security margin $\kappa = 0.1$ is used to further restrict the domains. For each symbolic regression, the maximal complexity is fixed to 12.

The different results are available in the annexe and it was indeed found that each segment is a linear function ($f \approx a \cdot (r+k) \cdot cos(\theta)$ or $f \approx b \cdot (r+c) \cdot sin(\theta)$ where $\{a, b, c, k\}$ are constants) which is indeed a linear transformation of the ground truth. It can also be observed that coefficients for the segment $[0, 2]$ and $[2, 3]$ are very similar which indicates that this is the same underlying equation.

$$y_1 = \cos(\theta) \cdot (0.0307 \cdot r - 0.0613) \qquad r \in [0, 2] \tag{7.2}$$

$$y_1 = \cos(\theta) \cdot (0.0305 \cdot r - 0.0606) \qquad r \in [2, 3] \tag{7.3}$$

$$y_1 = 0.0133 \cdot (\cos(\theta) + 1.13 \cdot 0.00421) \qquad r \in [3, 4] \tag{7.4}$$

$$y_1 = 0 \qquad \text{otherwise} \tag{7.5}$$

7.3 General additive models and symbolic regression

As previously discussed, applying symbolic regression to derive interactions with highly complex equations poses significant challenges. To address this issue, the following section introduces an approach aimed at mitigating these difficulties. This remains a proof of concept and further work on this method is necessary.

The core idea consists in decomposing the output of the edge model ϕ^e into a weighted combination of basis functions in a similar manner to general additive models (GAM)[64]. However, in this method, the weights are generated by a neural network. The goal is to produce simple weights that are independent of most inputs, facilitating easier analysis through symbolic regression compared to directly analyzing the overall interaction.

This method can be motivated using physical arguments. In physics, it is common to encounter specific types of functions. For example, quadratic potentials are more prevalent than cubic ones. Consequently, it might be feasible to model complex interactions by combining a limited set of functions.

Several works have share similar considerations. For instance, SinDy [65] and linear symbolic regression [51] also try to sum different basis functions. The difference between this method and such techniques is that in this case, weights are outputs of a neural network instead of simple coefficients and that there is no ground truth known beforehand.

As illustrated in Figure 7.5, the edge model ϕ^e within the GNN incorporates an MLP that computes the weights of the basis functions. These weights are then applied to linearly combine the basis functions, ultimately producing the messages passed within the GNN. To increase the expressiveness of the neural network, it is beneficial to generate high-dimensional messages. Therefore, instead of computing a single weight for each of the K basis functions, D will be computed. In this project, the polynomial basis with a degree up to 2 is used.

Moreover, it is crucial to apply L1 regularization to both the messages $m_{i,j}$ and the weights w during training. As shown in previous sections, this regularization technique encourages sparsity and allows only a handful of messages and weights will bear most of the information as in the previous sections and will limit the number of equations that has to be found.

Let $w \in \mathbb{R}^{D\times K}$ denote the weights produced by the MLP$_\theta$ with parameters θ. The edge model of this architecture can be expressed as follows:

$$w = MLP_\theta(e_{ij}) \tag{7.6}$$

$$m_{i,j} = \sum_{k=1}^{K} w_k f_k \tag{7.7}$$

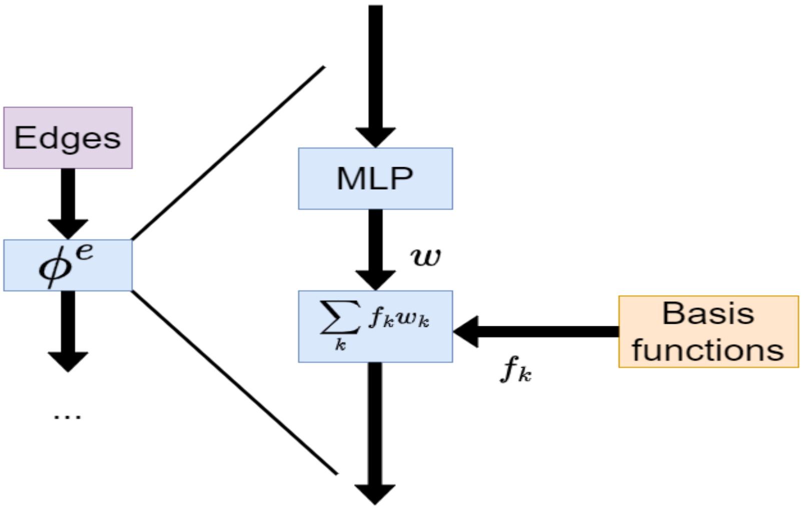


Figure 7.5: Illustration of the edge model for GAM symbolic regression. The edge model ϕ^e relies on an MLP to generate the weights that scale the basis functions as expressed in equation 7.7.

Application

This method is now applied to the same scenario as before. The first step involves assessing the impact of the different messages by calculating their standard deviations. For this purpose, a modified version of the Interaction Network with messages of dimension 128 and incorporating the action-reaction inductive bias ($m_{ij} = -m_{ji}$) is used. The l1 regularization of the weights is scaled with a constant factor of 10^{-4}. Non-noisy data is used as the default.

It can be seen in Figure 7.6 that two message features have standard deviations that are significantly larger than the others. The feature corresponding to the largest standard deviation is studied below.

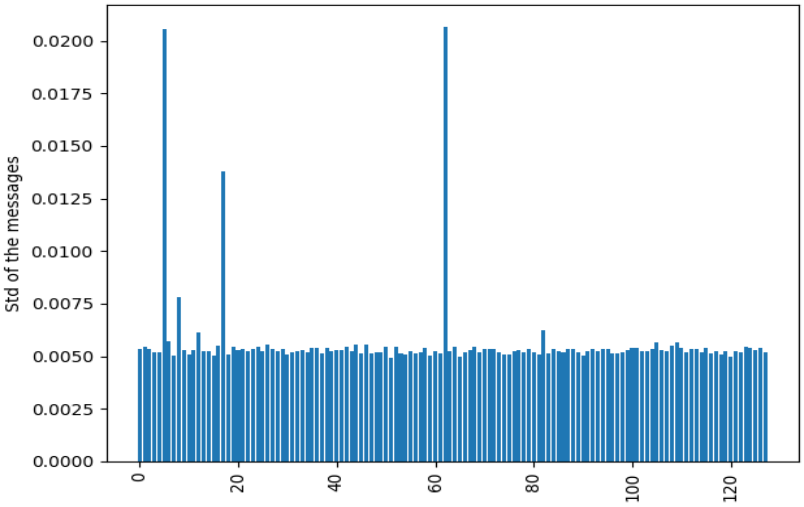
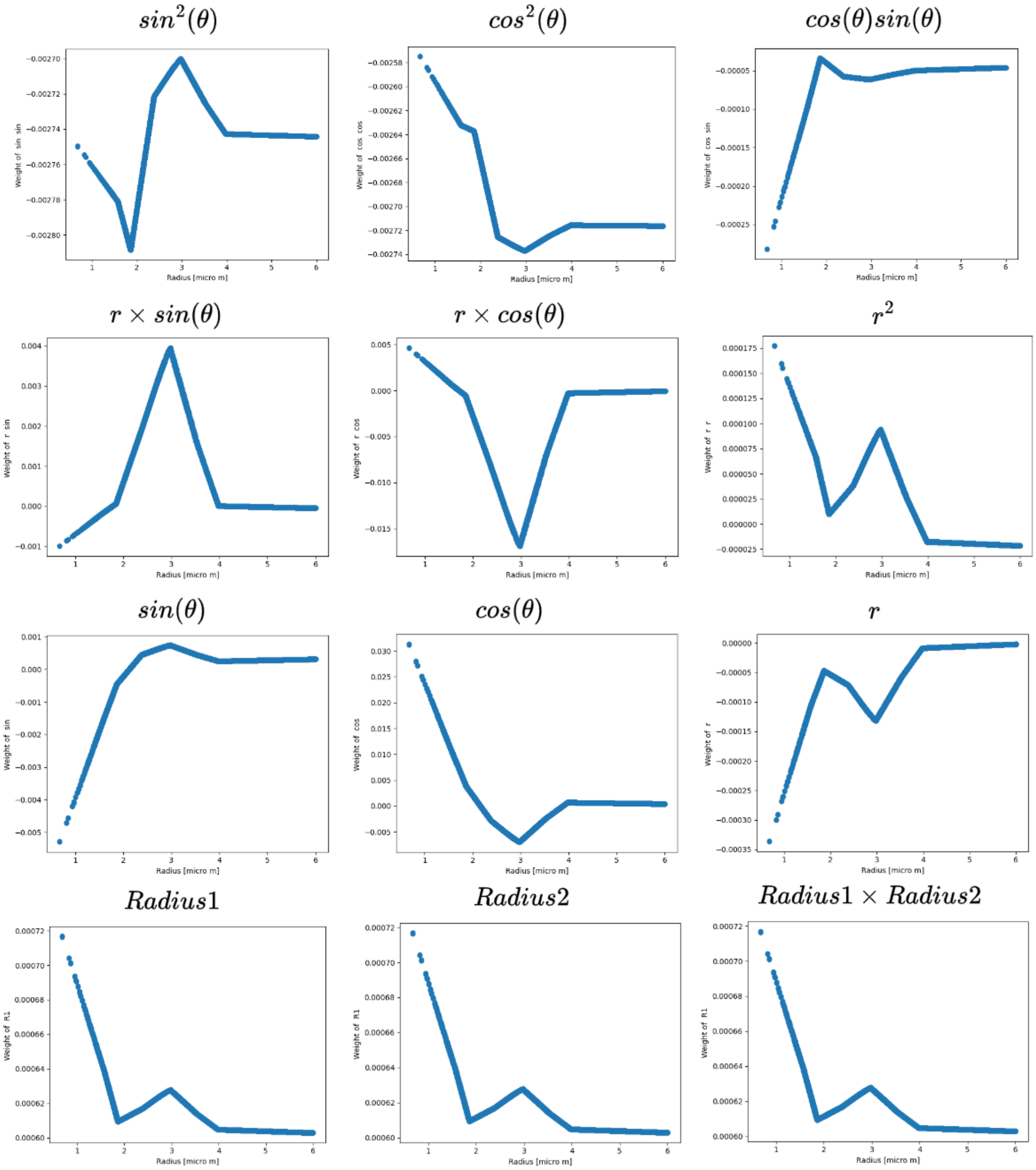


Figure 7.6: Standard deviation of the messages of the GAM with 128 message features. It can be observed that 3 features have standard deviations significantly larger than the other ones.

Figure 7.7 displays the relation between the different weights of the message with the highest standard deviation and the distance. Several key observations can be made from this figure.

Firstly, these plots suggest that the variance of the weight values is small when the distance between cells is fixed. This indicates that it is acceptable to assume that weights only depend on the distance.



$sin^2(\theta)$ $cos^2(\theta)$ $cos(\theta)sin(\theta)$

$r \times sin(\theta)$ $r \times cos(\theta)$ r^2

$sin(\theta)$ $cos(\theta)$ r

$Radius1$ $Radius2$ $Radius1 \times Radius2$

Figure 7.7: Evolution of the weights of the neural network with respect to the distance. Examples of other messages are available in the annex.

Next, as anticipated, the weights are simpler to analyze than the interactions from Figure 7.3. This can significantly ease symbolic regression. In this scenario, many of these weights can even be approximated using piecewise linear functions. This is reassuring as this is similar to the ground truth. It is also interesting to note that, as in the previous section, the neural network successfully learned the boundaries between segments.

It is important to recall that $Radius1 = Radius2 = 1$ in all the simulations. Interestingly, the network maintains symmetry with these inputs, as the related graphs are qualitatively similar and share comparable orders of magnitude. However, this symmetry introduces a challenge in the analysis, as it allows for the combination of terms in equation 7.7. For instance, the weights of $cos(\theta)$, $sin(\theta)$ and r can be scaled by a factor of three.

Furthermore, it can also be noted that the weights corresponding to $r \times cos(\theta)$ and $cos(\theta)$ are respectively very similar to those of $r \times sin(\theta)$ and $sin(\theta)$. Given that these are angular terms, this behaviour is expected for a model of linearly transformed interactions.

To conduct a more thorough analysis of the model, it is essential to identify the most important terms from the equation 7.7. This can be done by computing the standard deviation and the median of the weights. However, computing the median of the weights does not give a correct indication of the magnitude of each term as the different basis functions have significantly different ranges. 7.7. This is why, the standard deviation and the median of the absolute values of $w_k \times f_k$ are also computated. The results can be observed in Figures 7.8 - 7.11.

The analysis reveals that the two weights with the highest standard deviations are the $cos(\theta)$ and $r \times cos(\theta)$. This is interesting as the weight of $cos(\theta)$ is very similar to the graph of the norm of the interaction, which aligns with expectations. However, the high standard of the weight for $r \times cos(\theta)$ is more complicated to explain. It is likely to disappear with another training configuration. Nonetheless, the corresponding function $w \times cos(\theta) \times r$ exhibits a shape similar to the interaction (see annexe, Figure B.13).

At first glance, Figures 7.8 and 7.9 might raise concerns due to the high medians of the weights for $sin(\theta) \times sin(\theta)$ and $cos(\theta) \times cos(\theta)$. However, it is important to note that these two medians are very similar, and their standard deviations are quite small. Therefore, it seems reasonable to conclude that these two terms could combine to form a constant term.

By considering everything that was explained above and retaining only the significant weights (i.e. $cos(\theta)$ and $r \times cos(\theta)$), it is possible to obtain the following approximation of this message m_0:

$$m_0 \approx \begin{cases} \gamma_{1,k} \times r^2 \times cos(\theta) + \gamma_{2,k} \times r \times cos(\theta) + \gamma_{3,k} & \text{on each segment k} \\ 0 & \text{if } r \geq 4 \end{cases} \tag{7.8}$$

where the different γ are constants.

This is not exactly the solution, as there should not be any term in $r^2 \times cos(\theta)$ in the sum. Although the method did not fully capture the exact interaction force, it successfully uncovered several key insights related to the ground truth formula, which is encouraging for potential further refinement.

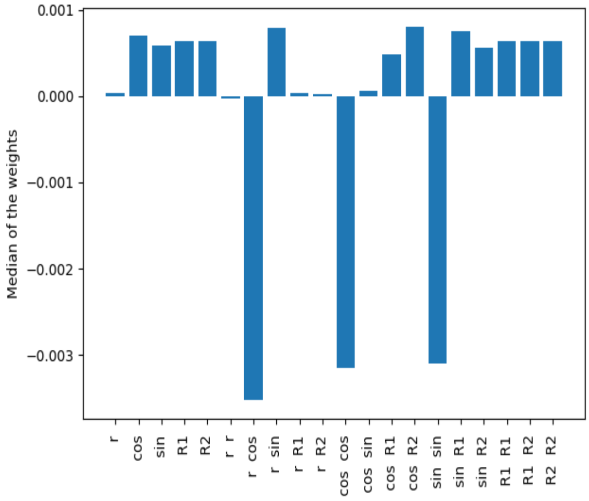


Figure 7.8: Median values of the weights of the message with highest standard deviation. The largest values are observed for the weights of $r \times cos(\theta)$ and $cos^2(\theta)$, $sin^2(\theta)$.

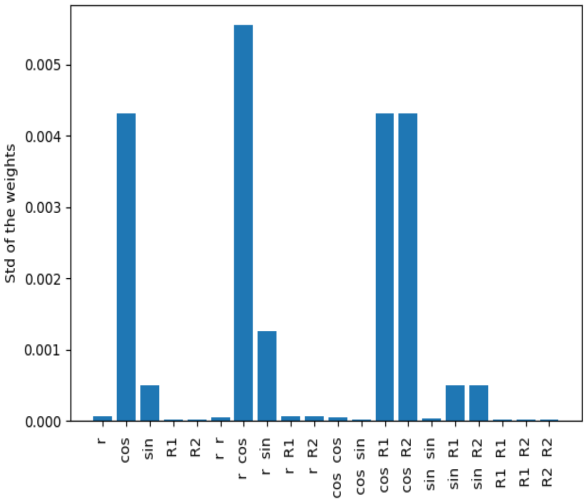


Figure 7.9: Median values of the weights of the message with highest standard deviation. It can be observed that the weights of $r \times cos(\theta)$ and $cos(\theta)$ have the largest values.

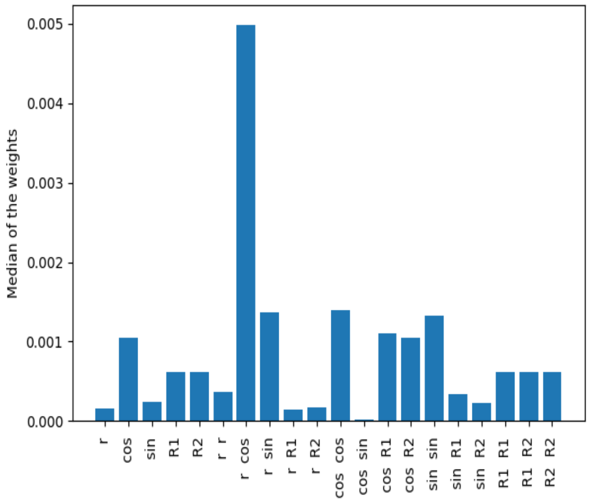


Figure 7.10: Median of the absolute value of $w_k f_k$. It can be observed that $r \times cos(\theta)$ is the largest. By considering the impact of the multiple *Radius*, it can be observed that the following terms with the highest medians correspond to $cos(\theta)$ and $Radius1$.

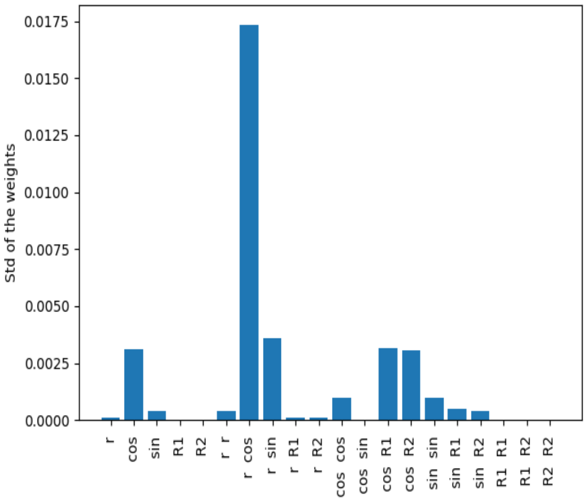


Figure 7.11: Standard deviation of the $w_k f_k$. It can be observed that the term in $w_k \times r \times cos(\theta)$ has the largest value. Many terms have standard deviations that are very small which indicates that these elements can be considered as constants.

Limitations:

Although some promising results have been achieved with this method, there is still room for improvement. One significant drawback is the somewhat arbitrary choice of basis functions. While it is possible to approximate any continuous functions using Taylor series or Fourier series, these approaches often result in a large number of basis functions which may be impractical for interpreting the model. A potential solution to this problem is to learn the basis functions using neural networks with an approach similar to Radenovic et al. [66]. It might also be possible to recursively apply this method to learn highly complex basis functions with sparse weights.

Another related issue is the inappropriate information repartition. An optimal model would have zero weights for all but the most significant basis functions, resulting in a much sparser and more interpretable model. However, the current results show that all weights vary. While it was possible to identify the most important terms by analyzing standard deviations, medians, and performing some simplifications, a model that is inherently simpler to interpret would be more desirable. Several potential solutions could be explored to address this. One approach is to train longer with more appropriate regularization. This might involve tuning the factor for the regularization or applying another regularization on the derivatives of the weights for instance. Another option is to use a ReLU right after the MLP. However, this often led to the dead ReLU problem when similar experiments were tried.

Chapter 8

Conclusion

This work aimed to explore various Graph Neural Network (GNN) architectures to simulate cellular migration systems. The experiments demonstrated that larger models with multiple GNN layers, such as EPD-Gat or EPD-GNN, achieved superior performance compared to smaller, single-layer models according to different summary statistics.

Moreover, it was found that single-layer models are easily interpretable. More precisely, similarly to previous works dealing with Newtonian systems [35], it was shown that linear transformations of the different interactions could be obtained by applying sparsity regularization to the messages with both classical and noisy versions of the system of equations considered here.

However, the application of symbolic regression presented challenges due to the complexity of the underlying analytic equations. To address this, the piecewise segments of the interactions were approximated, which allowed to use pySr on each of them to derive the ground truth equations.

The difficulties encountered with symbolic regression motivated the introduction of an end-to-end framework that generates easily interpretable equations. This framework achieves this by decomposing the messages into a weighted sum of predefined basis functions. In this context, the resulting weights are simple piecewise linear functions, which allows to directly derive the analytic expressions by analyzing the magnitudes of the weights.

Multiple works can be realized to further the study here. Firstly, it would be interesting to have a more realistic dataset that also incorporates cells with different radii, cell division and death to better reflect reality, and train the different architectures on it. Once satisfactory results are obtained, the next would consist of applying the method to real data to observe if the graph neural networks are able to generate trajectories similar to reality.

Another aspect that could be further explored is the application of sparsity regularization on architectures with multiple GNN layers. Moreover, the possibility of extending the theoretical demonstrations outlined in the appendix to multi-layer GNNs remains an open question.

Another critical consideration is the impact of the graph structure on the network's ability to recover the analytic forms of the interactions. Indeed, given that cellular systems often involve a large number of cells, it is impractical to work with a complete graph. Missing edges can be expected to make the network struggle. However, different options such as introducing virtual nodes [67] might alleviate the issue.

While only the two message features with the largest standard deviation are kept for symbolic regression in this work, it was also highlighted that other message features could provide valuable insights into interactions. For instance, it is possible to directly recover the norm of the interactions among the features. Exploring methods to leverage this additional information could lead to improved model discovery techniques.

Finally, there is considerable further development that is possible with the Generalized Additive Model (GAM) symbolic method that is introduced. As explained in the corresponding section, instead of selecting predefined basis functions, it may be possible to generate them with learned neural networks. This approach could enable the recursive application of the method to generate symbolic models for complex interaction functions. Additionally, enhancing the regularization techniques could improve performance and interpretability. In this work, a single kind of interaction is considered. However, since forces can be summed, they could potentially be modelled within the symbolic GAM framework. A logical next step for this method would then consist in assessing its performance when dealing with multiple kinds of interactions.

Bibliography

[1] Ryan Keisler. Forecasting global weather with graph neural networks, 2022.

[2] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, and Pascal Friederich. Graph neural networks for materials science and chemistry, 2022.

[3] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021.

[4] Lize Pirenne. Graph neural networks for physical models of collective cell migration. Master's thesis, Université de Liège, 2023.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[7] Thierry Mora, Aleksandra Walczak, Lorenzo Castello, Francesco Ginelli, Stefania Melillo, Leonardo Parisi, Massimiliano Viale, Andrea Cavagna, and Irene Giardina. Questioning the activity of active matter in natural flocks of birds. 11 2015.

[8] Takuma Sugi, Hiroshi Ito, and Ken Nagai. Collective pattern formations of animals in active matter physics. *Biophysics and Physicobiology*, 18, 10 2021.

[9] David Hu, S. Phonekeo, E. Altshuler, and Francoise Brochard-Wyart. Entangled active matter: From cells to ants. *The European Physical Journal Special Topics*, 225:629–649, 07 2016.

[10] Daniel J. Needleman, Zvonimir Dogic, and Zvonimir Dogic. Active matter at the interface between materials science and cell biology. *Nature Reviews Materials*, 2:17048, 2017.

[11] Audrey Nsamela, Aidee Itandehui Garcia Zintzun, Thomas Montenegro-Johnson, and Juliane Simmchen. Colloidal active matter mimics the behavior of biological microorganisms—an overview. *Small*, 19, 08 2022.

[12] M. Reza Shaebani, Adam Wysocki, Roland G. Winkler, Gerhard Gompper, and Heiko Rieger. Computational models for active matter. *Nature Reviews Physics*, 2(4):181–199, March 2020.

[13] Ricard Alert and Xavier Trepat. Physical models of collective cell migration. *Annual Review of Condensed Matter Physics*, 11(1):77–101, March 2020.

[14] Namid R. Stillman, Silke Henkes, Roberto Mayor, and Gilles Louppe. Graph-informed simulation-based inference for models of active matter, 2023.

[15] Daniel A. Matoz-Fernandez, Kirsten Martens, Rastko Sknepnek, Jean-Louis Barrat, and Silke E. Henkes. Cell division and death inhibit glassy behaviour of confluent tissues. *Soft matter*, 13 17:3205–3212, 2016.

[16] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[17] Yongjin Choi and Krishna Kumar. Three-dimensional granular flow simulation using graph neural network-based learned simulator, 2023.

[18] Yongjin Choi and Krishna Kumar. Graph neural network-based surrogate model for granular flows, 2023.

[19] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016.

[20] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control, 2018.

[21] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Flexible neural representation for physics prediction, 2018.

[22] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018.

[23] Victor Bapst, Thomas Keck, Agnieszka Grabska-Barwinska, Craig Donner, Ekin Dogus Cubuk, Samuel S. Schoenholz, Annette Obika, Alexander W. R. Nelson, Trevor Back, Demis Hassabis, and Pushmeet Kohli. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16:448–454, 2020.

[24] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[25] Amin Haeri and Krzysztof Skonieczny. Three-dimensionsal granular flow continuum modeling via material point method with hyperelastic nonlocal granular fluidity. *Computer Methods in Applied Mechanics and Engineering*, 394:114904, 05 2022.

[26] Siyu He, Yin Li, Yu Feng, Shirley Ho, Siamak Ravanbakhsh, Wei Chen, and Barnabás Póczos. Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, 116(28):13825–13832, June 2019.

[27] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2:729–734 vol. 2, 2005.

[28] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[29] Jiawei Zhu, Chao Tao, Hanhan Deng, Ling Zhao, Pu Wang, Tao Lin, and Haifeng Li. Ast-gcn: Attribute-augmented spatiotemporal graph convolutional network for traffic forecasting, 2020.

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[31] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[32] Erik J Bekkers. B-spline cnns on lie groups, 2021.

[33] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data, 2020.

[34] Miles D. Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks, 2019.

[35] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases, 2020.

[36] Pablo Lemos, Niall Jeffrey, Miles Cranmer, Shirley Ho, and Peter Battaglia. Rediscovering orbital mechanics with machine learning, 2022.

[37] Matteo Tiezzi, Giuseppe Marra, Stefano Melacci, Marco Maggini, and Marco Gori. A lagrangian approach to information propagation in graph neural networks, 2020.

[38] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators, 2019.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[43] Takeomi Mizutani, Hisashi Haga, and Kazushige Kawabata. Data set for comparison of cellular dynamics between human aavs1 locus-modified and wild-type cells. *Data in Brief*, 6, 01 2016.

[44] Jean-Pierre Hansen and Loup Verlet. Phase transitions of the lennard-jones system. *Phys. Rev.*, 184:151–161, Aug 1969.

[45] Walter Kob and Hans Christian Andersen. Testing mode-coupling theory for a supercooled binary lennard-jones mixture i: The van hove correlation function. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics*, 51 5:4626–4641, 1995.

[46] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[47] Murray Campbell, A. Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artif. Intell.*, 134:57–83, 2002.

[48] Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. Evaluating saliency map explanations for convolutional neural networks: A user study, 2020.

[49] Gary Marcus. The next decade in ai: Four steps towards robust artificial intelligence, 2020.

[50] D. Gerwin. Information processing, data inferences, and scientific generalization. *Systems Research and Behavioral Science*, 19:314–325, 1974.

[51] Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: A review, 2023.

[52] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023.

[53] Yilong Xu, Yang Liu, and Hao Sun. Rsrm: Reinforcement symbolic regression machine, 2023.

[54] Ahmed M. Alaa and Mihaela van der Schaar. Demystifying black-box models with symbolic metamodels. In *Neural Information Processing Systems*, 2019.

[55] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: a physics-inspired method for symbolic regression, 2020.

[56] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity, 2020.

[57] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. Contemporary symbolic regression methods and their relative performance. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[58] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.

[59] Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2021.

[60] Benjamin L. Davis and Zehao Jin . Discovery of a planar black hole mass scaling relation for spiral galaxies. *The Astrophysical Journal Letters*, 956(1):L22, October 2023.

[61] Anja Butter, Tilman Plehn, Nathalie Soybelman, and Johann Brehmer. Back to the formula - lhc edition. *SciPost Physics*, 16(1), January 2024.

[62] Arthur Grundner, Tom Beucler, Pierre Gentine, and Veronika Eyring. Data-driven equation discovery of a cloud cover parameterization, 2024.

[63] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[64] T.J. Hastie and R.J. Tibshirani. Generalized additive models. *Statistical Science*, 1986.

[65] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, March 2016.

[66] Filip Radenovic, Abhimanyu Dubey, and Dhruv Mahajan. Neural basis models for interpretability, 2022.

[67] Wenchuan Zhang, Weihua Ou, Weian Li, Jianping Gou, Wenjun Xiao, and Bin Liu. Robust graph structure learning with virtual nodes construction. *Mathematics*, 11:1397, 03 2023.

Appendix A

Ablation study

The different architecture choices that were performed are motivated based on the results that are available here. These models have been trained for 40 epochs.

Overall, it is observed that dropout rarely leads to improved performances. It is the opposite most of the time.

It is also observed that the one-step transition training leads to performances that are similar to each other most of the time compared to directly training rollout simulations. This is especially clear when comparing the evolution of the results when the number of layers changes in EPD-GAT and EPD-GNN. Moreover, it is also observed that models can reach great performances independently of the training process.

One of the inductive biases that was briefly explored consists in forcing opposite edges to have opposite messages. Since this is similar to the action-reaction principle, the corresponding networks are marked with 'ar'. However, the experiments here do allow to yield any conclusion outside of the fact that it does not largely worsen the performances.

When observing the effect of the L1 factor in the sparsity regularization, it is difficult to come up with any conclusions. The same can be said with the effect of the number of layers. While increasing the number of layers typically leads to an increase of performance, the errors are very close to each other.

The different results are accompanied with their standard deviations.

A.0.1 Interaction Network

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
L1 - 0.0001	0.00559 ± 0.00485	0.0798 ± 0.0798	0.00837 ± 0.00584	0.101 ± 0.00230
L1 - 0.001	0.00476 ± 0.00392	0.0720 ± 0.0720	0.00722 ± 0.00477	0.0872 ± 0.00199
L1 - 0.01	$\mathbf{0.00216 \pm 0.00242}$	$\mathbf{0.0529 \pm 0.0529}$	$\mathbf{0.00254 \pm 0.00300}$	$\mathbf{0.0485 \pm 0.00216}$
L1 - 1	0.00340 ± 0.00331	0.0755 ± 0.0755	0.00440 ± 0.00410	0.0731 ± 0.00135
L1-e-3-ar	0.00249 ± 0.00252	0.0687 ± 0.0687	0.00365 ± 0.00289	0.0534 ± 0.00146

Table A.1: Performances of the interaction network with different values of L1 regularization on the messages with non-noisy data. It can be observed that the L1 factor has a maximum value of 0.01.

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
L1 - 0.0001	0.00487 ± 0.00470	0.143 ± 0.143	0.00756 ± 0.00533	0.0908 ± 0.00214
L1 - 0.001	0.00356 ± 0.00349	0.142 ± 0.142	0.00562 ± 0.00395	0.0666 ± 0.00222
L1 - 0.01	0.00535 ± 0.00456	$\mathbf{0.117 \pm 0.117}$	0.00797 ± 0.00559	0.0953 ± 0.00207
L1 - 1	0.00525 ± 0.00487	0.121 ± 0.121	0.00808 ± 0.00558	0.0934 ± 0.00222
L1-e-3-ar	$\mathbf{0.00144 \pm 0.00180}$	0.142 ± 0.142	$\mathbf{0.00140 \pm 0.00206}$	$\mathbf{0.0361 \pm 0.00262}$

Table A.2: Performances of the interaction network with different values of L1 regularization on the messages with noisy data. It can be observed that the adding the inductive bias of action-reaction significantly improved the performances.

A.0.2 Baseline

	L1 Errors	**Angle Error**	**Norm Errors**	**MAE averaged**
No dropout-ar	0.00830 ± 0.00734	0.319 ± 0.319	0.0111 ± 0.00926	0.149 ± 0.00547
Dropout-ar	0.00767 ± 0.00707	$\mathbf{0.249 \pm 0.249}$	0.00965 ± 0.00888	0.148 ± 0.00395
No dropout	$\mathbf{0.00718 \pm 0.00649}$	0.304 ± 0.304	$\mathbf{0.00796 \pm 0.00831}$	$\mathbf{0.134 \pm 0.00478}$
Dropout	0.00755 ± 0.00694	0.255 ± 0.255	0.00900 ± 0.00889	0.142 ± 0.00554

Table A.3: Performances of the baseline network with non-noisy data. It can be observed that not using dropout leads to the best results overall

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
No dropout-ar	0.00739 ± 0.00680	0.308 ± 0.308	0.00887 ± 0.00863	$\mathbf{0.131 \pm 0.00385}$
Dropout-ar	0.00804 ± 0.00712	0.300 ± 0.300	0.00998 ± 0.00904	0.150 ± 0.00454
No dropout	$\mathbf{0.00678 \pm 0.00629}$	0.292 ± 0.292	$\mathbf{0.00700 \pm 0.00817}$	0.132 ± 0.00397
Dropout	0.00747 ± 0.00704	$\mathbf{0.262 \pm 0.262}$	0.00908 ± 0.00901	0.134 ± 0.00449

Table A.4: Performances of the baseline network with noisy data.

A.0.3 EPD-GNN

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000189 ± 0.00108	0.0322 ± 0.0322	0.0000785 ± 0.000982	0.0245 ± 0.00238
nb layer 2	0.000179 ± 0.00107	0.0403 ± 0.0403	0.0000563 ± 0.000960	0.0233 ± 0.00243
nb layer 3	$\mathbf{0.000174 \pm 0.00109}$	$\mathbf{0.0266 \pm 0.0266}$	0.0000435 ± 0.000994	$\mathbf{0.0230 \pm 0.00245}$
nb layer 4	0.000177 ± 0.00110	0.0359 ± 0.0359	$\mathbf{0.0000385 \pm 0.00101}$	0.0238 ± 0.00244

Table A.5: Performances of the EPD-GNN network with respect to the number of layers (in non-noisy data). It can be observed that increasing the number of layers leads to slightly better performances.

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000910 ± 0.00133	0.133 ± 0.133	**0.000368 ± 0.00126**	0.0320 ± 0.00267
nb layer 2	0.000918 ± 0.00134	0.145 ± 0.145	0.000369 ± 0.00127	0.0320 ± 0.00266
nb layer 3	**0.000889 ± 0.00130**	**0.131 ± 0.131**	0.000390 ± 0.00122	**0.0312 ± 0.00254**
nb layer 4	0.000921 ± 0.00134	0.145 ± 0.145	0.000370 ± 0.00127	0.0323 ± 0.00270

Table A.6: Performances of the EPD-GNN network with respect to the number of layers (in noisy data).It can be observed that the model using 3 layers has the best performances.

A.0.4 EPD-GaT

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	**0.000181 ± 0.00108**	0.0370 ± 0.0370	**0.0000444 ± 0.000981**	**0.0238 ± 0.00248**
nb layer 2	0.000211 ± 0.00106	0.0394 ± 0.0394	0.0000827 ± 0.000948	0.0257 ± 0.00211
nb layer 3	0.000196 ± 0.00107	0.0407 ± 0.0407	0.0000643 ± 0.000961	0.0249 ± 0.00218
nb layer 4	0.000191 ± 0.00108	**0.0283 ± 0.0283**	0.0000540 ± 0.000980	0.0239 ± 0.00228

Table A.7: Performances of the EPD-Gat network with respect to the number of layers (in non - noisy data). The models using 1 and 4 layers have the best performance.

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000917 ± 0.00134	**0.132 ± 0.132**	**0.000379 ± 0.00127**	0.0323 ± 0.00272
nb Layer 2	**0.000916 ± 0.00133**	0.135 ± 0.135	0.000382 ± 0.00124	0.0316 ± 0.00252
nb layer 3	0.000919 ± 0.00133	0.136 ± 0.136	0.000385 ± 0.00122	0.03098 ± 0.00243
nb layer 4	0.000917 ± 0.00133	0.135 ± 0.135	0.000381 ± 0.00126	**0.0307 ± 0.00258**

Table A.8: Performances of the EPD-Gat network with respect to the number of layers (in noisy data). The model using 4 layers has the best performance.

A.0.5 Rollout Interaction Network

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
Non-noisy	0.00468 ± 0.00459	0.0754 ± 0.0754	0.00737 ± 0.00512	0.0710 ± 0.00153
Noisy	0.00270 ± 0.00289	0.150 ± 0.150	0.00316 ± 0.00329	0.0496 ± 0.00201

Table A.9: Performance of the interaction network when trained using rollout.

A.0.6 Rollout Baseline

	Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
Non-noisy	0.00761 ± 0.00755	0.268 ± 0.268	0.00821 ± 0.00968	0.07134 ± 0.00192
Noisy	0.00665 ± 0.00673	0.246 ± 0.246	0.00668 ± 0.00862	0.06928 ± 0.00166

Table A.10: Performance of the baseline model when trained using rollout.

A.0.7 Rollout EPD-GNN

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000900 ± 0.00123	0.0794 ± 0.0794	0.000732 ± 0.00119	0.0492 ± 0.00191
nb layer 2	0.000432 ± 0.00125	0.0501 ± 0.0501	0.000274 ± 0.00118	0.0300 ± 0.00221
nb layer 3	0.000492 ± 0.00147	**0.0445 ± 0.0445**	0.000370 ± 0.00151	0.0335 ± 0.00244
nb layer 4	**0.000423 ± 0.00119**	0.0553 ± 0.0553	**0.000230 ± 0.00104**	**0.0274 ± 0.00258**

Table A.11: Performance of the EPD-GNN model when trained using rollout. Using four GNN layers leads to the best results

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000952 ± 0.00138	**0.123 ± 0.123**	0.000544 ± 0.00130	0.0371 ± 0.00250
nb layer 2	**0.000923 ± 0.00141**	0.130 ± 0.130	0.000429 ± 0.00135	**0.0339 ± 0.00263**
nb layer 3	0.000991 ± 0.00174	0.130 ± 0.130	**0.000406 ± 0.00187**	0.0354 ± 0.00253
nb layer 4	0.00109 ± 0.00175	0.130 ± 0.130	0.000628 ± 0.00186	0.0400 ± 0.00256

Table A.12: Performance of the EPD-GNN model when trained using rollout. Using two GNN layers leads to the best results

A.0.8 Rollout EPD-GAT

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	**0.000283 ± 0.00104**	**0.0384 ± 0.0384**	**0.000128 ± 0.000867**	0.0298 ± 0.00206
nb layer 2	0.000319 ± 0.00102	0.0531 ± 0.0531	0.000215 ± 0.000801	**0.0280 ± 0.00214**
nb layer 3	0.00180 ± 0.00336	0.173 ± 0.173	0.00123 ± 0.00326	0.0565 ± 0.00311
nb layer 4	0.00633 ± 0.00466	0.348 ± 0.348	0.00615 ± 0.00430	0.0827 ± 0.00179

Table A.13: Performance of the EPD-Gat model when trained using rollout. Using two GNN layers leads to the best results

	L1 Errors	**Angle Error**	**Norm Errors List**	**MAE averaged**
nb layer 1	0.000971 ± 0.00126	0.137 ± 0.137	0.000626 ± 0.00109	0.0342 ± 0.00215
nb layer 2	**0.000848 ± 0.00115**	0.138 ± 0.138	0.000526 ± 0.000915	0.0277 ± 0.00228
nb layer 3	0.000886 ± 0.00121	**0.113 ± 0.113**	**0.000487 ± 0.00102**	**0.0268 ± 0.00228**
nb layer 4	0.000879 ± 0.00119	0.125 ± 0.125	0.000566 ± 0.00100	0.0285 ± 0.00212

Table A.14: Performance of the EPD-Gat model when trained using rollout. Using three GNN layers leads to the best results

A.1 Demonstrations

This section introduces the demonstration of the edges. As explained in the main section, Cranmer et al [35] were able to show that the messages were linear transforms of the forces by relying on some assumptions.

A.1.1 Demonstration in the literature

In the literature, the architectures have acceleration as output. The papers remain in simple scenarios relying on classical Newtonian mechanics such as gravitational systems. In this case, it is straightforward to demonstrate that the output of the edge modem ϕ^e is a linear transformation of the forces from Newton's second law ($F = ma$) and the linearity of the forces that dictates that the total force applied on an entity is the sum of all the forces applied on it ($F = \sum f$).

Let us consider an update model ϕ^v that exactly yields the acceleration:

$$\phi_i^v\left(\sum_{j\in\mathcal{N}_i}\phi^e(e_{ij})\right) = \phi_i^v\left(\sum_{j\in\mathcal{N}_i} m_{ij}\right) = a = \frac{F}{m} = \sum_{j\in\mathcal{N}_i}\frac{f_{ij}}{m}. \tag{A.1}$$

where $\phi^e(e_ij) = m_{ij}$ are the messages of the GNN and $\phi^v(\mathbf{v}_i, .) := \phi_i^v(.)$.

This result has to hold for any number of edges linked to a node. It is then possible to find the expression of the individual forces and insert it in the equation above.

$$\phi_i^v\left(m_{ij}\right) = a = \frac{f_{ij}}{m}. \tag{A.2}$$

$$\Rightarrow \sum_{j\in\mathcal{N}_i}\phi_i^v(m_{ij}) = \phi_i^v\left(\sum_{j\in\mathcal{N}_i} m_{ij}\right). \tag{A.3}$$

Therefore, ϕ_i^v is a linear operator. This operator is inversible when it is a bijection which occurs when the message dimension and the spatial dimensions are the same. Since the inverse of a linear transformation is also linear, it is possible to observe that the messages are linear transformations of the forces. Hence, neglecting the mass:

$$m_{ij} = \left(\phi_i^v\right)^{-1}\left(f_{ij}\right). \tag{A.4}$$

A.1.2 Demonstration for cellular systems

This demonstration can be extended to the non-noisy physical system of this work as follows. Let us start by recalling the form of the equation. In this demonstration, only the projected force is considered. Denoting the speed at timestep t with v_t, the active force with w_0 and the total interaction forces with $F = \sum_{j \in \mathcal{N}_i} f_{ij}$.

$$v_t = v_0 + F. \tag{A.5}$$

In this case, the update model is considered to yield the speed exactly:

$$v_t = \phi_i^v \left(\sum_{j \in \mathcal{N}_i} \phi^e(e_{ij}) \right) = \phi_i^v \left(\sum_{j \in \mathcal{N}_i} m_{ij} \right) = v_0 + F. \tag{A.6}$$

This is true for any number k of edges. By considering the case of a single edge and exploiting the form of the interaction, it is possible to obtain a new equation that has to be respected $\forall k$.

$$v_t = \phi_i^v \left(m_{ij} \right) = v_0 + f_{ij} \tag{A.7}$$

Inserting the new formulation of the individual force in the generic case of a node with k edges:

$$\phi_i^v \left(\sum_{j \in \mathcal{N}_i}^{k} m_{ij} \right) = v_0 + \sum_{j \in \mathcal{N}_i}^{k} (\phi_i^v \left(m_{ij} \right) - v_0) \qquad \forall k \tag{A.8}$$

$$\Leftrightarrow \phi_i^v \left(\sum_{j=1}^{k} m_{ij} \right) - v_0 = \sum_{j=1}^{k} (\phi_i^v \left(m_{ij} \right) - v_0) \qquad \forall k \tag{A.9}$$

It is then possible to introduce the new operator $\rho_i^v(.) := \phi_i^v(.) - v_0$.

$$\rho \left(\sum_{j=1}^{k} m_{ij} \right) = \sum_{j=1}^{k} \rho(m_{ij}) \tag{A.10}$$

This is a linear operator. By assumption, this is a bijection and is then inversible. A similar result as the one from the previous page is then obtained. The messages are linear transformations of the forces.

$$\rho(m_{ij}) = f_{ij} \tag{A.11}$$

$$\Rightarrow m_{ij} = \rho^{-1}(f_{ij}) \tag{A.12}$$

Appendix B

Additional figures

Additional results are displayed in this section.

B.1 Rollout training

Figures B.1 and B.2 are the evaluation of the models during training. The EPD models rely on two GNN layers. It can be observed that the EPD models have similar performances and outperform the baseline and the Interaction Network.

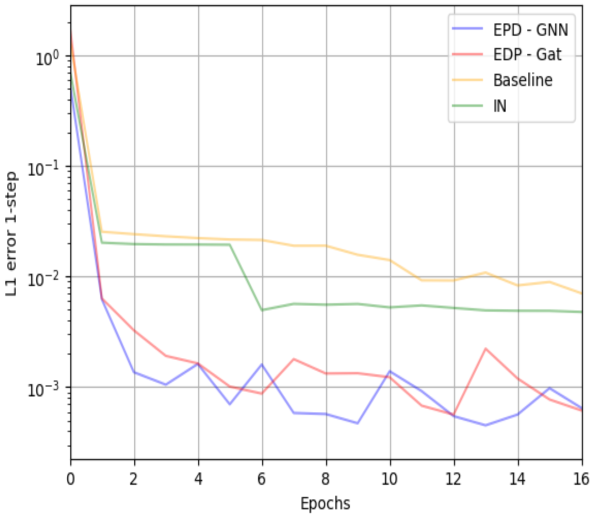


Figure B.1: Absolute error on step transitions when models are trained with rollout.

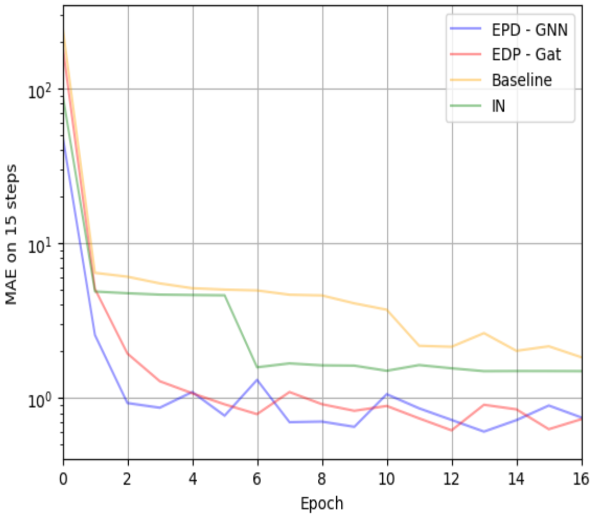


Figure B.2: MAE error on 15 rollout steps for models when training with rollout.

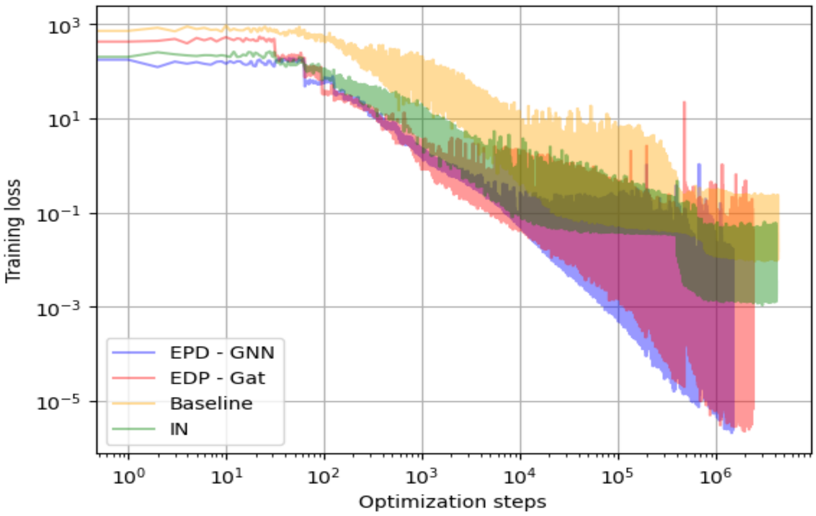


Figure B.3: Rollout training loss of the different models

B.2 Speed distributions of the main models

Figures B.4 - B.7 show the speed distribution of different models

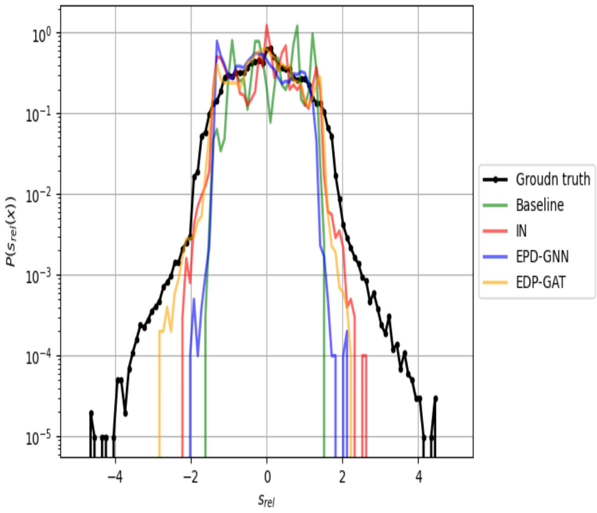


Figure B.4: Speed norm distribution for non-noisy data along x coordinate.

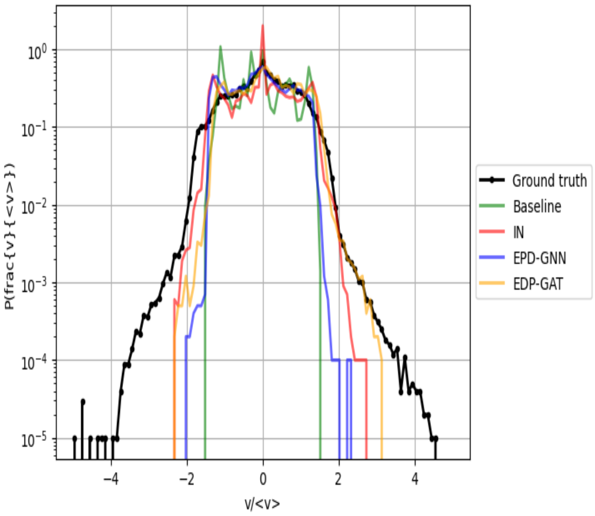


Figure B.5: Relative speed histograms along x coordinate (noisy data).

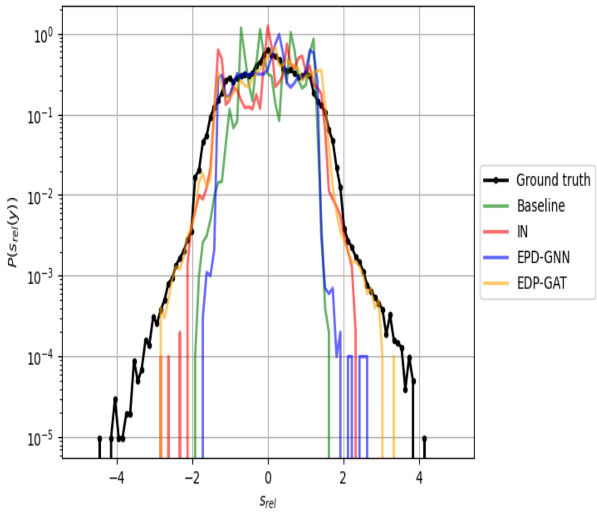


Figure B.6: Speed norm distribution for non-noisy data along y coordinate.

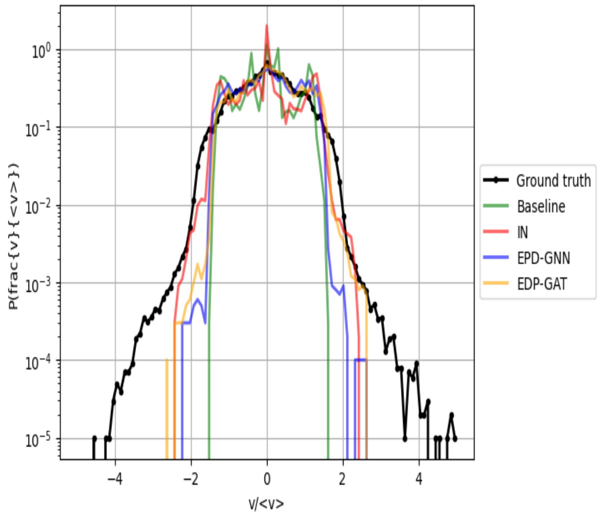


Figure B.7: Relative speed histograms along y coordinate along y coordinate (noisy data).

As explained in the main section, the interaction network and the GaT network seem to perform better than other networks when this statistics is observed.

B.3 Message of Interaction Network with dropout

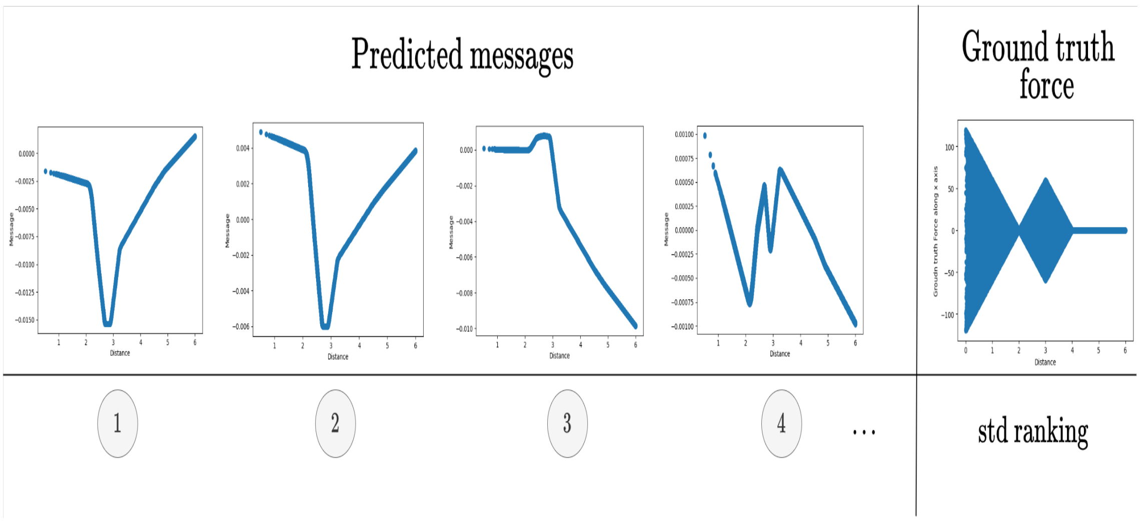


Figure B.8: Impact of the distance on the most relevant features when dropout is applied. It can be observed that the network does not have the linear transformation of the interaction in its messages anymore

B.4 Messages of the baseline network

The messages of the baseline are displayed below. It can be observed that the network understood the limits of the different segments of the ground truth interactions. However, the outputs are not linear transformations of the ground truth interactions.

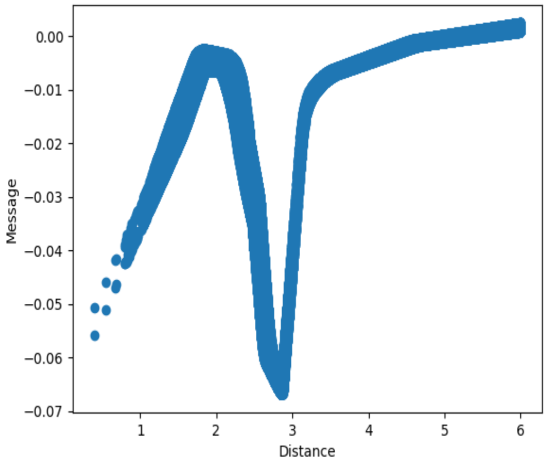


Figure B.9: Evolution of the first message of the baseline network with respect to the distance.

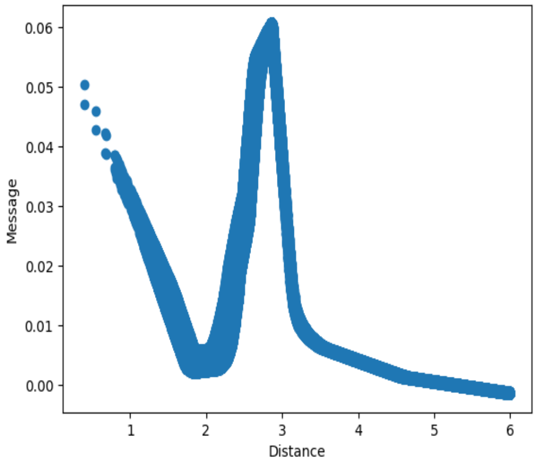


Figure B.10: Evolution of the second message of the baseline network with respect to the distance.

B.5 GAM

In this section, different figures related to the GAM section are displayed. More precisely, the evolution of the weights w_k and the terms $w_k f_k$ are displayed with respect to the distance.

B.5.1 GAM - non-noisy data - 128

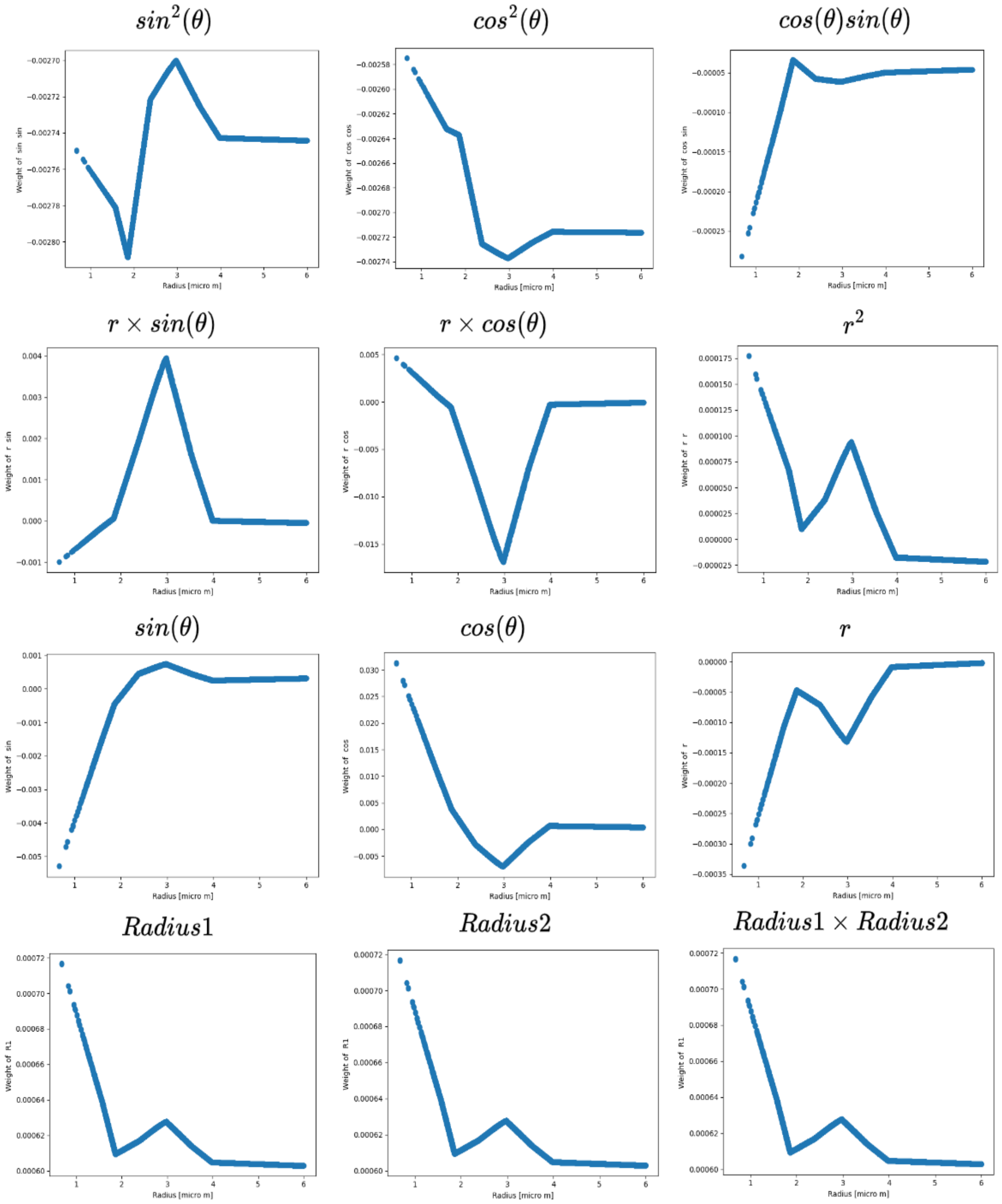


Figure B.11: Evolution of the weights of the neural network with respect to the distance.

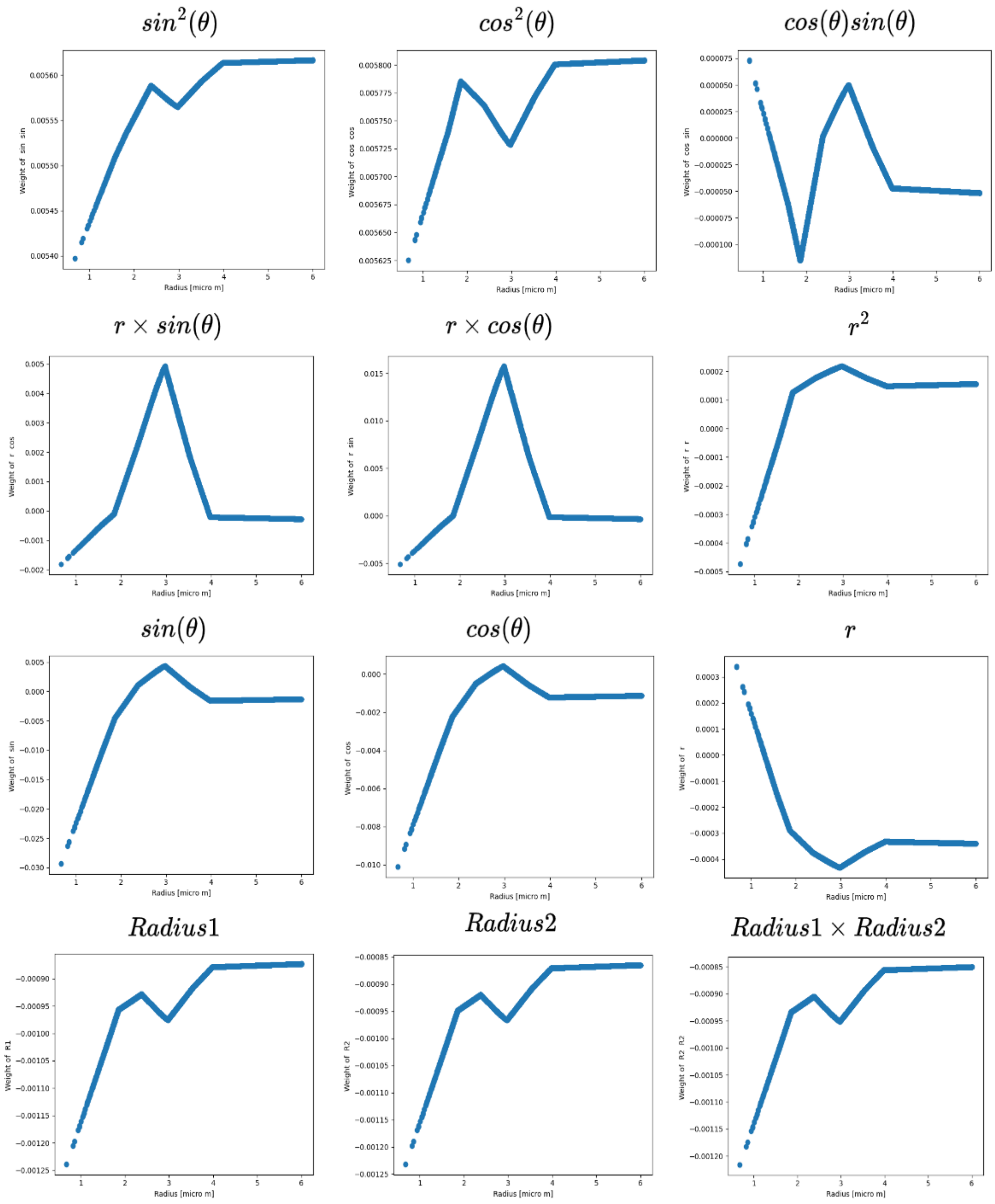
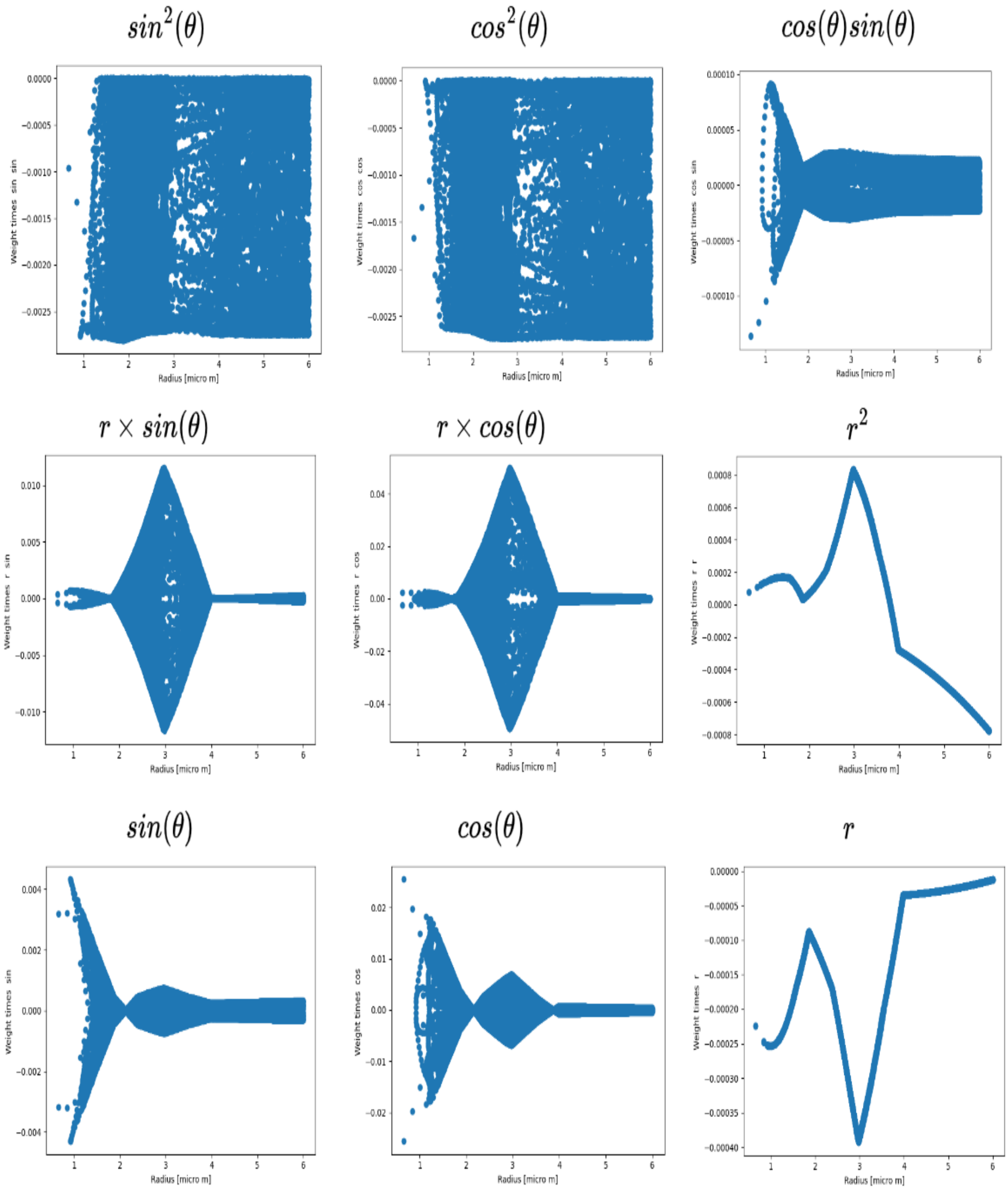


Figure B.12: Evolution of the weights of the neural network with respect to the distance. This graph illustrates the weights of the second-largest standard deviation.


$sin^2(\theta)$ $cos^2(\theta)$ $cos(\theta)sin(\theta)$

$r \times sin(\theta)$ $r \times cos(\theta)$ r^2

$sin(\theta)$ $cos(\theta)$ r

Figure B.13: Illustration of the $f_k w_k$ for the first message. It can be observed that all the messages share a similar shape with the ground truth interactions.

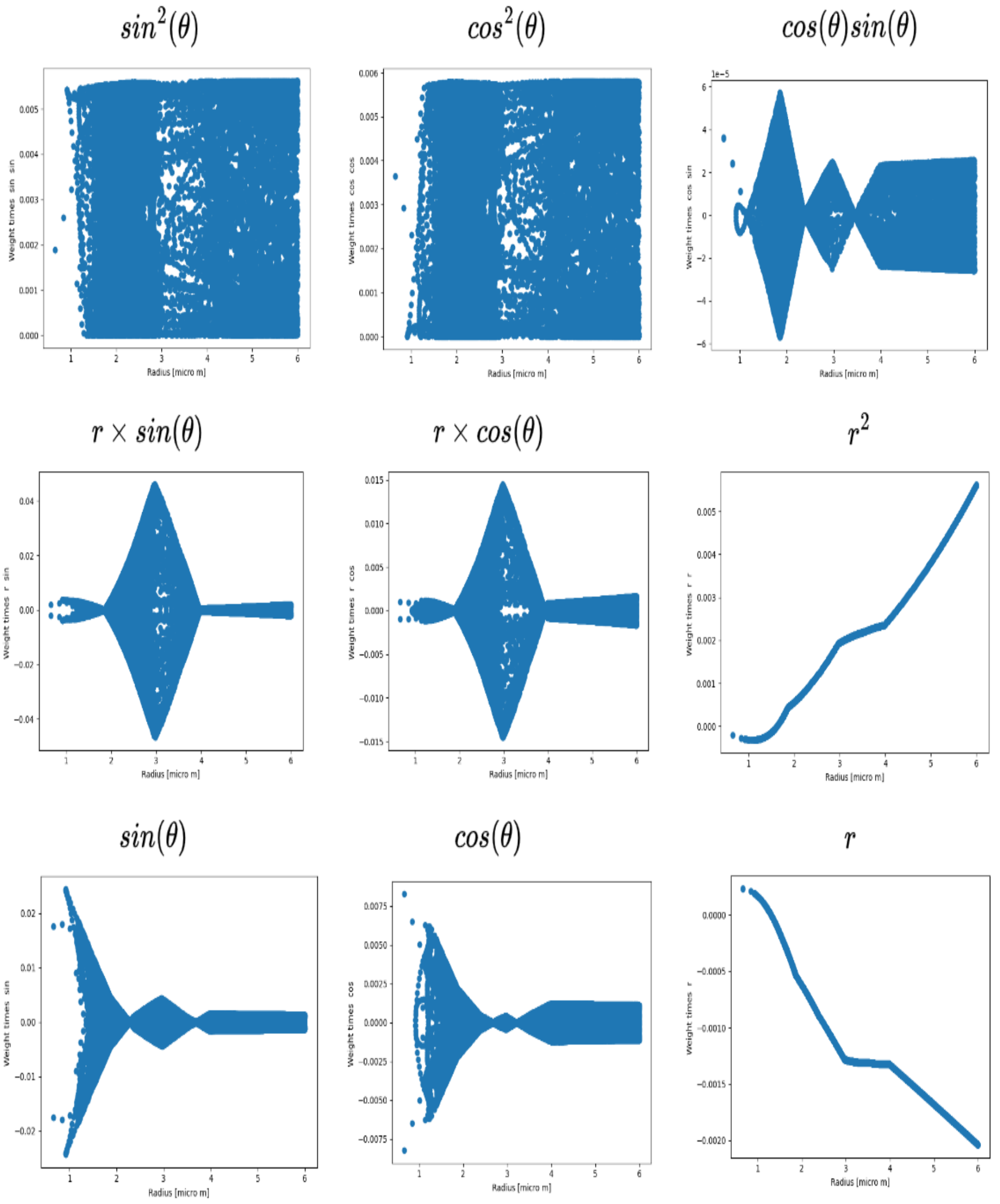


Figure B.14: Illustration of the $f_k w_k$ for the second message. It can be observed that all the messages share a similar shape with the ground truth interactions.

These figures show the network has learned similar weights for the terms related to the cosine and sine increasing linearly with the distance which is expected as cosine and sine have the same nature. Moreover, it can be observed that the weights of the cos^2 and sin^2 terms have a very small range which further motivates that their sum can be considered as a constant. However, it also highlights a limitation of this method for now: even though the network learns that the variables should be comprised within a small range, it does not learn to create constants. It is even possible to observe the impact of the different segments in them.

B.6 PySr Equations:

PySr raw results for the different architectures are described in this section.

The pySr results for the different segments are available below. It can be observed in each of them that the analytic equations are recovered. The table of the PySR results for the whole equation was to big to be put in a single page.

Non-noisy

Equation between 0 **and** 2

Equation	Complexity	Loss	Score
$y_0 = -8.75 \cdot 10^{-5}$	1	$8.53 \cdot 10^{-5}$	0.0
$y_0 = \cos(\theta)\,(-0.0112)$	5	$2.32 \cdot 10^{-5}$	0.326
$y_0 = (\cos(\theta)\,(-0.0112) + 0.00780)$	7	$2.32 \cdot 10^{-5}$	0.000165
$y_0 = (r - 1.99)\cos(\theta)\,(0.0302)$	9	$5.69 \cdot 10^{-6}$	0.702
$y_0 = (r - 1.99) \cdot 0.0302 \cdot \cos(\theta)\,(-8.75 \cdot 10^{-5})$	11	$5.69 \cdot 10^{-6}$	0.000674

Table B.1: Results for y_0 Equations in $[0, 2]$. (non-noisy)

Equation	Complexity	Loss	Score
$y_1 = -5.81 \cdot 10^{-5}$	1	$8.53 \cdot 10^{-5}$	0.0
$y_1 = \sin(\theta)\,(-0.0111)$	5	$2.29 \cdot 10^{-5}$	0.329
$y_1 = (\sin(\theta)\,(-0.0111) + 0.00530)$	7	$2.29 \cdot 10^{-5}$	$7.55 \cdot 10^{-5}$
$y_1 = \sin(\theta)\,(r \cdot 0.0308 - 0.0614)$	9	$5.39 \cdot 10^{-6}$	0.724
$y_1 = (\sin(\theta)\,(r \cdot 0.0308 - 0.0614) + 0.00530)$	11	$5.38 \cdot 10^{-6}$	0.000387

Table B.2: Results for y_1 Equations in $[0, 2]$.(non-noisy)

Equation between 2 **and** 3

Equation	Complexity	Loss	Score
$y_0 = -9.66 \cdot 10^{-5}$	1	0.000198	0.0
$y_0 = 0.0175 \cdot \cos(\theta)$	3	$3.48 \cdot 10^{-5}$	0.870
$y_0 = \cos(\theta)(0.00699 \cdot r)$	5	$2.53 \cdot 10^{-5}$	0.159
$y_0 = \cos(\theta)\,(r \cdot 0.0290 - 0.0569)$	7	$1.22 \cdot 10^{-5}$	0.367
$y_0 = (\cos(\theta) + 0.00178) \cdot 0.0290 \cdot (r - 1.96)$	9	$1.21 \cdot 10^{-5}$	$5.90 \cdot 10^{-5}$

Table B.3: Results for y_0 Equations in $[2, 3]$.(non-noisy)

Equation	Complexity	Loss	Score
$y_1 = 2.38 \cdot 10^{-5}$	1	0.000181	0.0
$y_1 = \sin(\theta)(0.0175)$	3	$3.85 \cdot 10^{-5}$	0.774
$y_1 = 0.00703 \cdot r \cdot \sin(\theta)$	5	$2.86 \cdot 10^{-5}$	0.149
$y_1 = \sin(\theta)\,(r \cdot 0.0319 - 0.0642)$	7	$1.30 \cdot 10^{-5}$	0.396

Table B.4: Results for y_1 Equations in $[2, 3]$.(non-noisy)

Equation between 3 **and** 4

Equation	Complexity	Loss	Score
$y_0 = 3.01 \cdot 10^{-5}$	1	0.000122	0.0
$y_0 = \cos(\theta)(0.0131)$	5	$3.21 \cdot 10^{-5}$	0.334
$y_0 = 0.0131\,(\cos(\theta)(0.0131) + 0.00229)$	7	$3.21 \cdot 10^{-5}$	$1.42 \cdot 10^{-5}$
$y_0 = \cos(\theta)\,(0.115 + r \cdot (-0.0288))$	9	$8.56 \cdot 10^{-6}$	0.661
$y_0 = (0.115 + r \cdot (-0.0288)) \cdot (\cos(\theta)(0.0131) + 0.00229)$	11	$8.56 \cdot 10^{-6}$	$6.40 \cdot 10^{-5}$

Table B.5: Results for y_0 Equations in $[3, 4]$.(non-noisy)

Equation	Complexity	Loss	Score
$y_1 = 9.32 \cdot 10^{-5}$	1	0.000127	0.0
$y_1 = \sin(\theta)(0.0139)$	5	$3.43 \cdot 10^{-5}$	0.327
$y_1 = (\sin(\theta)(0.0139) + 0.00664) \cdot 0.0139$	7	$3.42 \cdot 10^{-5}$	0.000125
$y_1 = \sin(\theta)\,(0.124 - 0.0312 \cdot r)$	9	$7.81 \cdot 10^{-6}$	0.739
$y_1 = (0.124 - 0.0312 \cdot r) \cdot (\sin(\theta)(0.0139) + 0.00888)$	11	$7.80 \cdot 10^{-6}$	0.000573

Table B.6: Results for y_1 Equations in $[3, 4]$.(non-noisy)

PySr results with noisy dataset

Equation between 0 **and** 2

Equation	Complexity	Loss	Score
$y_0 = 2.85 \cdot 10^{-5}$	1	$7.56 \cdot 10^{-5}$	0.0
$y_0 = -0.0105 \cdot \sin(\theta)$	5	$2.03 \cdot 10^{-5}$	0.328
$y_0 = -0.0105(\sin(\theta) - 0.00272)$	7	$2.03 \cdot 10^{-5}$	$2.00 \cdot 10^{-5}$
$y_0 = \sin(\theta)(0.0305 \cdot r - 1.99)$	9	$8.06 \cdot 10^{-7}$	1.61
$y_0 = (r - 1.99) \cdot \sin(\theta)(0.0312)$	11	$7.77 \cdot 10^{-7}$	0.0186

Table B.7: PySr results of the first message feature in $[0, 2]$.(noisy)

Equation	Complexity	Loss	Score
$y_1 = 1.66 \cdot 10^{-5}$	1	$6.64 \cdot 10^{-5}$	0.0
$y_1 = 1.34 \cdot 10^{-5} \cdot r$	3	$6.64 \cdot 10^{-5}$	$1.58 \cdot 10^{-6}$
$y_1 = -0.0100 \cdot \cos(\theta)$	5	$1.62 \cdot 10^{-5}$	0.705
$y_1 = 1.70 \cdot 10^{-5} - 0.0100 \cdot \cos(\theta)$	7	$1.62 \cdot 10^{-5}$	$8.89 \cdot 10^{-6}$
$y_1 = \cos(\theta) \cdot (0.0307 \cdot r - 0.0613)$	9	$5.80 \cdot 10^{-7}$	1.67
$y_1 = \cos(\theta)(0.0307 \cdot r - 0.0613) + 1.70 \cdot 10^{-5}$	11	$5.80 \cdot 10^{-7}$	0.000250
$y_1 = \cos(\theta)(0.0307 \cdot r - 0.0613) + r \cdot 1.34 \cdot 10^{-5}$	13	$5.79 \cdot 10^{-7}$	0.000182

Table B.8: PySr results of the second message feature in $[0, 2]$. (noisy)

Equation between 2 and 3

Equation	Complexity	Loss	Score
$y_0 = -8.74 \cdot 10^{-5}$	1	0.000174	0.0
$y_0 = -3.43 \cdot 10^{-5} \cdot r$	3	0.000174	$4.31 \cdot 10^{-7}$
$y_0 = 0.0165 \cdot \sin(\theta)$	5	$2.92 \cdot 10^{-5}$	0.891
$y_0 = \sin(\theta) \cdot r \cdot 0.00672$	7	$1.84 \cdot 10^{-5}$	0.232
$y_0 = \sin(\theta)(0.0297 \cdot r - 0.0587)$	9	$1.83 \cdot 10^{-6}$	1.15
$y_0 = (0.0297 \cdot r - 0.0587)(\sin(\theta) - 0.00850)$	11	$1.81 \cdot 10^{-6}$	0.00673
$y_0 = \sin(\theta)(r \cdot (0.0593 + r \cdot (-0.00590)) - 0.0956)$	13	$1.79 \cdot 10^{-6}$	0.00510

PySr results of the first message feature in $[2, 3]$. (noisy)

Equation	Complexity	Loss	Score
$y_1 = 3.19 \cdot 10^{-5}$	1	0.000166	0.0
$y_1 = 0.0175 \cdot \cos(\theta)$	5	$2.22 \cdot 10^{-5}$	0.503
$y_1 = 0.00701 \cdot \cos(\theta) \cdot r$	7	$1.38 \cdot 10^{-5}$	0.240
$y_1 = \cos(\theta) \cdot (0.0305 \cdot r - 0.0606)$	9	$1.17 \cdot 10^{-6}$	1.23

Table B.9: PySr results of the second message feature in $[2, 3]$. (noisy)

Equation between 3 **and** 4

Equation	Complexity	Loss	Score
$y_0 = -0.000167$	1	0.000115	0.0
$y_0 = -4.72 \cdot 10^{-5} \cdot r$	3	0.000115	$2.17 \cdot 10^{-7}$
$y_0 = 0.0138 \cdot \sin(\theta)$	5	$2.65 \cdot 10^{-5}$	0.735
$y_0 = 0.0138 \cdot \sin(\theta) - 0.000167$	7	$2.65 \cdot 10^{-5}$	0.000527
$y_0 = \sin(\theta)(0.119 - 0.0299 \cdot r)$	9	$1.99 \cdot 10^{-6}$	1.29
$y_0 = \sin(\theta)(0.119 + r \cdot (-0.0299)) - 0.000167$	11	$1.97 \cdot 10^{-6}$	0.00705
$y_0 = r \cdot (-4.72 \cdot 10^{-5}) + \sin(\theta)(0.119 + r \cdot (-0.0299))$	13	$1.97 \cdot 10^{-6}$	$1.76 \cdot 10^{-5}$

Table B.10: PySr results of the first message feature in $[3, 4]$. (noisy)

Equation	Complexity	Loss	Score
$y_1 = 6.25 \cdot 10^{-5}$	1	0.000123	0.0
$y_1 = 0.0133 \cdot \cos(\theta)$	5	$2.86 \cdot 10^{-5}$	0.365
$y_1 = 0.0133(\cos(\theta) + 0.00475)$	7	$2.86 \cdot 10^{-5}$	$6.96 \cdot 10^{-5}$
$y_1 = 0.0133(\cos(\theta) + 1.13 \cdot 0.00421)$	9	$2.86 \cdot 10^{-5}$	$6.99 \cdot 10^{-8}$
$y_1 = \cos(\theta)\,((-1.75) \cdot r \cdot 0.0172 + 0.120)$	11	$9.69 \cdot 10^{-7}$	1.69

Table B.11: PySr results of the second message feature in $[3, 4]$. (noisy)