Graph Neural Networks for Physical Models of Collective Cell Migration

Auteur : Pirenne, Lize
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2022-2023
URI/URL : https://github.com/Pangasius/graph-displacement; http://hdl.handle.net/2268.2/18186

Graph Neural Networks for Physical Models of Collective Cell Migration

Université de Liège - Faculté des Sciences Appliquées

Travail de fin d'études réalisé en vue de l'obtention du grade de master Ingénieur Civil en informatique, à finalité spécialisée en "intelligent systems" par Pirenne Lize.

Avec Louppe Gilles en tant que promotteur académique.

Avec Stillman Namid en tant que directeur de thèse.

Année académique 2022-2023

1 Introduction

Experimentation represents the basis of all modern sciences. "Physics, and natural science in general, is a reasonable enterprise based on valid experimental evidence, criticism, and rational discussion" [1] explains the Stanford encyclopedia of philosophy. The interest in the world that surrounds us resides in the very fact that every action has a consequence and humans want to understand the mechanics hidden underneath.

Often times, one can directly observe the consequences to verify their hypothesis. A concrete example for this approach can be the analysis of fatigue and stress in materials. The experimenter repeatedly flexes and bends an object until comes the inevitable forecast event of a plastic deformation or the complete rupture [2].

The immediate reaction of the reader to this experiment might depend on the background but most could agree that such data collection, while fundamental, quickly becomes expensive both in time and money. In construction and hydraulics, the use of scaled down versions that can model parts of the complete and intricate physics has continued staying relevant over the years [3]. However, even small, such tangible models take time to create and are often built and applicable for only a singular project.

The rise of computational power following the self-fulfilling prophecy of Moore's Law [4] over the past fifty year has introduced the shift of perspective from the physical to the digital. The transition eventually led to the description of finite element methods [5]. In this framework, the space gets subdivided into finite elements, which are localised enough to be considered homogeneous, described by constant values. These can be used to solve partial differential equations that describe the exact physics behind a phenomenon.

When the underlying mechanics are not entirely defined or known and the models must make statistical inference, they rapidly become powerless; Indeed, working with probabilities often entails the description of the likelihood, the joint probability $p(x, y)$ of the observed data x as a function of the parameters of the model y, described in eq. (1) for the discrete case. This equation is marginalized over the unobserved variable z.

$$L(y|x) = p(x|y) = \frac{\sum_{x,z} p(x,y,z)}{\sum_{z} p(x,y,z)} \tag{1}$$

The computational power needed to estimate the likelihood of an observation grows intractably [6]. Some techniques can work with additional assumptions to reduce the cost of one computation, but the ultimate goal is to find the model whose parameters y maximize the likelihood; in other words, the goal is to best explain the outcomes with a model. This creates a necessity to compute $argmax_y L(y|x)$, the optimal parameters, and this becomes even more intractable. One cannot simply hope for Moore's Law to come to the rescue on this issue; it has recently encountered its first barriers with the size of CMOS reaching their theoretical minimum and the law's continuity now depends on the ingenuity of the physical designs more than ever [7].

One of the alternative solution that does not rely entirely on the sheer amount of computation that can be done comes from a branch of artificial intelligence : neural networks. The neural network architecture attempts to mimic the inner workings of the brain, the neurons [8]. The functions of a singular neuron consists in receiving a signal which gets integrated and finally communicated to other neurons. Since only electrical signals can be propagated, the composition of the messages can be interpreted a set of real numbers, turned into another set of real numbers. The most important feature comes from the fact that the integration part can be non-linear, but stays simple. The aggregation of neurons generates a powerful net capable of simulating any effectively computable function, provided they are correctly organized [9], while keeping a modest size.

Rather than computing directly all optimal y, the training of a neural network in a supervised setting aims to slowly converge towards a local extrema of the likelihood function for a set of input-output pairs x, y. Naturally, the more pairs are available, the better the approximation of the true likelihood, thus leading to a 'more general' local extrema.

Using this capable modelling solution, the focus can be shifted to the problem at the heart of this work.

Biology often require modeling interactions between different objects where many variables are hardly observable, whether it be molecules

and proteins [10], cells or viruses [11], or more macro-scale projects like population migration [12].

The smallest elements create mechanisms analogous to communication due to their proximity using the very laws of physics. The ways in which proteins could arrange are astronomical. However, purified proteins refold spontaneously in vitro after being completely unfolded. This means that the particular disposition of the distances and angles between atoms is determined by their primary structure, and the invisible mechanisms they interact with [13].

Further up in scale, one can study for example the interactions of viruses with one another. They can exchange and interact using genes directly, interact by the intermediary of the host in which they live in, or even by immunological interactions [14]. Those types of communication, while different in their exact nature, can be analogous to the different processes and means by which 'the living' organizes. In a more abstract description, there are many direct, indirect, implicit or explicit ways in which one or more individuals share and influence others in their immediate or distant surroundings. In term, a network of individuals, objects and mechanisms is constructed and explains perfectly the spontaneous emergence of collective behaviours. The field of collective dynamics focuses on studying the emerging properties that naturally arise when these complex systems develop over time.

These networks can be understood as generic graphs, which are agnostic to the particular disposition and details of the inner workings of the group studied. Studying the evolution of different systems can be done efficiently using those graphs in association with the power of neural networks. Given an initial disposition of each element an their connection to one another, one can model how the system would evolve or what properties would emerge.

This contributed to the recent interest in Graph Neural Networks [15], or GNNs for short. Such a framework has shown numerous times its ability to model communication networks efficiently and accurately [16]. Large data-sets have been compiled and methods were developed to harvest the similarities in the behavior in order to ease the transition to a more specific and smaller problem [17].

Most use cases of GNNs rely on the attribution of different charac-

teristics computed using the combined information at a graph level to then classify the individuals or identify particular characteristics [18, 19]. Some focus on modeling the interaction of arbitrary particles [20] and even extract their data from videos [21]. In biology, most group dynamics studies try to predict events such as zebrafish turning [22] or cell dividing and delaminating [23], see fig. 1.

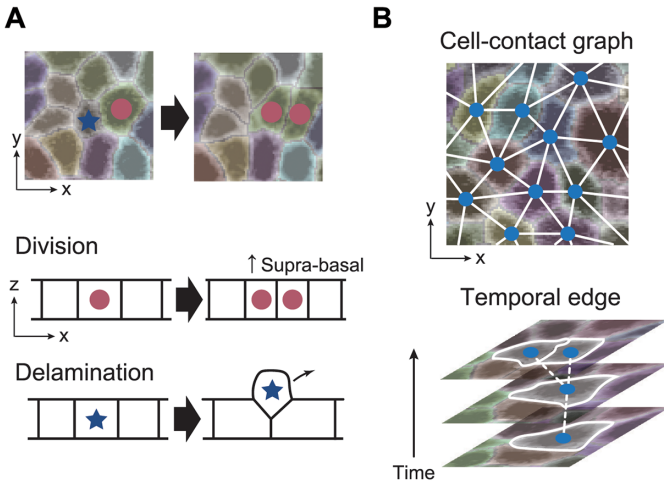


Figure 1: Use of a GNN to study delimination. (A) Schematic of cell division and delamination in the basal layer of the mouse epidermis. (B) Schematic of the cell-contact graph and temporal edges in the basal layer.[23]

In computational biology, the simulation of cell group mechanics can be used to understand the healing of wounds, the development of cancers or even embryo development [24]. Indeed, being able to accurately follow the evolution of the ratio of a cell's perimeter to its area, the jamming index [25], can help predicting changes in behavior; high motility leads to the index being higher and, for example, in a cancer of epithelial cells, the non-motile original structure can evolve into a partly motile tumor [26]. Due to this correlation between the movement and shape amongst other factors, it is interesting to follow and study each cell's trajectory in its environment.

During this thesis, some data follow the displacement of neural crest cells. These are responsible for a wide range of tissues and structures, including parts of the nervous system, the jaw and skull bones, and the face tissue. In addition to their multipotentiality in the living, these cells are highly migratory and their migratory behaviour is understood to be

critical in developmental pathways [1].

The individual and global study of these displacements constitute the field of cell migration.

The methods used traditionally rely on equations modelling the most important interactions to make the predictions close to reality. Those predictions are then verified using summary statistics, computed using the behavior as a group of the cells with the ultimate purpose of closely matching experimental data. These vary from the number of cells over time to the average pressure experienced by individual cells, such an example can be observed in fig. 2 [29].

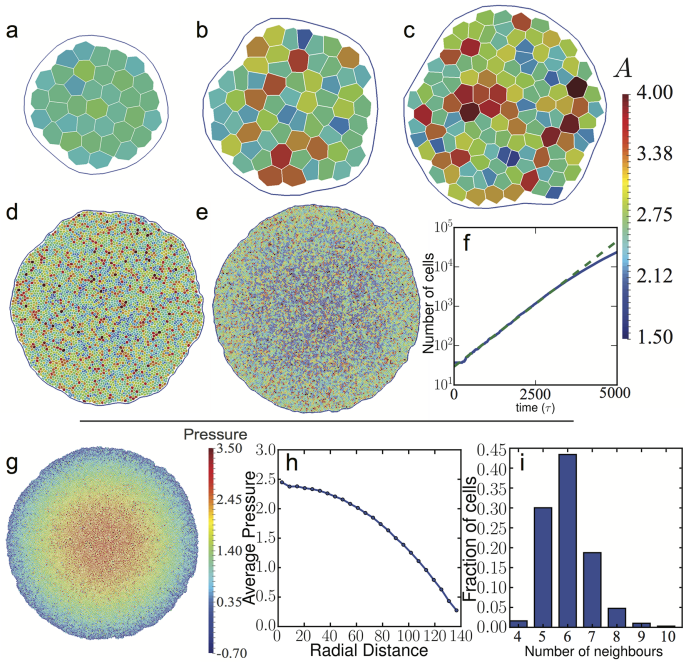


Figure 2: Snapshot of a simulation of growing epithelial tissue, together with summary statistics [29]

While it is not expected for these models to describe all the underlying mechanisms, they are used as a conceptual tool to explain emerging features and plan future experiments. They are often composed of few parameters, most falling under the tens. In fig. 3, one can understand the importance of cell migration in the development of the living.

[1]Paraphrased from Stillman Namid, supported by [27], [28]

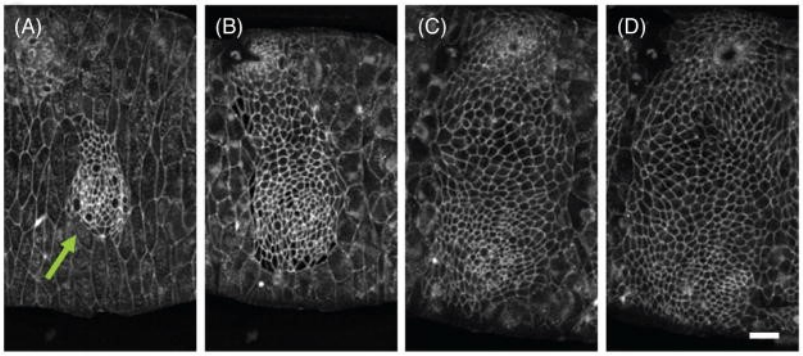


Figure 3: Collective cell migration in development. Abdomen of Drosophila melanogaster a cluster of histoblasts (green arrow) [30]

The other category of models used is well over-parameterized, and appear in the form of Deep Neural Networks. This shift in framework often allows for predictions consistent with the data at the cost of interpretability. A welcomed byproduct of this trade-off is the ability to construct these models without prior knowledge on the subjects studied [31, 30]

When it pertains to cell migration, successful work has been done to decrease the amount mislocated events [32], but the simulation of trajectories based on real data has, to my knowledge, not been studied with GNN.

In addition to helping understand cell migration, GNNs have been used to help produce more accurate tracking in real data [33]. The model created for this thesis can ultimately provide an additional tool to help follow trajectories more accurately and in turn help produce better models.

In this thesis, a graph neural network generates likely distributions of the set of points corresponding to the positions of cells after a discrete time step from a particular initial disposition. The predictions of this probabilistic program will be sampled to be used over and over again to simulate the trajectories of each cell. From this generated set of trajectories, summary statistics that reflect the expected behavior will be compared against the same statistics computed on 2D synthetic data generated from a cell migration simulator and then real, computer annotated, data coming from roaming neural

crest cells in a dish. Finally, the mechanics of the model will be analyzed in order to collect an understanding of its decision processes, which will be compared to the known mechanics of the simulator.

2 Graph Neural Networks

The relationships between independent objects and their interaction is particularly well represented as graphs. Indeed each object can be abstracted as a vertex or vertex that has a particular set of parameters. The connections between vertices can be represented as edges, to which are attached other parameters, such as the distance between the two vertices.

In a more formal definition, a graph $G = (V, E)$ contains n different vertices $V \in 1, ..., n$ linked by the edges $E \in V \times V$. The i^{th} vertex is represented by the vector $h_i \in R^d$.

The idea behind GNN is thus to share the internal state of a vertex with its direct neighbors, using the edges to identify them. The specific information sent is usually called a message. Messages received by a vertex are then aggregated together with the internal state to produce a new internal state $h'_i \in R^{d'}$, expressed in eq. (2) [34]. The functions *combine* and *aggregate* are what often differentiate implementations.

$$h'_i = combine(h_i, aggregate(h_j | j \in N(i)\})) \tag{2}$$

For example, in a standard convolution operation of a GCN [35], the *combine* operator is a learnable matrix and the *aggregate* is a simple sum over the neighbors.

By strategically doing multiple rounds of messaging, one can spread the local information inside a vertex to potentially the whole graph, provided it is fully connected.

Similarly to standard neural networks, it has been shown that GNN with random initialization can approximate all functions on graphs [34].

This framework allows for varying input sizes and small models can handle interactions between numerous subjects, which is crucial in simulations where the exact layout is not known in advance and can potentially be extensive.

While GNN can be used to compute high level data from the whole graph, this thesis will be focusing on using the internal vertices of communications as stand ins for real material exchanges or active forces and the output will reflect the update between the inputs and the next inputs,

over the span of a time step.

3 Architecture

For each time step, the model produces the most likely rate of change $\dot{x}$ of each variable x as well as the uncertainty associated to that prediction expressed in the form of the scale of the normal (or Laplace) distribution. Let the parametric functions $\mu(m, \theta)$ and $\sigma(m, \theta)$ (resp. $b(m, \theta)$) represent the neural network of parameters θ and input m, the distribution $p(\dot{n}|m, \theta)$ of the rate $\dot{n}$ is expressed by eq. (3) (resp. eq. (4)).

The particular choice of which distribution to use has been investigated with the upcoming comparison of hyper-parameters. The reason for the limited scope of this comparison falls under an additional assumption on the mechanics of the systems studied.

$$p(\dot{n}|m, \theta) = \mathbb{N}(\dot{n}; \mu(m, \theta), \sigma(m, \theta)) \tag{3}$$

$$p(\dot{n}|m, \theta) = Laplace(\dot{n}; \mu(m, \theta), b(m, \theta)) \tag{4}$$

Once the values are predicted, they are added back to the previous time step to produce the next input. To avoid unbounded computational power use, the longest spans of input that are covered in the prediction is *horizon*. This value is arbitrary but has a large influence on the performances of the model, this will be discussed in the experiment section. The cycle is depicted on fig. 4. To start this cycle, only the initial graph is given, the model takes charge of incrementally lengthening its input with its previous outputs. It stops when the maximal value *duration* reached.

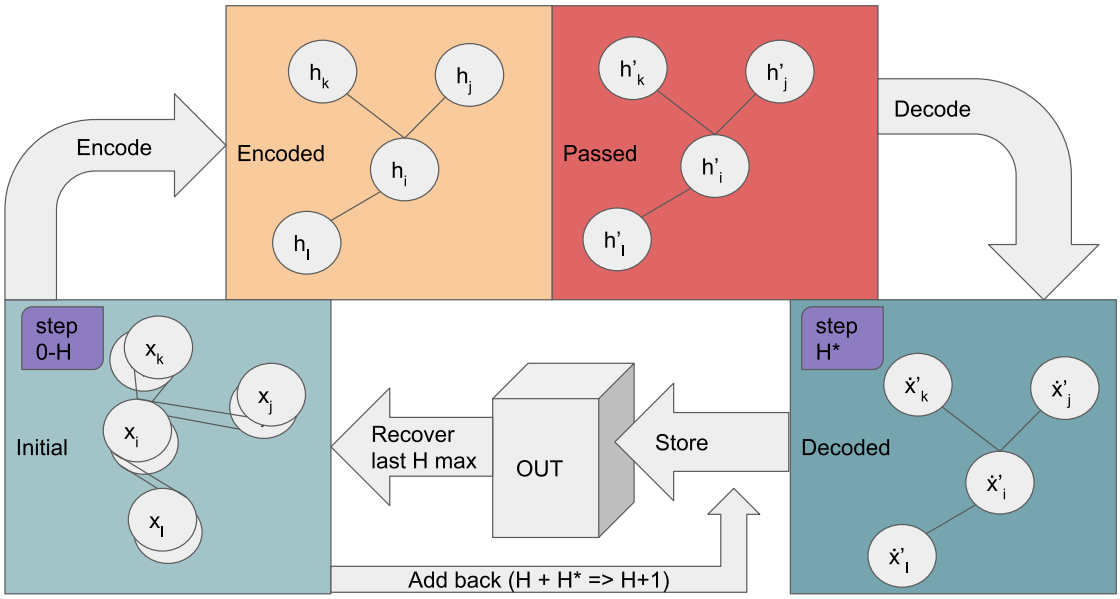


Figure 4: Cycle of the data for prediction in the training phase

The measure of the inaccuracy of the predictions during training, the loss, is estimated at a vertex level by the negative log likelihood of the displacements given the original positions $-log(\dot{n}|m, \theta)$ - or other parameters to predict - with normal or laplace uncertainty depending on the experiment.

The performances of the models are estimated using graph-level tasks, which will be discussed further in the experiments part.

3.1 GNN considerations

The time complexity of GNNs vary depending on their internal representation, usually it is $O(n^2)$, n being the number of vertices. If the matrix is sparse, the complexity pivots to $O(m)$, m being the number of edges. In rare cases, the pooling method can generate additional costs and, for example, when doing vertex pair-wise shortest path computation, the total is brought to $O(n^3)$ [36].

For the purpose of this research, there already can be a pre-selection of the GNN model given that two features emerge from the application. Firstly, the goal is to be able to handle a great number of vertices, since each represent a cell; the collective behavior can be better studied with

more subjects. Secondly, given the physical limitations of the environment in which the cells evolve, the range of direct communication is quite limited. This leads to the number of edges being quite low. The determination of a suitable model will thus be restricted to those having a sparse matrix representation, ensuring a time complexity of $O(m)$.

Due to computational restrictions, in most real cases, it is untractable to produce a fully connected graph. The way in which this problem is overcome is that the neighborhood of a particular cell is restricted to the ten closest, no matter the distance. This graph is known as a knn-graph, where the k stands in for the number of neighbors. This restriction is not prohibitive to learning because it is possible to iteratively send messages through edges, implicitly extending the number of connections, granted a vertex is capable of integrating and passing on the messages without compromising its internal state.

The construction of the connections can also be synthesized using a GNN [37]. However, since the structure can directly corresponds to physical proximity [38, 39] and that the number of neighbors rarely surpasses the tens, it is not necessary to over complexify the model.

The reason there is usually not a lot of connections is that, in a 2D space, the vertex coordination is exactly 6 for the dual graph of a Voronoi construction [40, page 67], which can be used to model biological systems [41], illustration on fig. 5 [42]. Even when considering other geometries and layouts, such as disk packing, the average degree in a maximal planar graph like an Apollonian network is close to 6 as well[43, page 5] [44, page 3]. Therefore, 10 seems to be a fitting upper bound.

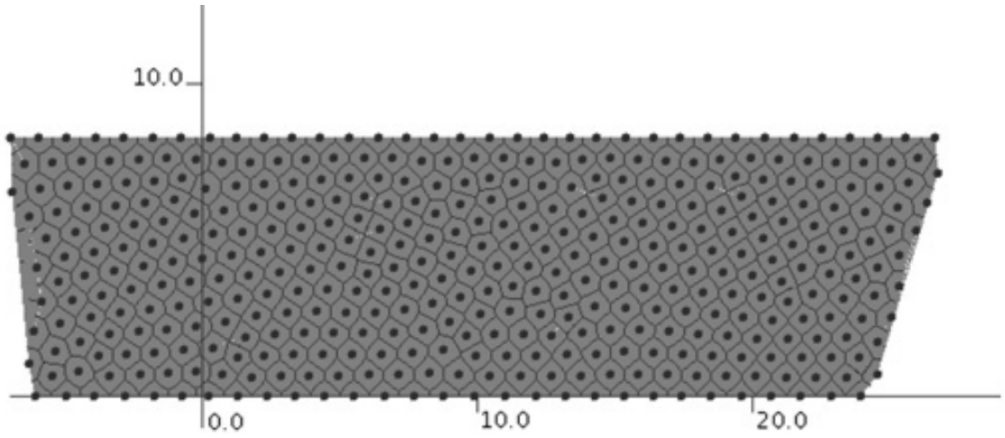


Figure 5: Voronoi area of cells under compression

That upper bound leads to a time complexity of $O(m) = O(10n)$, justifying the earlier choice of restricting the search in the possible GNN models to those having a sparse representation.

After reviewing the different contestants for a GNN architecture, GATv2 [45] seemed a fitting choice.

The main advantage of using GATv2 instead of a regular graph convolutional network is that the model can choose how much influence the neighbors will have on its internal update dynamically. Two extreme scenarios can be thought of. In the first, the cell is being governed entirely by a group behavior, thus setting its looping connection to a low value and the others to a high value. In the second, a cell is isolated from the others and focuses on itself almost exclusively. It is also advertised as having better results than GCN, GraphSage [46] and GIN [47] - when attention is needed and the interactions are complex - in its introductory paper, though it is explicitly mentioned that it is usually impossible to determine in advance which would be best in a particular setting [45].

The precise complexity of the interaction is relatively low in synthetic data, but in real cases it may be great enough to justify employing a framework with high expressiveness.

Introducing attention in graph networks also has the benefit to improve the signal to noise ratio and reduce the graph complexity. Indeed, attention permits the avoidance of particularly noisy regions [48].

It is notable that using a graph method in general improves interpretability, and adding attention to that approach increases it even more

[48].

The core mechanisms can be explained in the three main eqs. (5) to (7). The message to be sent $e(h_i, h_j)$ from edge i to j, having the internal state h_i and h_j respectively, depends on the attributes of the edges $e_{i,j}$ and the internal states by the intermediary of the attention given the eqs. (5) and (6), [45]. In these equations, the parameters of the model $a \in R^{2d'}$ and $W \in R^{d' \times d}$ are the ones being learned, and $[\cdot||\cdot]$ denotes the concatenation operator. Finally the new state of cell h_i is updated to h'_i in eq. (7) using every cell h_j in the neighborhood of h_i, N_i, mitigated by the attention.

$$e(h_i, h_j) = a^T LeakyRelu(W \cdot [h_i||h_j||e_{i,j}]) \tag{5}$$

$$\alpha_{ij} = softmax(e(h_i, h_j)) \tag{6}$$

$$h'_i = \sigma(\sum_{j \in N_i} \alpha_{ij} \cdot W h_j) \tag{7}$$

The previous update is done for each head of the GATv2, and, since the authors recommended to have numerous heads, this number has been set to be 8.

This means that out of the 10 different maximum neighbors, a cell will choose 8 times how it divides its attention between them. The resulting 8 states will then be averaged then passed on to the next GATv2 as input which will regroup them however it sees fit to produce again 8 states. The final GATv2 will produce its output and average them back in a single state.

The types of graph that will be used in the following parts are undirected, homogeneous and dynamic. Undirected meaning the attributes of an edge from a vertex A to a vertex B is not necessarily the same as the edge in the opposite direction, which can also not exist. Homogeneous in the sense that every vertex is of the same type, and every edge also. Lastly dynamic indicates that the structure is able to evolve over time.

The dynamic component in particular is implemented by recomputing the edges and their attributes in between two steps of time. This means it is possible to connect an additional network, that would have the purpose of describing, for example, cellular division and death. This is beyond what was studied for this thesis, and would require additional work to neatly fit with the fixed tensors *pytorch* utilizes.

3.2 Synthetic data model

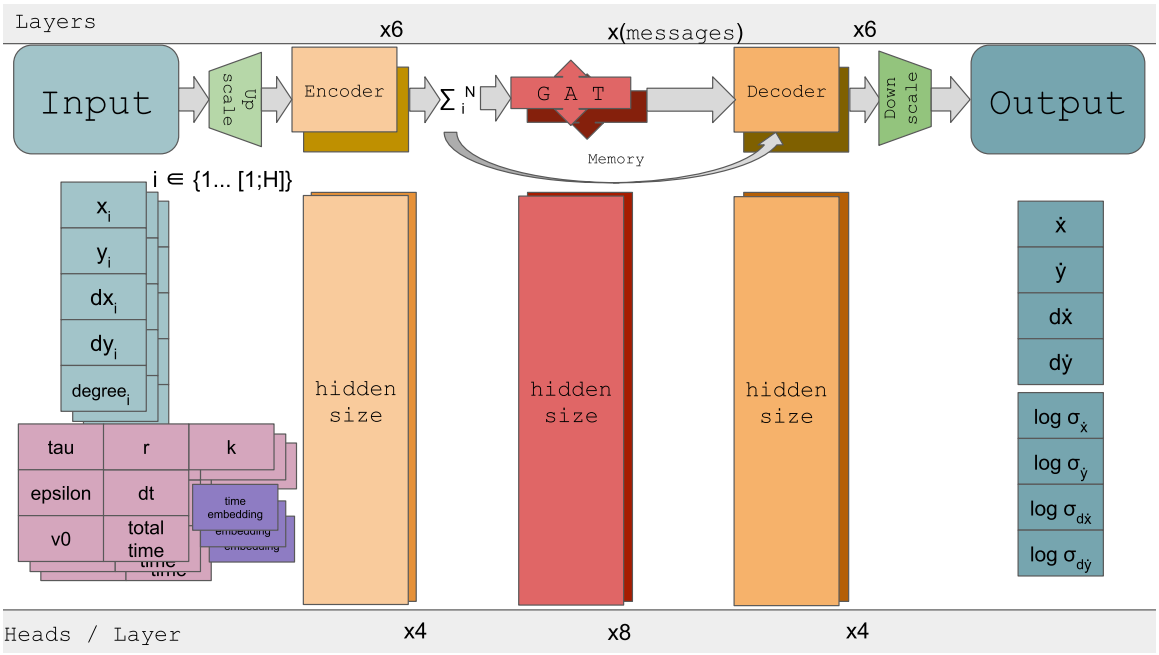


Figure 6: Architecture of the GNN for synthesized data

The main idea behind the architecture of the model is that values at a vertex level get encoded in a latent space of a certain size. This will form the message that is passed on to neighboring vertices. Since the graph is constructed with self referring edges, the content of a vertex is updated together with all the vertices connected in one step. This operation is repeated "messages" number of times by every vertex. Once all the information has been shared, the information contained in the vertex is decoded in the desired format.

For a particular position $\{m\}$ I want to predict the next position $\{n\}$. To do this I will predict the displacement between the two time steps $\{\dot{n}\}$ and add it to $\{m\}$ to have $\{m + \dot{n}\} = \{n\}$.

The value of the speed from the simulator was not utilized because it would be too accurate compared to real data since it comes directly from the inner states of the simulator. Indeed, in a real scenario, the only values accessible are the positions and, from them can be computed the one time step speeds along each axis, denoted dx and dy.

A few more parameters such as τ an inversely proportional measure of

stochasticity, the adhesion forces in cell interactions ϵ, the average radius of all cells r, the time between two steps dt, the length of the experiment $totaltime$, and the stiffness between pairs k have been added to be able to differentiate environments.

The fig. 6 summarizes the synthetic data architecture in a drawing.

3.3 Real data model

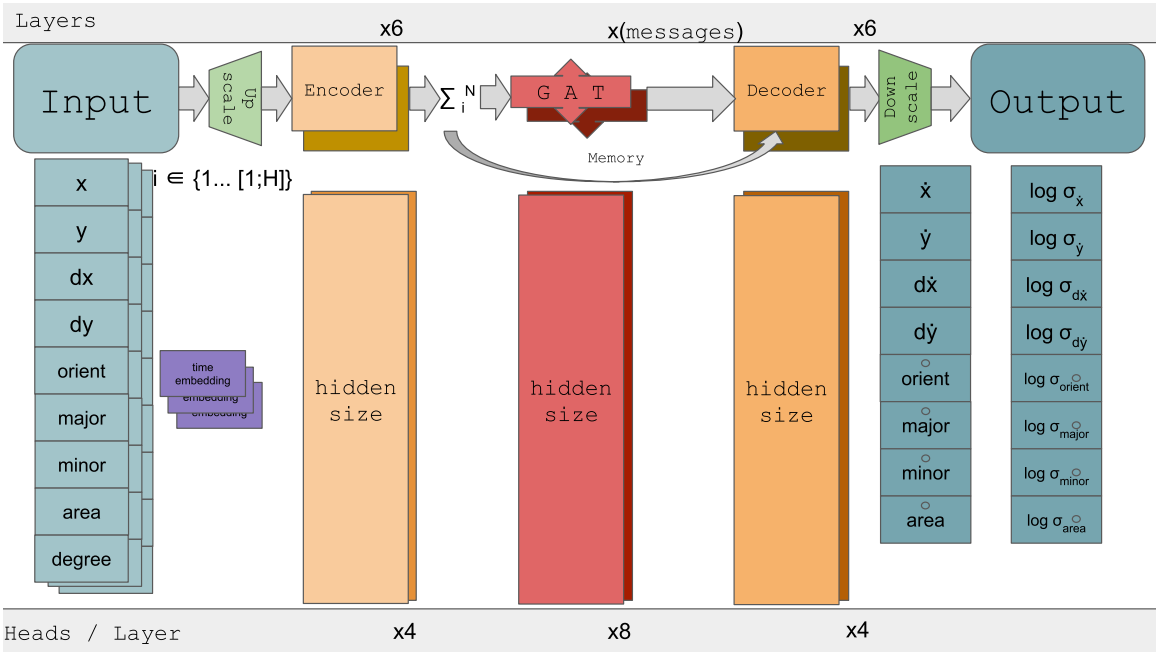


Figure 7: Architecture of the GNN for real data

The only difference between this model and the one used for the synthesized data is that the attributes that can be accessed and those that are predicted can be different.

The positions, similarly to the previous case, are used to compute the one time step speeds. Additionally, the bayesian object tracker used to isolate the cells in a video outputs the orientation of the cell, the length of its minor and major axis and the area it occupies. While those are not strictly necessary, there is no downside to adding them, and it will provide more ways to compare the generated values against.

It is to be noted that the orientation being cyclic implies the loss has to incorporate a cyclic function for this field in particular. For this, the

difference $\Delta\dot{\theta}$ is being recomputed as in eq. (8). A linear term has been added to disfavor unlimited growth of the actual value $\dot{\theta}$ predicted.

$$\Delta\dot{\theta} \leftarrow sin(mod(|\Delta\dot{\theta}|, \pi)) + 0.01 * |\Delta\dot{\theta}| \tag{8}$$

The environmental descriptor that gave out the particular settings of the simulation are no longer accessible. In the case of this thesis, every video that was used features the same type of cells left to roam freely in a dish; there should be no need for additional parameters.

For each attribute is computed the mean and variance.

The fig. 6 summarizes the real data architecture in a drawing.

3.4 Encoding and Decoding

The first step in the processing of the inputs is to realize that providing them directly to be passed on as messages would impair the capabilities of the model. Indeed, some parts are possibly not relevant or redundant for the production of the output. Others could be more efficiently utilized by another variable representing them together.

It is then natural to encode the features of an input X with a dimension of d_X such that $X \in R^{d_X}$ into the latent space Z with dimension d_Z using a deterministic function $f : X \to Z$. Such a function, as any other, can be represented by a sufficiently large and deep multi-layer perceptron (MLP).

Once the sharing of messages has been done by the GNN, the process can be reversed with another function $f' : Z \to Y$, represented by another MLP.

To add the possibility of having a sequence of inputs $X_1, X_2, ..., X_H$ of the length horizon H, a sequence-to-sequence model can be introduced in place of the encoder MLP. While recurrent neural networks (RNN) like LSTMs [49] and GRUs [50] have been used in the past, the trend has shifted with the apparition of transformers [51]. Transformers have the advantage of leveraging GPU power to simultaneously process a great number of inputs. Indeed, RNN have the downside of successively iterating on each part of the sequence to form an internal representation of it, which is usually much slower. Their computational complexity also is

quadratic in the size of their hidden dimension, compared to linear for transformers. Their architecture as imagined in the original paper [51] can be seen on fig. 8.

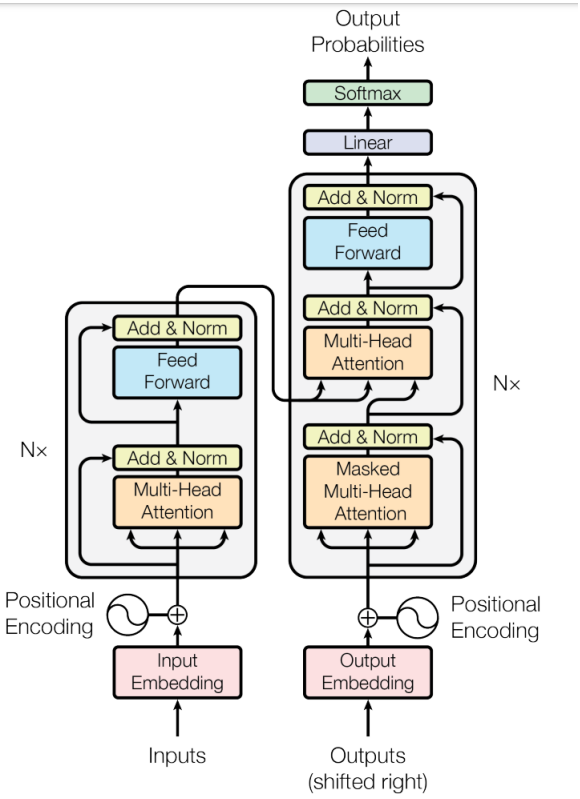


Figure 8: The transformer architecture [51]

The main mechanism behind transformers is self-attention. It produces an output for each head given three inputs for a sequence length n and hidden dimension d : $Q, K, V \in R^{n \times d}$, according to the eq. (9) [52]. This can be seen in the fig. 9. Each output is then concatenated or aggregated.

$$O = softmax(QK^T)V \tag{9}$$

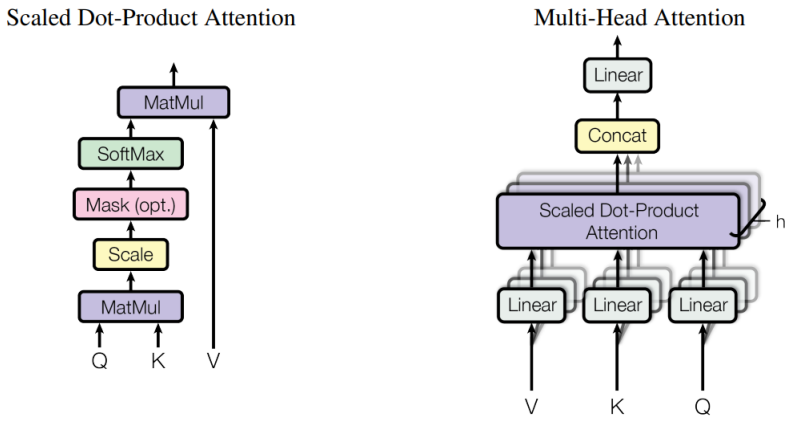


Figure 9: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [51]

Transformer still do suffer from a feature of its main mechanism : attention. Each part of the input sequence is used with every other to produce attention metrics, which creates a quadratic complexity in the length of the sequence. (Note : this is addressed by the dilated attention introduced recently [52], having linear complexity) Thankfully, the horizon, which corresponds to the length of the sequence, is expected to be very limited.

Another advantage of using transformers is that they naturally leverage the encoded information prior to message passing and uses it in conjunction with the updated state to produce the desired output. This is more of a byproduct of the pytorch implementation that was used, but is still most welcome.

Lastly, it is important to note that there needs to be a way for the transformer to know the positions of the inputs in the sequence. For this, a positional encoding is created using a variable named "time embedding". Simply defined as $t_i = \frac{p_i}{length}$, where p_i is the position in the sequence of the i^{th} member, ranging from 1 to $length$.

For the purpose of this thesis, only the encoder and decoder parts of the transformer are used. They correspond to the two main blocks of fig. 8. The depth has been fixed to $N = 6$ arbitrarily, these were the number used in the original paper [51].

The dimensionality of the inputs and outputs has been changed from the original $d_{model} = 512$ to $d = 256$ for the final results, though this

particular number will be appearing in the different tunings as "size of messages". To avoid having a multitude of hyper-parameters, the messages exchanged and all internal dimensions share the same value, the aforementioned "size of messages".

The number of heads has been decreased from $h = 8$ to 4. This is justified by the fact that the horizon value, which determine the length of the sequence, is usually very low. And a small sequence should not require as many heads.

The input and output corresponding directly to the data do not have a size of 256 "as is". This is the reason behind the introduction of simple MLPs preceding the encoder and following the decoder; one up-scales the size from the standard input to the desired number, the other down-scales it from 256 to the number of output channels, being one for each quantity, doubled to include the uncertainty measure.

3.5 Assumptions

A few assumptions are made in this prediction. The first being that the system behaves as a Markov's chain of order one, meaning an assumption is that the displacement can be predicted with only the previous position and the previous displacement. Or more formally, given $A_i := \{m_i, \dot{n}_i\}$, we have the eq. (10).

$$P(A_{i+1}|A_i) = P(A_{i+1}|A_0, A_1, ..., A_i) \tag{10}$$

The length of time over which we aggregate past results expands this relation to include not one but H - the horizon - previous states, as can be seen in eq. (11). This remains a Markov's chain. Indeed, one can aggregate H states $A_{i-H}, A_{i-H+1}, ..., A_i$ in a macro state $M_i = \bigcup_{h=0}^{5} A_{i-h}$ and obtain a formula analogous to eq. (10) in the form of eq. (12).

$$P(A_{i+1}|A_{i-H}, A_{i-H+1}, ..., A_i) = P(A_{i+1}|A_0, A_1, ..., A_i) \tag{11}$$

$$P(M_{i+1}|M_i) = P(M_{i+1}|M_0, M_1, ..., M_i) \tag{12}$$

The second assumption made is that the distribution of the conditional displacements in a given direction $p(\dot{n}_x)$, where X is ensemble of

values the position in the axis x can take, can be represented as a normal (or laplace) distribution using the mean (or location) and the standard deviation (or diversity).

By construction, the variables $\{n_x\}$ and $\{n_y\}$, and more precisely $\{\dot{n}_x\}$ and $\{\dot{n}_y\}$, are drawn independently from one another. Meaning they can only represent $p(\dot{n})$ as a sum of independently drawn variables of their distribution.

This leads to the vector $\{\dot{n}\}$ following the distribution with $\mu_{\dot{n}} = \mu_{\dot{n}_x} + \mu_{\dot{n}_y}$ and $\sigma^2_{\dot{n}} = \sigma^2_{\dot{n}_x} + \sigma^2_{\dot{n}_y} + 2\rho\sigma_{\dot{n}_x}\sigma_{\dot{n}_y}$.

The third major assumption comes from the assumed distributions themselves. Indeed restricting the probability space to a particular distribution imposes many additional implicit assumptions.

The choice of the normal distribution came from the fact that it is rather common in many problems, such as measurement errors [53]([54] presents the use of a scaled mixture of skewed-normal) which can be somewhat analogous to its use in this thesis. To go even further, the normal distribution is also the distribution of "orthogonal velocity components of particles moving freely in a vacuum" [55, page 5]. Granted cells are not particles and are not moving freely, it is still a path that needed to be investigated.

On the other hand, it felt needed to have another reference distribution to compare and contrast to. For this, the Laplace distribution, another giant in statistics, was chosen. The problem leading its creation was stated as "Déterminer le milieu que l'on doit prendre entre trois observations données d'un même phénomene.", which is close to the likelihood aimed to be computed here.

Future work might be interested in looking at other distributions, mainly multivariate and/or skewed, to better approximate the true hidden underlying mechanics.

3.6 Many to one

The reason for the use of a sequence only for the input in the encoder is that, to transmit the messages, the edges in the graph have to be computed. One could use the same graph disposition for one full *horizon* prediction but the flexibility of the graph representation would then be

lost.

Indeed, the architecture used performs as in algorithm 1, while the other would act as in algorithm 2. Since it is possible to generate a graph only on X or O, there would be an offset of *horizon* between the actual state of the graph and the one used in the computation.

Algorithm 1 Many to one behavior, X input, Y internal state, O output

$[X_{t-H}, ...X_t]$ = Sample With Horizon(t, H)
G = Propagation Graph(X_t)
Y_t = Encode($[X_{t-H}, ...X_t]$)
Y_{t+1} = Propagate(Y_t, G)
O_{t+1} = Decode(Y_{t+1}, Y_t)

Algorithm 2 Many to Many behavior, X input, Y internal state, O output

$[X_{t-H}, ...X_t]$ = Sample With Horizon(t, H)
G = Propagation Graph(X_t)
$[Y_{t-H}, ...Y_t]$ = Encode($[X_{t-H}, ...X_t]$)
$[Y_{t+1}, ...Y_{t+H+1}]$ = Propagate($[Y_{t-H}, ...Y_t]$, G)
$[O_{t+1}, ...O_{t+H+1}]$ = Decode($[Y_{t+1}, ...Y_{t+H+1}]$, $[Y_{t-H}, ...Y_t]$)

This method is also remarkably slower - especially in testing -, and did not bring any benefit in performance, see in the Appendix fig. 35.

4 Experiments

For each data set available, a tuning of the hyper-parameters of the models' architecture has been performed to provide the best results possible. These tests and comparisons also allows for a deeper understanding of the importance of each parameters.

Once the parameters have been found, an in-depth analysis of different statistics summarizing the development and activity of the groups of cells under scrutiny has been made. This will be used to compare how accurately the simulation made by the model follows the expected behavior.

Finally, the decision processes of the model have been studied by exploiting the explainability of GNNs.

4.1 Data-sets

4.1.1 Synthetic data

Allium is a simulator created specifically for modelling collective cell migration using a minimal number of bio-physically inspired parameters by Dr. Namid Stillman and Prof. Silke Henkes. It creates trajectories of active Brownian particles subjected to forces whilst in proximity to one another [56].

For the tests realized in this thesis, two parameters are of particular interest. v_0 or v the propulsion force of a particle and the persistence timescale τ. High values of v will lead to velocities being higher. Lower τ will decrease the time between changes in direction, leading to more randomness in their trajectory.

Four data sets have been produced using these two variables : high tau (ht) and high velocity (hv), high tau and low velocity ($ht\ lv$), low tau and high velocity ($lt\ hv$) and finally low tau and low velocity ($lt\ lv$). It is expected that the more predictable trajectories $ht\ hv$, which are governed almost entirely by their propulsion force and interactions, will yield to better results. On the opposite hand, $lt\ lv$, highly unpredictable and inconsistent, is expected to be the hardest to fit.

The particular equations used to update the position $\mathbf{r}$ are defined in eqs. (13) and (14), where η is the friction in the system and n is the

normal to the particle. Also appear k, the stiffness of the interaction, r the distance between two particles of respective sizes R_i and R_j. Finally, there is the dimensionless parameter ϵ.

$$\dot{\mathbf{r}}_i = v_0 \hat{\mathbf{n}}_i + \frac{1}{\eta} \sum_j \mathbf{F}_{ij} \tag{13}$$

$$F_{ij}(r) = \begin{cases} k(r - b_{ij}) & if \ \frac{r}{R_i + R_j} < 1 + \epsilon \\ -k(r - bij - 2\epsilon b_{ij}) & if \ 1 + \epsilon < \frac{r}{R_i + R_j} < 1 + 2\epsilon \\ 0 & otherwise \end{cases} \tag{14}$$

4.1.2 Real data

In this setting, neural crest cells are left to roam and evolve freely in a dish. The videos were selected for their low rate of division and death, and were collected in the lab of Prof. Roberto Mayor and his postdoc Dr. Adam Shellard [57]. A gallery of images from a video automatically annotated can be seen on fig. 10 and the corresponding tracked cells on fig. 11.

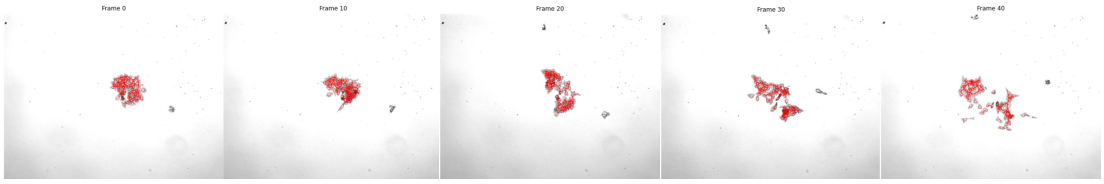


Figure 10: Gallery of computer annotated images from a dataset video : Data3-25.avi

Figure 11: Gallery of reconstructed cells from a dataset video : Data3-25.avi

Once the positions were annotated, a bayesian tracker [58], [59] identified individuals cells and their positions over time. These tracks can be seen on fig. 12. The codes for this and data were provided by Dr. Namid Stillman.

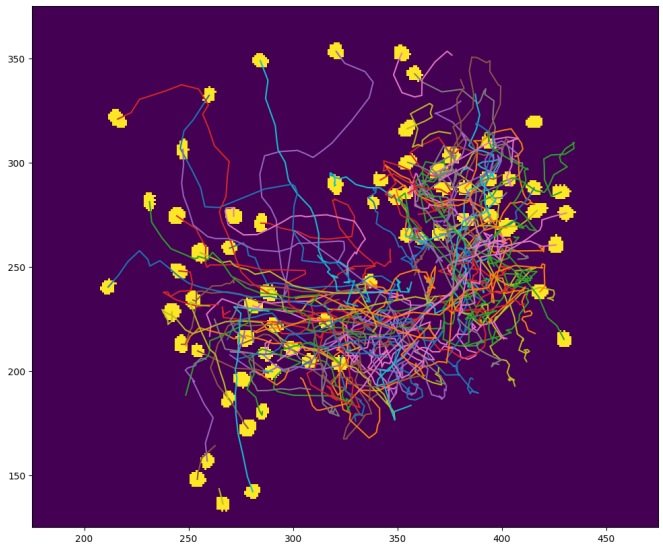


Figure 12: Tracks of all cells in Data3-25

It is to be noted that such a method contains inherent uncertainty and mistakes that are not always possible to automatically filled. The table 1 contains the different statistics of the elements that were retrieved from each video. The filling rate corresponds to $1 - sparsity$ of the matrix of size $[0 : max(time), tracks, attributes]$ containing all tracks where the first apparition of the track is considered as a missing value because the speed cannot be correctly evaluated.

Video	T_l	$T_{l,g}$	$Filling$
V. 3-12	19	17	0.816
V. 3-18	296	224	0.199
V. 3-25	109	81	0.485
V. 3-72	81	68	0.590

Table 1: Table of statistics for real data.
T_l : the number of tracks longer than four time steps.
$T_{l,g}$: T_l with fewer than two gaps in a row.
$Filling$: the filling rate of $T_{l,g}$

4.2 Protocol

4.2.1 Training and testing

For each cell-specific value, except for the degree of the vertex, a distribution is predicted by the neural network. Since two values are produced, both normal and Laplace distributions can be used. They will be compared in the experiments.

During the training phase, only the location μ is used. The scale σ (or b for laplace) acts as a confidence metric to measure how much the mean squared error is important; the more the model trains, the closer this value becomes to the aleatoric uncertainty of the data.

To update these values, the loss of eqs. (15) and (16) are used. In the real dataset, missing values can appear. To avoid fitting on uninitialized data, a mask M composed of ones (present) and zeros (missing) is passed on as well. The loss is simply updated to be $L_{masked} = L \odot M$, where $\odot$ represents the element-wise or Hadamard product.

$$L_{Normal} = \frac{\left(\mu(m,\theta) - \dot{n}_{true}\right)^2}{2\sigma^2(m,\theta)} + log\ \sigma(m,\theta) \tag{15}$$

$$L_{Laplace} = \frac{|\mu(m,\theta) - \dot{n}_{true}|}{b(m,\theta)} + log\ b(m,\theta) \tag{16}$$

The algorithm to compute the update of the training phase is defined in algorithm 3.

To smooth out the beginning of the training, curriculum learning is used. It will gradually increment the horizon of prediction. The start needs only the model to predict the immediate next step. At every epoch, this duration will increase until arriving at the threshold, being 8 for synthetic data and 4 for real. From which point on it will stay constant.

Since there is no need for the entire length of the data at once, a random valid starting position is chosen and the update will only take this part into account. In the case of the real data set, only positions in which every detected cell is being detected for the entire length are

considered valid. This is the reason why the maximum predicted length is shorter, it will lead to more valid starting points.

This first data point is then updated as being the output to iteratively produce further time steps. The graph needs to be updated to reflect the disposition of the newly created entry; the edges, edges attributes and degree of each vertex are recomputed. In the case a prediction for the speed of the cells is not produced, this reconstruction also provides the updated immediate velocity.

At every prediction, only the maximum likelihood estimation is added back to the attributes of the cell during training (including the regular testings that creates the loss estimation).

Algorithm 3 Training

```
while e in epochs do
    length = CurriculumLength(e)
    while d in data do
        data_start_random = RandomSample(d, length)
        current_step = FirstData(data_start_random)
        produced = {}
        while l in length do
            produced ← MaxLikelihood(Simulate(current_step))
            current_step = current_step + LastOf(produced)
            ReconstructGraph(current_step)
        loss = ComputeLoss(produced, data_start_random)
        UpdateModel(loss)
```

In the evaluation phase, when the training and regular testing are concluded, the algorithm is slightly modified to algorithm 4. In this version, the entire distribution is sampled from with *Draw* rather than taking the maximum likelihood. This allows for the creation of a unique plausible trajectory which is embedded with the aleatoric uncertainty.

Algorithm 4 Evaluation

```
while e in epochs do
    length = CurriculumLength(e)
    while d in data do
        data_start_random = RandomSample(d, length)
        current_step = FirstData(data_start_random)
        produced = {}
        while l in length do
            produced ← Draw(Simulate(current_step))
            current_step = current_step + LastOf(produced)
            ReconstructGraph(current_step)
        loss = ComputeLoss(produced, data_start_random)
        UpdateModel(loss)
```

The update of the model is performed via gradient descent, which uses automatic differentiation. The neural network is represented as a set of weights. When it performs a guess it propagates forward the input by multiplying and/or adding it with its internal weights. The prediction is then given by the last values computed.

The derivative of the loss - of the error made with respect to the data sample - is computed with respect to each parameter of the neural network. Depending on the optimized used, a portion of these derivatives are then subtracted to perform a gradient descent.

The particular optimizer used in this thesis is AdamP [60]. The weight update is performed as in eq. (17) with the learning rate η. Depending on its value - chosen in eq. (20), either a simple momentum update as in eq. (18) is performed, or a projection as defined in eq. (19). $cos(a, b) := \frac{|a^T b|}{||a||\ ||b||}$ being the cosine similarity, $\delta = 0.1$ a parameter, the l2-normalized vector $\hat{w} = \frac{w}{||w||_2}$.

$$w_{t+1} = w_t - \eta q_t \tag{17}$$

$$p_t = \beta p_{t-1} + \nabla_{w_t} f(w_t) \tag{18}$$

$$\Pi_{w_t}(x) := x - (\hat{w}\dot{x})\hat{w} \tag{19}$$

$$q_t = \begin{cases} \Pi_{w_t}(p_t) & if\ cos(w_t, \nabla_w f(w_t)) < \frac{\delta}{dim(w)} \\ p_t & otherwise \end{cases} \tag{20}$$

This choice is motivated by the better results advertised in the introductory paper [60]. Indeed, they claim that the traditional momentum update suffers from rapid decrease in effective step size; the step size in the in the "actual" space of search S^{d-1}. So they scale the step size from the state of search R^d to S^{d-1}, preventing this early fall-off and improper convergence.

All the experiments were performed on the Alan GPU Cluster. For the synthetic data-set, training lasted 50 epochs of 900 samples on a single 12GB GPU amounting to around 8h. The real data-set training lasted 80 000 epochs of 3 samples, amounting for 15h on a single 12GB GPU. After every one (resp. 50) epoch, the model was tested against the 50 (resp. one) sample test set to produce a test loss curve.

4.2.2 Evaluation

Before any training has occurred in supervised learning, when the amount of data allows it, the training samples are divided in three parts. The training, testing and validation set. On the first the model will be fitted in training as the name suggests, the iterative improvements and selections of free parameters are performed using the testing set and the validation set is only at the very end to have an unbiased estimate of the true performances. This is the manner in which the synthetic data has been treated.

When the quantity of data gets too small, we can use the leave one out method. One singular data sample is isolated from the other before training and left to be the test sample. New models are successively trained for each left out sample. The performances of the methods used then have to be inferred from the ensemble of models created. Since

only four samples were available for the real data, this method was the preferred choice.

When the training is concluded, four different summary statistics will help in determining the performances of the model. Each of those being commonly used for active systems like cells (Vast use of MSD in tracking living cells [61], Velocity analysis [62, page 50]).

- Mean square displacement

  The matrix of positions *pos* is organised as a matrix of dimensions (T, N, A), respectively the time length T, the number of cells N and the number of axes A.

  The equation to determine the value of MSD at a time i is eq. (21).

$$msd_i = \frac{\sum_{t=0}^{T-i} \sum_{cell=0}^{N} \sum_{a \in axes} (pos[i+t, cell, a] - pos[t, cell, a])^2}{N * (T - i)} \tag{21}$$

  The measure is particularly useful to determine the nature of the movement. In the logarithmic scale, a linear MSD indicates, in average, a ballistic motion; the trajectory of the group or its expansion will appear as if no other force acted upon it, and it will simply continue unperturbed.

- Mean velocity

  The matrix of speeds is arranged in the same manner as for the positions. The equation is thus given in eq. (22), and gives for a particular point in time i the L2-norm of the speeds.

$$mv_i = \frac{\sqrt{\left(\sum_{cell=0}^{N} \sum_{a \in axes} (speed[i, cell, a])^2\right)}}{N} \tag{22}$$

- Scaled velocity component distribution

The SVCD is the L2-norm of the normalized histograms for each axes. Defining the function in as in eq. (23) and an interval of values $[a, b]$ subdivided in s partitions of length l leads to the introduction of the formula for the SVCD as eq. (25).

$$in(s, v) = \begin{cases} 1, v \in [a + s * l, a + (s + 1) * l] \\ 0, otherwise \end{cases} \tag{23}$$

$$mean\_speed = \frac{\sum_{t=0}^{T} \sum_{cell=0}^{N} \sum_{a \in axes} speed[t, cell, a]}{T * N * axes} \tag{24}$$

$$svcd(s) = \sqrt{\sum_{a \in axes} in(s, \sum_{t=0}^{T} \sum_{cell=0}^{N} \frac{speed[t, cell, a]}{mean\_speed})^2} \tag{25}$$

The SVCD offers a more in depth look at the velocities in each axis. In the case of the synthetic data, a distinct spike is expected for the values chosen in the simulator. A symmetric SVCD indicates the center of mass stays in place on average, while an offset will reflect a shift over time.

- Scaled velocity magnitude distribution

  The SVMD is the normalized histogram of the L2-norm of the speed vector.

$$svmd(s) = \frac{\sum_{t=0}^{T} \sum_{cell=0}^{N} in(s, \frac{\sqrt{\sum_{a \in axes} speed[t,cell,a]^2}}{mean\_speed})}{T * N} \tag{26}$$

4.3 Result

4.3.1 Synthetic data - Tuning

Before analysing the results, all parameters need to be chosen. To avoid random guessing, a comparison of performances for one parameter was done, all other being equal.

All tested parameters and their values are summarized in table 2. The best results were achieved for the parameters in **bold red**, while **bold green** is a correction for visual quality over metric. Only one setting -*lt hv*- out of the four reported on -*ht hv*, *lt hv*, *lt lv*, *ht lv*- is used in the comparison to limit the number of graphics, which appear in the Appendix.

The distance between each curve with the reference curve has been summarized in tables 3 to 7.

Name	Base	Range
Setting	lt hv	lt hv
Distribution	laplace	**laplace**, **normal**
Horizon	5	**1**,2,3,4,5
Number of messages	4	1,**2**,3,4
Size of messages	128	32,64,128,**256**,512,1024
Number of channels	8	4, **8**

Table 2: Tuning range and base parameters

Curve	Laplace	**normal**
MSD	14597.07	**9124.14**
MV	**1.94**	2.24
SVCD	2.97	**2.87**
SVMD	11.61	**11.55**

Table 3: Distances for each metric in the tuning of the distribution

Curve	**1**	2	3	4	5
MSD	44793.36	32309.14	19003.61	17537.93	**14597.07**
MV	**1.39**	1.42	1.41	1.64	1.94
SVCD	**2.0**	2.26	2.63	2.87	2.97
SVMD	**10.24**	10.92	11.37	11.54	11.61

Table 4: Distances for each metric in the tuning of the horizon

Curve	1	**2**	3	4
MSD	**11128.26**	14855.8	12894.06	14597.07
MV	1.56	**1.46**	1.49	1.94
SVCD	3.0	**2.92**	3.01	2.97
SVMD	11.63	**11.58**	11.64	11.61

Table 5: Distances for each metric in the tuning of the number of messages

Curve	32	64	128	**256**	512	1024
MSD	**10202.08**	15076.61	14597.07	22606.02	17429.5	18326.32
MV	1.72	1.79	1.94	1.75	**1.6**	1.84
SVCD	2.99	3.02	2.97	**2.74**	2.88	2.86
SVMD	11.62	11.64	11.61	**11.46**	11.55	11.54

Table 6: Distances for each metric in the tuning of the size of messages

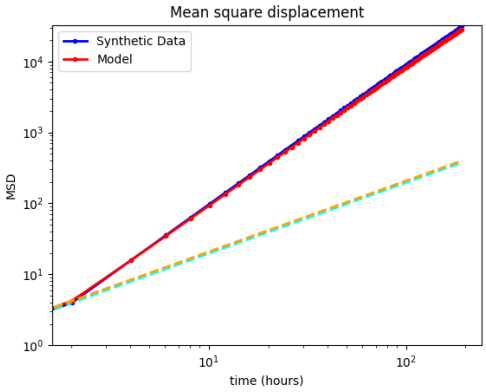
Curve	4 Channels	**8 Channels**
MSD	**1747.0**	14597.07
MV	**1.57**	1.94
SVCD	3.23	**2.97**
SVMD	11.71	**11.61**

Table 7: Distances for each metric in the tuning of number of channels

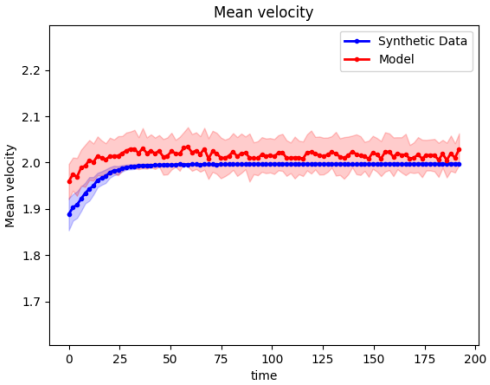
4.3.2 Synthetic data - Performances

All models under this section were trained using the tuned hyper-parameters for *lt hv*.

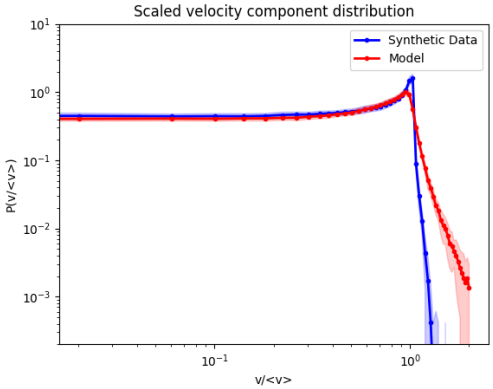
4.3.2.1 high tau, high velocity



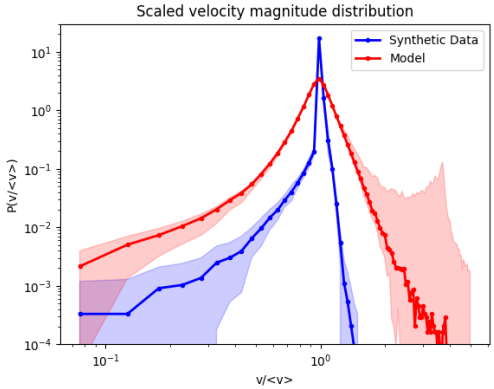
(a) Mean squared displacement



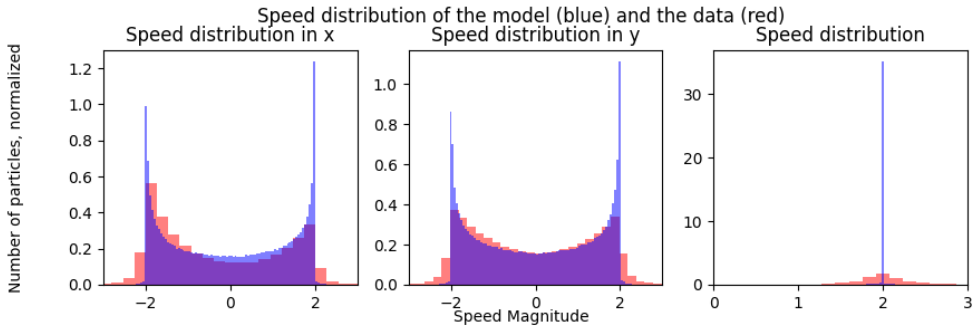
(b) Mean velocity



(c) Scaled velocity component distribution



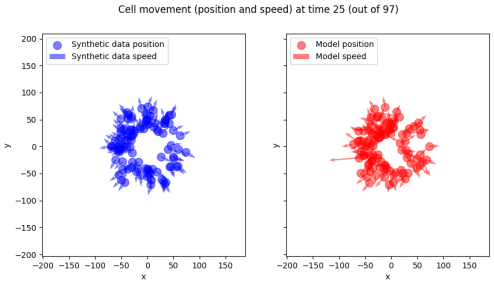
(d) Scaled velocity magnitude distribution



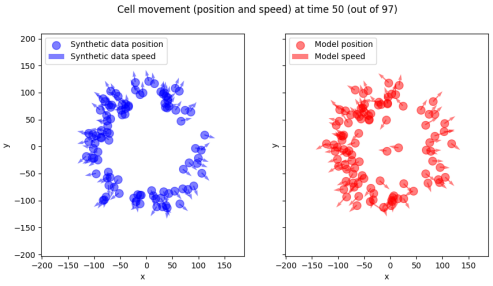
(e) Velocity distribution, component (x,y) and magnitude

Figure 13: Results in the validation data set high tau high velocity, tuned parameters

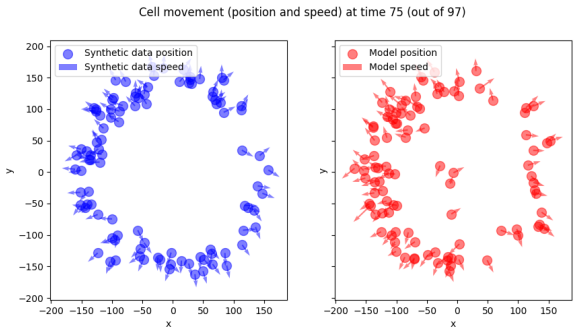
As expected, *ht hv* is the easiest setting. Even though the parameters have been optimized for *lt hv*, all summary statistics seem to be reproduced quite well.



(a) Frame 25



(b) Frame 50



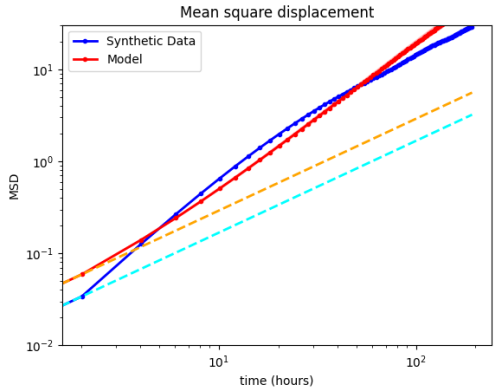
(c) Frame 75

Figure 14: Comparison of the trajectories for a sample in the validation data set high tau high velocity

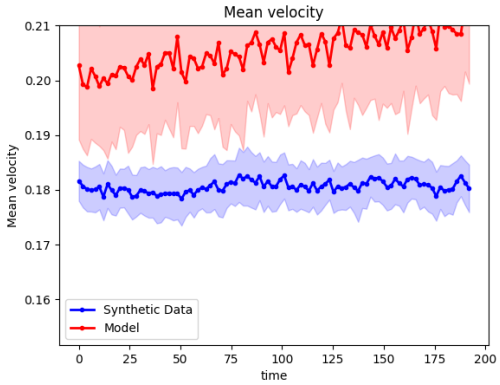
Interestingly enough, some cells seem to stay inside the circular pattern in figs. 14b and 14c.

It may be the case that this behavior often appears and the synthetic data sampled happened to not have it. It could also be that the model uses some particles as bridge to communicate to the other side, though this is just speculation.

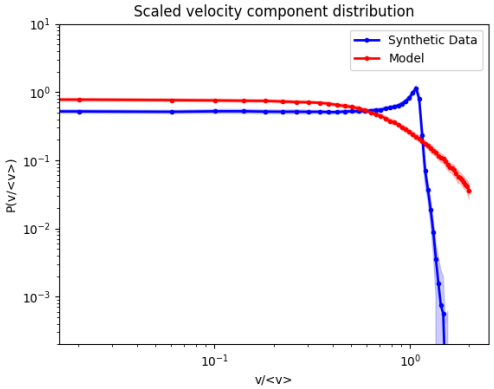
4.3.2.2 low tau, low velocity



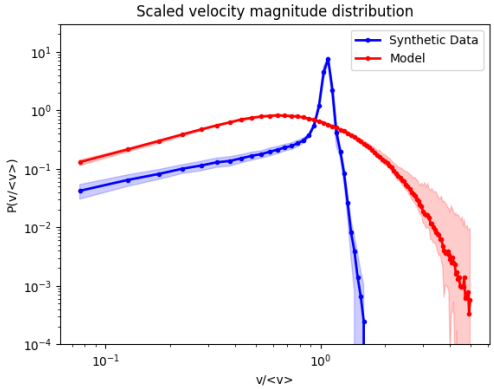
(a) Mean squared displacement



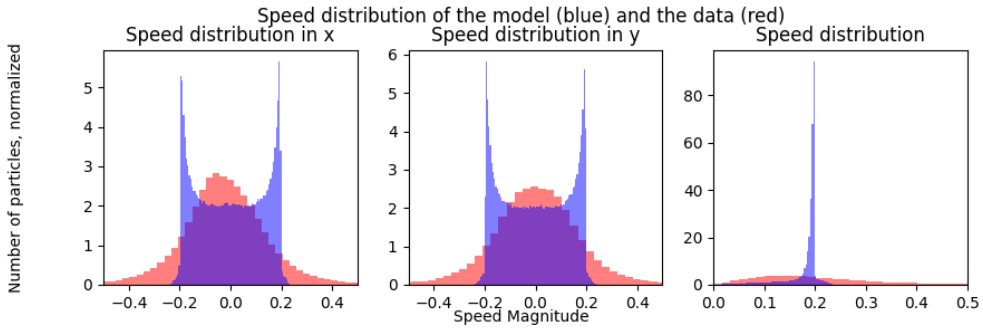
(b) Mean velocity



(c) Scaled velocity component distribution



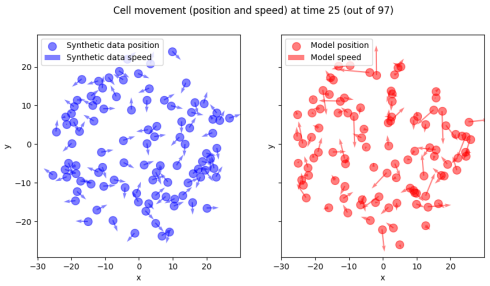
(d) Scaled velocity magnitude distribution



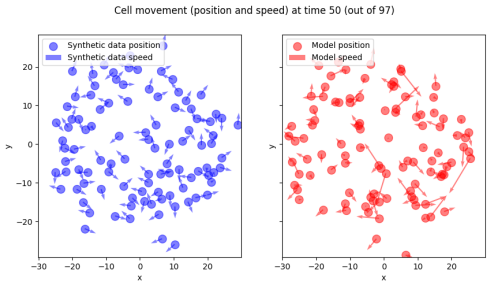
(e) Velocity distribution, component (x,y) and magnitude

Figure 15: Results in the validation data set low tau low velocity, tuned parameters

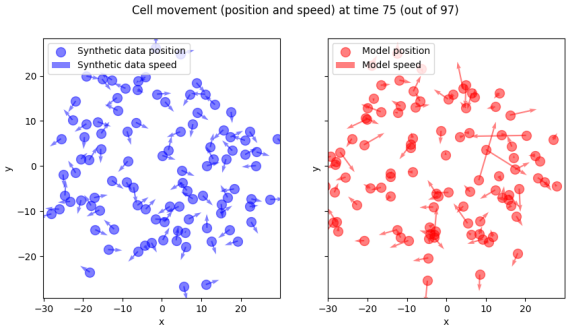
The model for *lt lv* fails in all summary statistics. The speed distribution in fig. 15e for the x axis is slightly skewed, but it is nowhere close the ground truth. The mean velocity seem to be rising constantly instead of staying constant.



(a) Frame 25



(b) Frame 50



(c) Frame 75

Figure 16: Comparison of the trajectories for a sample in the validation data set low tau low velocity

The dispersion of the cell over time is better than the case of *lt hv*, which can be seen in the Appendix on fig. 27a, but it suffers the same issue with the speeds vectors being too big as in *ht lv*, also appearing in the Appending on figs. 25c and 25d.

Tuning has been attempted directly on *lt lv* but there was no substantial difference from the tuning on lt_hv. Those can be seen in fig. 34 in the Appendix.

4.3.3 Real data - Tuning

Tuning also has been realized on the real data in case the optimal parameters for the simulator are different.

The setting corresponds to the number of the sample left out in the leave one out method. All parameters tested appear in table 8, the final choices are highlighted in bold red.

The distance between each curve with the reference curve has been summarized in tables 9 to 12.

Name	Base	Range
Setting	Leave 0	Leave 0
Distribution	laplace	laplace, **normal**
Horizon	1	1,2,3,**4**,5
Number of messages	2	1,**2**,3,4
Size of messages	256	32,**64**,128,256,512
Number of channels	8	**8**

Table 8: Tuning range and base parameters for the real data set

Curve	Laplace	**normal**
MSD	579455.66	**135598.01**
MV	138.52	**135.26**
SVCD	2.49	**0.78**
SVMD	2.36	**0.75**

Table 9: Distances for each metric in the tuning of the distribution

Curve	1	2	3	**4**	5
MSD	579455.66	3779164.11	675918.97	**111205.17**	167132.76
MV	138.52	672.83	194.42	**63.04**	115.27
SVCD	2.49	2.27	3.0	**2.0**	3.1
SVMD	2.36	**1.67**	2.65	1.86	2.61

Table 10: Distances for each metric in the tuning of the horizon

Curve	1	**2**	3	4
MSD	17027398.02	**583976.91**	6902861.04	1837563.3
MV	1359.04	**138.39**	708.75	365.44
SVCD	**2.03**	3.01	2.41	2.73
SVMD	**1.86**	2.89	2.07	2.05

Table 11: Distances for each metric in the tuning of the number of messages

Curve	32	**64**	128	256	512	1024
MSD	1081909.13	**84307.12**	13105693.17	584518.37	2322588.77	271614.37
MV	330.93	**50.4**	527.99	150.81	380.99	135.37
SVCD	3.39	**1.49**	3.7	2.47	2.73	2.18
SVMD	2.62	**1.16**	2.38	2.17	1.94, 2.01	

Table 12: Distances for each metric in the tuning of the size of messages

4.3.4 Real Data - Performances

To diminish the amount of variance in the training that was spotted to potentially be problematic, the training loss has been reduced from $1e-3$ to $5e-5$. The curriculum training has also been eased from $min(epoch+2, 4)$ to $min(epoch//32+2, 6)$.

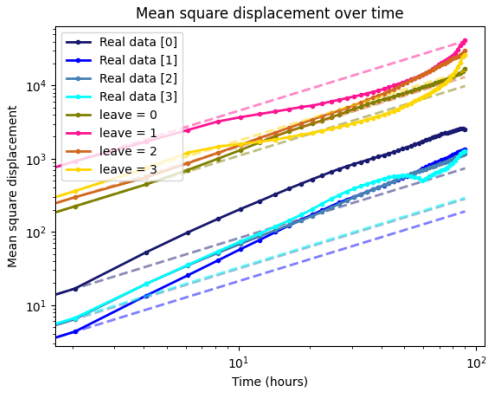
While in lesser effect, the variability in the losses -especially in the training loss- is still present in fig. 17f. Perhaps due to the limited data set size, the model starts over-fitting quite early. "Leave 1" started with a better initialization and the variability it experiences in the over-fitting region is way less prominent.

The skewness in the velocity has a hard time being translated in the model, which shows a preference for small velocities around 0 in the SVCD and SVMD of figs. 17c and 17d.

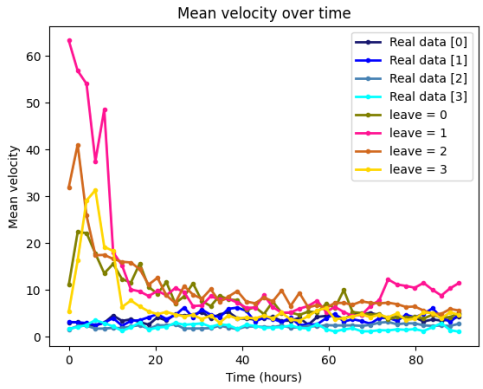
In the first twenty iterations of any scenario the velocity is much higher than it needs to, then the system settles down to a really good approximation of the true velocity in **??**. Although the velocity recovers, the MSD never does and its shape does not match the reference shapes.

It is to be noted that, on rare occasions, the training goes haywire and the loss reaches $-\inf$ in a few training samples. This is problematic

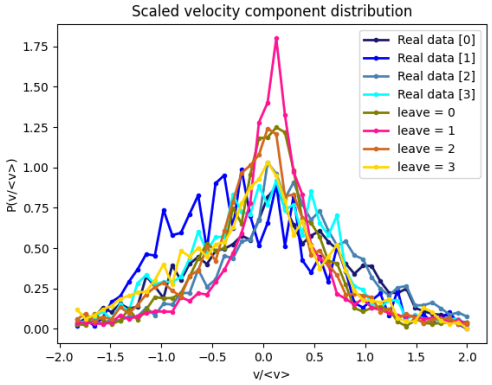
because, in term, the model produces *nan* only. Relaunching a new training with a new model is a band-aid solution but the source of this issue remains to be found.



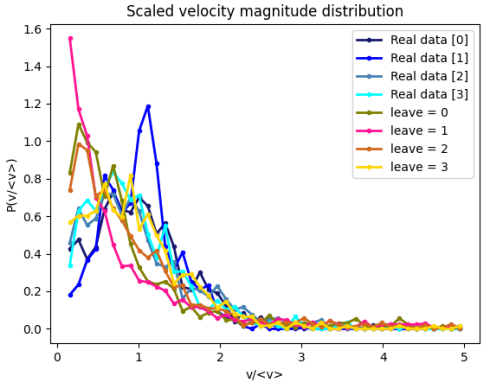
(a) Mean squared displacement



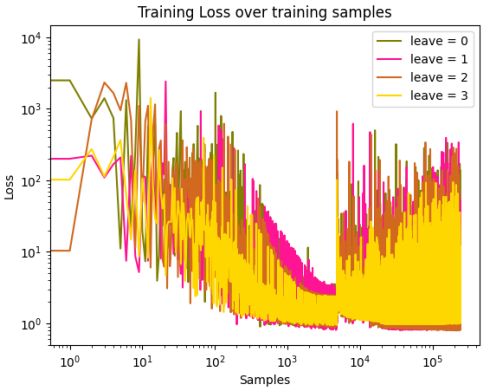
(b) Mean velocity



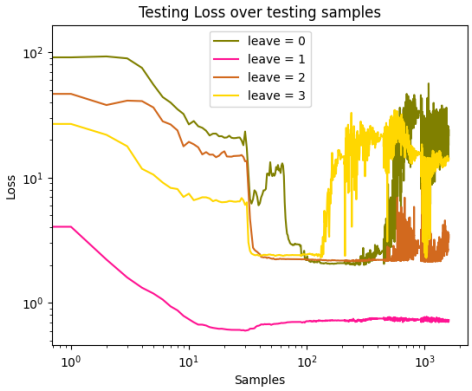
(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Training losses



(f) Testing losses

Figure 17: Final results for the real dataset

4.3.5 Real data - Decision processes

There is at least two types of choices the model proposes makes that can be explained through its attention mechanisms and its graph nature.

The first exploits the attention computed in eq. (5). Each edge has the attribute of the distance between the two vertices it connects only. GaTv2 computes an attention weight based upon this value and the internal states of the connected vertices. One can thus analyse the relationship of the distance to those weights an propose an explanation of the computation of the edge attentions.

The second focuses on the state of each vertex with its neighbors and how every attribute contributes to their final state.

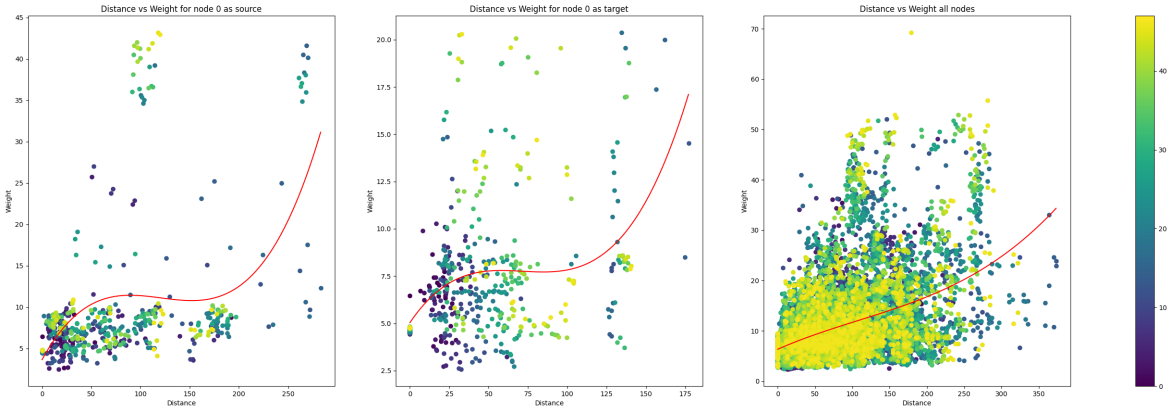
4.3.5.1 Edge attention

The implementation (pytorch geometric) of the GNN allows for an easy retrieval of the attention after the softmax of eq. (6) using the Attention-explainer. A copy of the matrix is saved after the softmax operation. Saving one line earlier allows for the extraction of the raw weights.

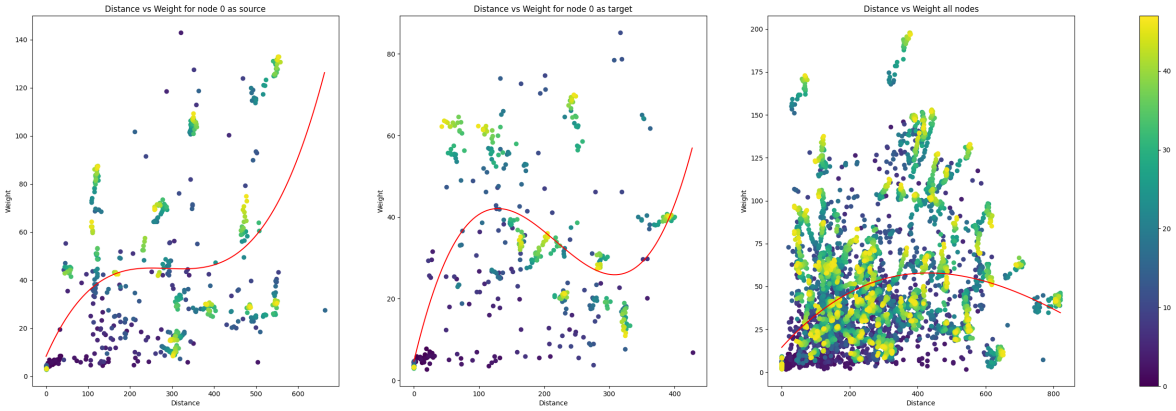
In the following figs. 18a to 18d, respectively from left to right, any vertex can be analyzed directly from its outgoing (source) or incoming (target) edge, or all can be viewed together. In particular, a random vertex named "vertex 0" has been picked in each scenario.

From the shape of the MSD in the previous results, it is expected that, as the time step evolve over time (purple to yellow), the average distance increases. Certain cells can also form clusters and stay close, explaining the small distances that can still appear later on.

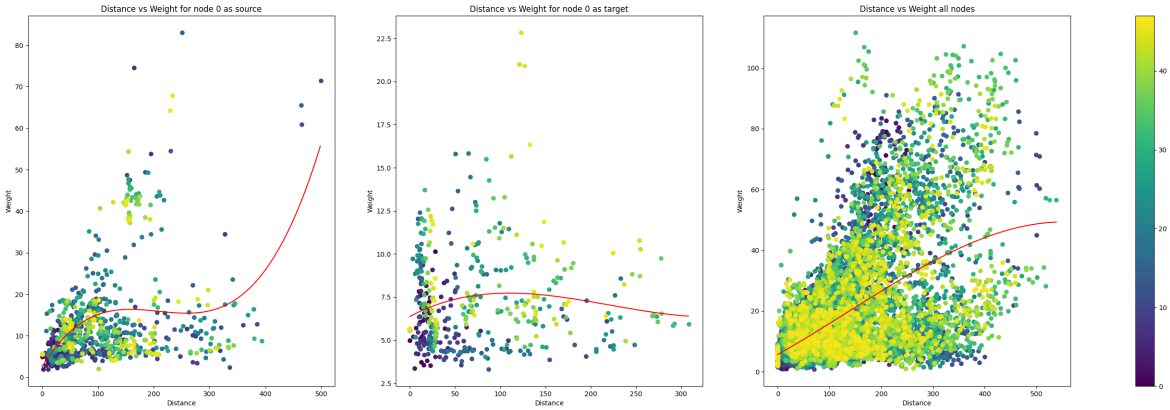
A third order regression, in red, has been provided for reference.



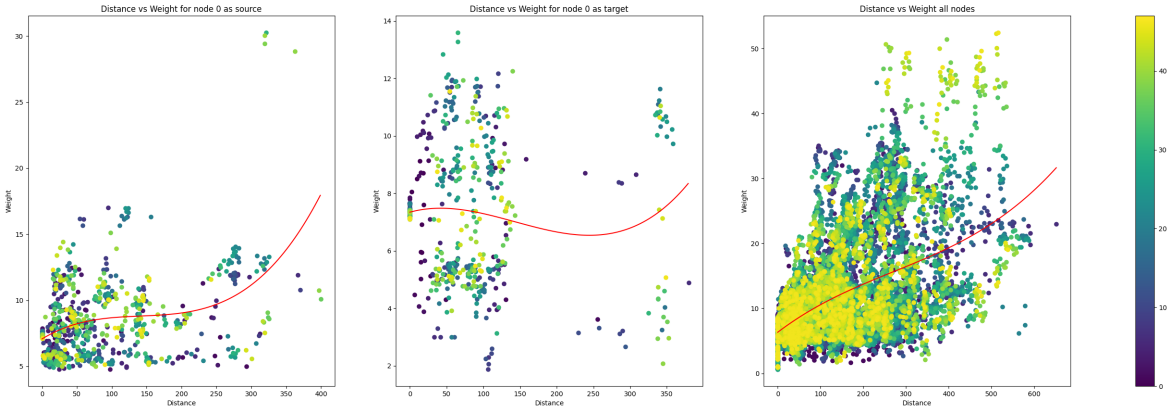
(a) Leave n°0 out



(b) Leave n°1 out



(c) Leave n°2 out



(d) Leave n°3 out

Figure 18: Attention weight over distance between cells
Single cell as source (left) Single cell as target (middle) All edges confounded (right)
Time step represented as colors from 0 (purple) to 47 (yellow)

On multiple graphs, especially in figs. 18c and 18d, evidence of correlation appear clearly in the confounded edge weights plotted against the distance between the vertex they connect. The precise shape of this correlation is hard to determine, though there seem to be multiple overlapping linear trends in fig. 18c (right).

The Allium simulator having defined a linear relationship for its driving force, it makes sense to retrieve the same behavior. This relationship is much more tame in the synthetic data, see in Appendix fig. 29.

It may be the case that the significantly higher uncertainty of the real model imposes a need to create a stronger bond as the cells distanciate to force a grouping. In essence, the message passing averages the contents of the vertices depending on their attention; the attention heavily being increased tends to lob the softmax into averaging more to the distant cells.

4.3.5.2 Attributes importance

The contribution of any attribute or edge can be measured using the mutual information. From this, the GNN-explainer identifies the subgraph $G_S \in G$ and the features $\{x_i | v_i \in G_S\}$ that are important for the prediction, using the eq. (27) [63].

$$\max_{G_S} MI(Y, (G_s, X_S)) = H(Y) - H(Y|G = G_s, X = X_s) \tag{27}$$

The measure of these contribution for each attribute, averaged over the time span of the simulation, is represented in fig. 19.

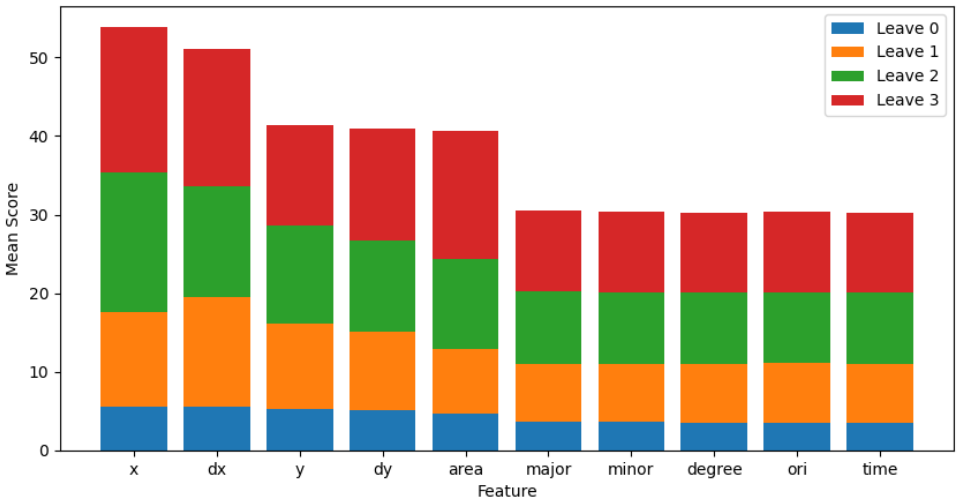


Figure 19: Average scores of different attributes per test sample

There is a clear distinction between the attributes actually used by the model -position, speeds and area- and those who are not. It also appears that the x axis offers a greater information about the system than the y axis. It may indicate a bias in the original samples selected.

5 Discussion

5.1 Limitations

This study was limited by the amount of real data available. However, this limitation has shown that even with an extremely small data set it is still possible to construct a model that exhibits acceptable summary statistics.

It is no secret that the real data present is difficult to use due to the trajectories being cut short only to continue a few time step further in another trajectory. This issue is inherent to the field of biology which, despite the use of advanced techniques such as the Bayesian trackers, remains plagued with unknowns and uncertainty. This work may prove ultimately beneficial to cover those issues since a good simulator could help improve automated tracking.

While studies on graph neural networks applied to migration in general did exist, none has been found on them specifically being applied to cell migration. The nature of the existing studies found all tried to infer either more global behaviors or discrete decision processes, which differ a lot in practice from the continuous individual velocities that were studied here.

The initial focus of this study has changed over time leading to a more restricted schedule. The synthetic data-set has also been changed from the original that featured a bounded world, with its edge wrapping around, to the current unbounded. Due to this, some issues had to be solved and never made it to the final version, losing much time in the process.

The results achieved in this thesis are only relevant to the particular data-sets used since no other source has been incorporated or tested upon. This limits greatly the confidence of the results due to the unknown generalization capabilities.

5.2 Findings

From the results on the synthetic data set, we can observe that it is possible to reproduce closely the results of a purpose made simulator

as long as the randomness of the motion does not overpower its driving forces.

The model produced fits easily on an average personal computer and allows for fast inference from a single snapshot in time of the data.

Although the distributions of the velocities is less of a good match for the lv cases, the evolution of the cells in the models' simulation still visually closely resembles the original from Allium. Moreover, many summary statistics overlapped closely with ground truth.

From the different tunings can be learned that the distribution used for the aleatoric uncertainty as well as horizon influence greatly the correctness of the distribution of speeds. And this choice also differ between the real and synthetic data sets.

Contrary to the expectation of "more is better", when it comes to the input data, adding subsequent snapshots of the environment (in the form of the horizon leading to the event) decreases performances. However, for the real data set, there seems to be the opposite conclusion, and a sweet spot can be found for a window of size greater than one.

In all cases but one, the loss itself does not seem to vary by any significant amount in the testing set, reinforcing the need to measure actual performances using summary statistics.

The precise reason why there appears to have different optimal tuning parameters for different data sets may indicate that either the generalization properties of the model could be improved, or that there exists fundamental differences in their mechanisms.

The analysis of the importance of the attributes confirm that including the speeds along each axis does have a significant impact on the summary statistics and the loss. This result is all the more surprising because, in this analysis of the real case, an horizon of 4 was used; the information is already present in the form of past snapshots but the model seems to have an easier time taking this shortcut.

The attention analysis can uncover fundamental decision mechanisms by highlighting correlations between variables and internal weights although it may prove difficult to relate them to actual quantities. Some additional work may be needed to isolate the traces of linear dependencies that sometimes exuded and, from an empirical analysis, try to relate its slope to concrete driving forces.

Due to the abstract nature of the data provided, this method can be directly applied to the simulation of any network of homogeneous individual.

5.3 Future Work

Areas of improvement for this thesis include the coverage of different architectures of GNN to compare performances, though the use of the most general model may indicate an upper bound for the size and parameters chosen.

The assumptions, while stated, were not put in question. Some, like the architecture itself, might have benefited from trying other main stream models for particle movement prediction.

Many values for hyper-parameters, such as the learning rate, number of heads and depth of the model were directly lifted from the paper recommendations (AdamP, GaTv2, Attention) and did not go under scrutiny. While it may be a good starting point to follow recommendations it would be useful to push the investigation of the optimal parameters further in this direction.

The particular method of optimization used in this thesis relied on comparing all deviations to the arbitrarily defined "base" parameters. This could be improved in further work by using constrained minimization at the cost of much computing power and time.

Recent efforts in the field of transformers have highlighted that the initialization of the parameters was crucial to the stability and performances [64]. This issue was not covered and all initialization were left to the default values of their implementation.

Some questions were asked but not answered, especially the intriguing case of the many-to-many model giving worse results. An answer may lie in the fact that auto-regressive inference usually yields better results because the model can use its past inferences as ground truth.

Following the limitations in the data-sets provenance and size, future work may focus on studying the results based on bigger and/ or more diverse data.

It could be interesting to provide capabilities for the model to ingest networks with heterogeneous members to expand the areas in which it

could be useful.

5.4 Code

Most of the code used during this thesis has been saved in this github repository : https://github.com/Pangasius/graph-displacement.

References

[1] Allan Franklin and Slobodan Perovic. Experiment in Physics. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 edition, 2021.

[2] Subra Suresh. *Fatigue of materials*. Cambridge university press, 1998.

[3] Valentin Heller. Scale effects in physical hydraulic engineering models. *Journal of Hydraulic Research*, 49(3):293–306, 2011.

[4] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[5] Junuthula Narasimha Reddy. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

[6] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

[7] John Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 378(2166):20190061, 2020.

[8] Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[9] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures, 2019.

[10] Zi-Ang Shen, Tao Luo, Yuan-Ke Zhou, Han Yu, and Pu-Feng Du. NPI-GNN: Predicting ncRNA–protein interactions with deep graph neural networks. *Briefings in Bioinformatics*, 22(5), 04 2021. bbab051.

[11] Jie Zhang, Yishan Du, Pengfei Zhou, Jinru Ding, Shuai Xia, Qian Wang, Feiyang Chen, Mu Zhou, Xuemei Zhang, Weifeng Wang,

et al. Predicting unseen antibodies' neutralizability via adaptive graph neural networks. *Nature Machine Intelligence*, pages 1–13, 2022.

[12] Xiaogang Ren, Xue Zhang, and Chengli Zhao. Migration data-based graph neural network for disease forecasting. In *2022 8th International Conference on Big Data and Information Analytics (BigDIA)*, pages 278–284, 2022.

[13] Thomas E Creighton. Protein folding. *Biochemical journal*, 270(1):1, 1990.

[14] Telma DaPalma, Bently P Doonan, Nicole M Trager, and Laura M Kasman. A systematic approach to virus–virus interactions. *Virus research*, 149(1):1–9, 2010.

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[16] Jose Suarez-Varela, Paul Almasan, Miquel Ferriol-Galmes, Krzysztof Rusek, Fabien Geyer, Xiangle Cheng, Xiang Shi, Shihan Xiao, Franco Scarselli, Albert Cabellos-Aparicio, and Pere Barlet-Ros. Graph neural networks for communication networks: Context, use cases and opportunities. *IEEE Network*, pages 1–8, 2022.

[17] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020.

[18] Shizhen Huang, ShaoDong Zheng, and Ruiqi Chen. Multi-source transfer learning with graph neural network for excellent modelling the bioactivities of ligands targeting orphan g protein-coupled receptors. *Mathematical Biosciences and Engineering*, 20(2):2588–2608, 2023.

[19] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[20] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[21] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[22] Francisco JH Heras, Francisco Romero-Ferrero, Robert C Hinz, and Gonzalo G de Polavieja. Deep attention networks reveal the rules of collective motion in zebrafish. *PLoS computational biology*, 15(9):e1007354, 2019.

[23] Takaki Yamamoto, Katie Cockburn, Valentina Greco, and Kyogo Kawaguchi. Probing the rules of cell coordination in live tissues by interpretable machine learning based on graph neural networks. *PLOS Computational Biology*, 18(9):e1010477, 2022.

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

[25] Gabriel Popkin. Jammed cells expose the physics of cancer. *QuantaMagazine*, 2016.

[26] Eliane Blauth, Hans Kubitschke, Pablo Gottheil, Steffen Grosser, and Josef A Käs. Jamming in embryogenesis and cancer progression. *Frontiers in Physics*, 9:666709, 2021.

[27] Nicole M Le Douarin and Elisabeth Dupin. Multipotentiality of the neural crest. *Current Opinion in Genetics & Development*, 13(5):529–536, 2003.

[28] JAMES A. WESTON. The migration and differentiation of neural crest cells* *this paper is dedicated to sally wilens on the occasion of

her retirement. she acquainted me with "the chief" though i never met him. volume 8 of *Advances in Morphogenesis*, pages 41–114. Elsevier, 1970.

[29] Daniel L. Barton, Silke Henkes, Cornelis J. Weijer, and Rastko Sknepnek. Active vertex model for cell-resolution description of epithelial tissue mechanics. *PLOS Computational Biology*, 13(6):1–34, 06 2017.

[30] David Flaherty et al. Simulation based inference in'tiny data'biology. 2022.

[31] Namid R. Stillman and Roberto Mayor. Generative models of morphogenesis in developmental biology. *Seminars in Cell & Developmental Biology*, 147:83–90, 2023. Special issue: Rearrangement of co-cultured cellular systems: biological and physical aspects.

[32] Ren-Hua Wang, Tao Luo, Han-Lin Zhang, and Pu-Feng Du. Plagnn: Computational inference of protein subcellular location alterations under drug treatments with deep graph neural networks. *Computers in Biology and Medicine*, 157:106775, 2023.

[33] Tal Ben-Haim and Tammy Riklin Raviv. Graph neural network for cell tracking in microscopy videos. In *European Conference on Computer Vision*, pages 610–626. Springer, 2022.

[34] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[35] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[36] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[37] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman,

N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[38] Peter Veranič, Maruša Lokar, Gerhard J. Schütz, Julian Weghuber, Stefan Wieser, Henry Hägerstrand, Veronika Kralj-Iglič, and Aleš Iglič. Different types of cell-to-cell connections mediated by nanotubular structures. *Biophysical Journal*, 95(9):4416–4425, 2008.

[39] Marie-Luce Vignais, Andrés Caicedo, Jean-Marc Brondello, and Christian Jorgensen. Cell connections by tunneling nanotubes: effects of mitochondrial trafficking on target cell metabolism, homeostasis, and response to therapy. *Stem cells international*, 2017, 2017.

[40] Denis Weaire and Nicolas Rivier. Soap, cells and statistics—random patterns in two dimensions. *Contemporary Physics*, 25(1):59–99, 1984.

[41] Daniel M Sussman and Matthias Merkel. No unjamming transition in a voronoi model of biological tissue. *Soft matter*, 14(17):3397–3403, 2018.

[42] P Pathmanathan, J Cooper, A Fletcher, G Mirams, P Murray, J Osborne, J Pitt-Francis, A Walter, and S J Chapman. A computational study of discrete mechanical tissue models. *Physical Biology*, 6(3):036001, apr 2009.

[43] Zhongzhi Zhang, Jihong Guan, Bailu Ding, Lichao Chen, and Shuigeng Zhou. Contact graphs of disk packings as a model of spatial planar networks. *New Journal of Physics*, 11(8):083007, 2009.

[44] Jonathan PK Doye and Claire P Massen. Self-similar disk packings as model spatial scale-free networks. *Physical Review E*, 71(1):016128, 2005.

[45] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[46] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[47] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[48] Alana de Santana Correia and Esther Luna Colombini. Attention, please! A survey of neural attention models in deep learning. *CoRR*, abs/2103.16775, 2021.

[49] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[50] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[52] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens, 2023.

[53] Annamaria Guolo. Robust techniques for measurement error correction: a review. *Statistical Methods in Medical Research*, 17(6):555–580, 2008. PMID: 18375458.

[54] V. H. Lachos, F. V. Labra, H. Bolfarine, and Pulak Ghosh. Multivariate measurement error models based on scale mixtures of the skew–normal distribution. *Statistics*, 44(6):541–556, 2010.

[55] Jagdish K Patel and Campbell B Read. *Handbook of the normal distribution*, volume 150. CRC Press, 1996.

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

[57] Adam Shellard and Roberto Mayor. Rules of collective migration: from the wildebeest to the neural crest. *Philosophical Transactions of the Royal Society B*, 375(1807):20190387, 2020.

[58] Kristina Ulicna, Giulia Vallardi, Guillaume Charras, and Alan R. Lowe. Automated deep lineage tree analysis using a bayesian single cell tracking approach. *bioRxiv*, 2020.

[59] Anna Bove, Daniel Gradeci, Yasuyuki Fujita, Shiladitya Banerjee, Guillaume Charras, and Alan R. Lowe. Local cellular neighborhood controls proliferation in cell competition. *Molecular Biology of the Cell*, 28(23):3215–3228, 2017.

[60] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights, 2021.

[61] Naama Gal, Diana Lechtman-Goldstein, and Daphne Weihs. Particle tracking in living cells: a review of the mean square displacement method and beyond. *Rheologica Acta*, 52:425–443, 2013.

[62] Vincent Hakim and Pascal Silberzan. Collective cell migration: a physics perspective. *Reports on Progress in Physics*, 80(7):076601, apr 2017.

[63] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks, 2019.

[64] Siddharth Krishna Kumar. On weight initialization in deep neural networks, 2017.

A Appendix

A.1 Developments

This is the development of the eqs. (15) and (16).

In both cases, the loss comes from the need to maximize the likelihood $p(X|\theta)$, take the minimization objective from eqs. (28) and (29) then single out a pair $(\dot{n}_i, m_i)$ as $(\dot{n}_{true}, m)$. Finally remove the constant term.

A.1.1 Normal

$$\begin{aligned} arg\max_{\theta} p_{normal}(X|\theta) &= arg\max_{\theta} \prod_{\dot{n}_i, m_i \in X} p_{normal}(\dot{n}_i|m_i, \theta) \\ &= arg\max_{\theta} \prod_{\dot{n}_i, m_i \in X} \frac{1}{\sqrt{2\pi}\sigma(m_i)}\ exp\left(-\frac{(\dot{n}_i - \mu(m_i))^2}{2\sigma^2(m_i)}\right) \\ &= arg\min_{\theta} \sum_{\dot{n}_i, m_i \in X} \frac{(\dot{n}_i - \mu(m_i))^2}{2\sigma^2(m_i)} + log(\sigma(m_i)) + C \end{aligned} \tag{28}$$

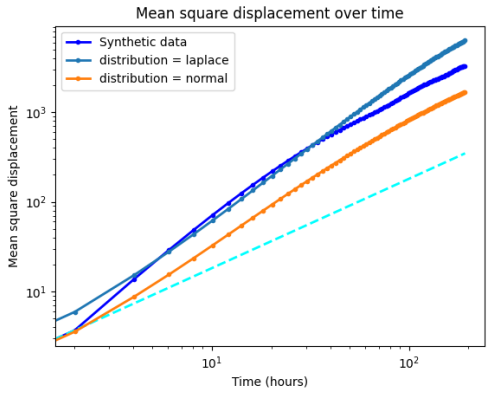
A.1.2 Laplace

$$\begin{aligned} arg\max_{\theta} p_{laplace}(X|\theta) &= arg\max_{\theta} \prod_{\dot{n}_i, m_i \in X} p_{laplace}(\dot{n}_i|m_i, \theta) \\ &= arg\max_{\theta} \prod_{\dot{n}_i, m_i \in X} \frac{1}{2b(m_i)}\ exp\left(-\frac{|\dot{n}_i - \mu(m_i)|}{b(m_i)}\right) \\ &= arg\min_{\theta} \sum_{\dot{n}_i, m_i \in X} \frac{|\dot{n}_i - \mu(m_i)|}{b(m_i)} + log(b(m_i)) + C \end{aligned} \tag{29}$$

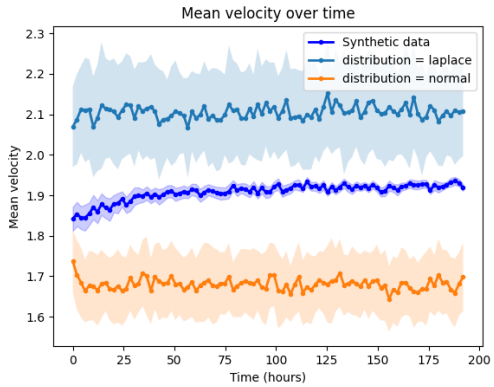
A.2 Synthetic data - Tuning

A.2.1 Distribution

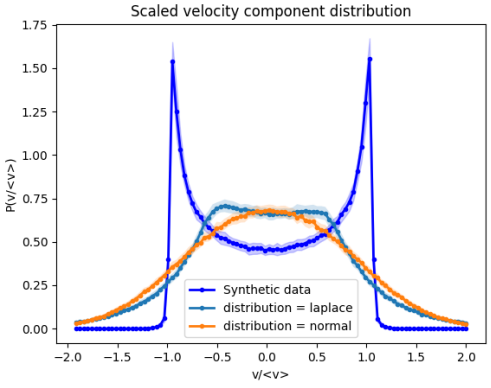
Two different options were investigated for the choice of distribution for the representation of the aleatoric noise : the Normal and Laplace distribution. We have to keep in mind that the losses cannot be directly compared in terms of value because the first is more akin to a difference squared while the other is closer to an absolute difference. Their shape can still bring us more insight on their training and testing however.



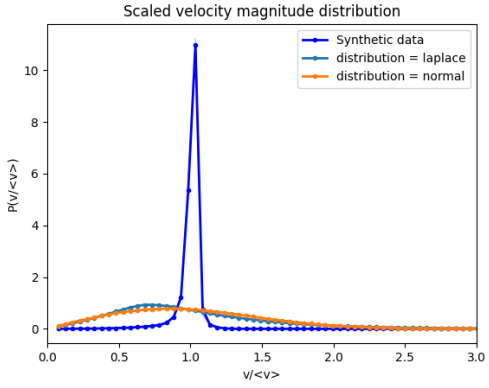
(a) Mean squared displacement



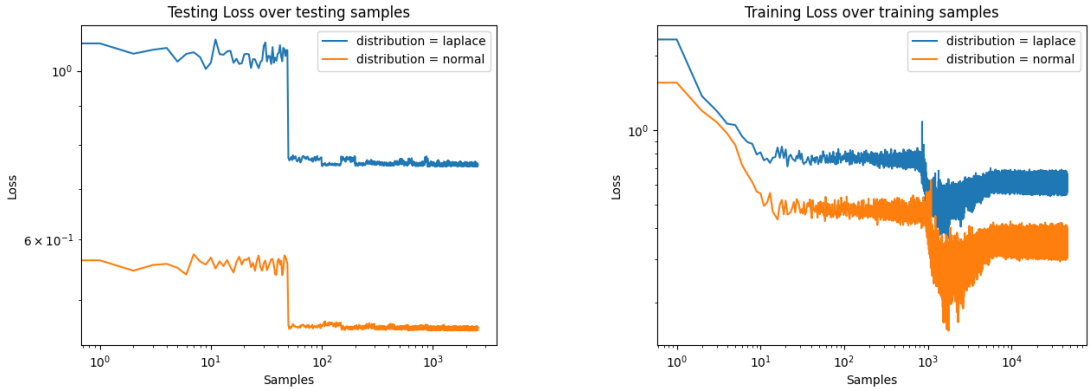
(b) Mean velocity



(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Testing loss

(f) Training loss

Figure 20: Tuning of the sampling distribution

The mean square displacement (MSD) over time seems to be fitted better with the laplace distribution, along with the scaled velocity component distribution (SVCD).

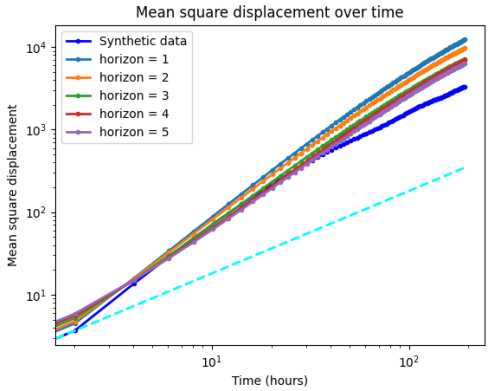
Both distributions lead to a difference of 0.2, laplace overshoots and has greater uncertainty while normal undershoots.

The scaled velocity magnitude distribution has no clear winner either since laplace is sharper but offset and normal is more spread out but centered correctly.

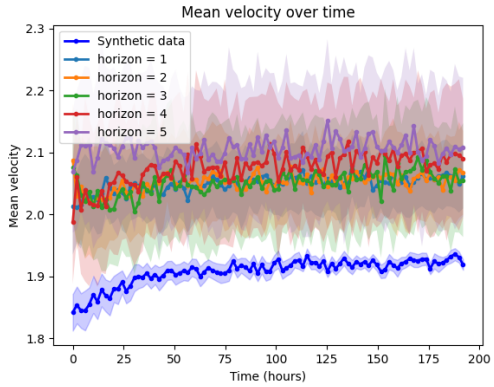
The losses indicate they both have similar behavior during training.

A.2.2 Horizon

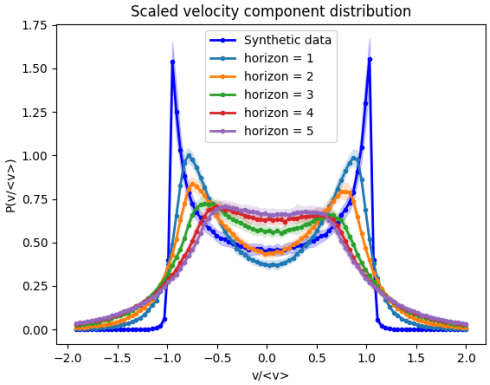
As mentioned previously, we can expand the horizon of input of the Markov's chain representing our knowledge about the world by increasing the length of input to our encoder.



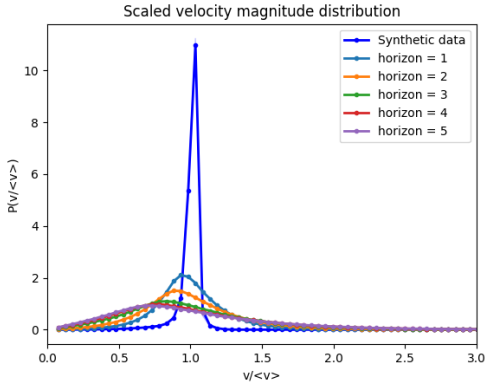
(a) Mean squared displacement



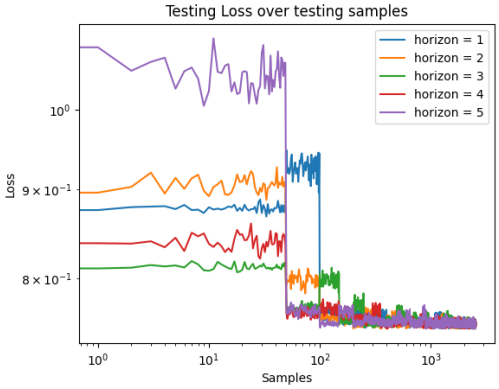
(b) Mean velocity



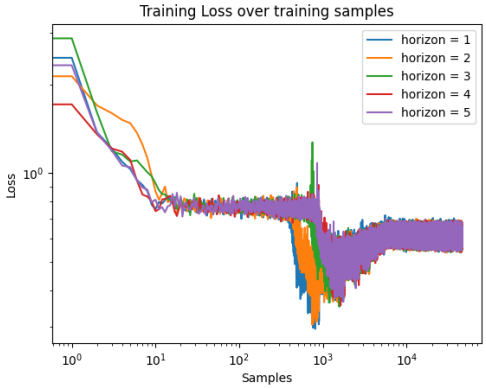
(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Testing loss



(f) Training loss

Figure 21: Tuning of the input's horizon

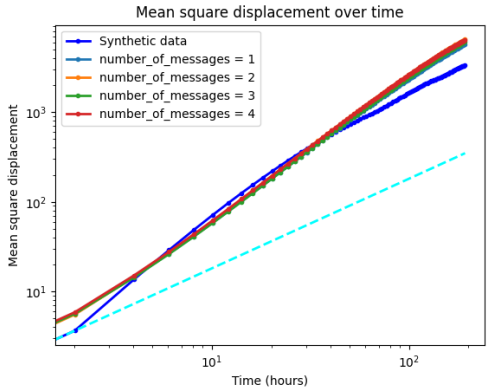
The most important part of the MSD to fit is after the initial system has relaxed around 30 hours and before the uncertainty builds up too much. This is the reason horizon 4 and 5 come out on top.

However, for the mean velocity, SVCD and SVMD, horizon 1 greatly outperforms the rest of the models.

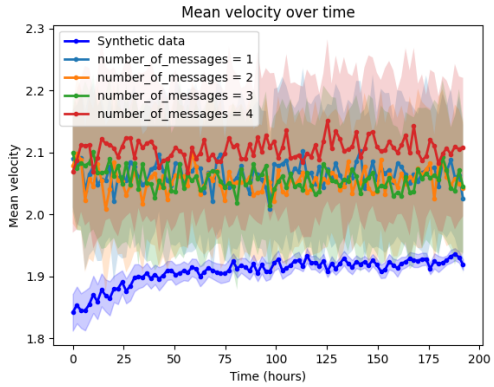
Their training show that the more horizon we add, the slower the 'breakthrough' of the training occurs. Though on the testing loss, this conclusion does not seem to apply.

A.2.3 Number of messages

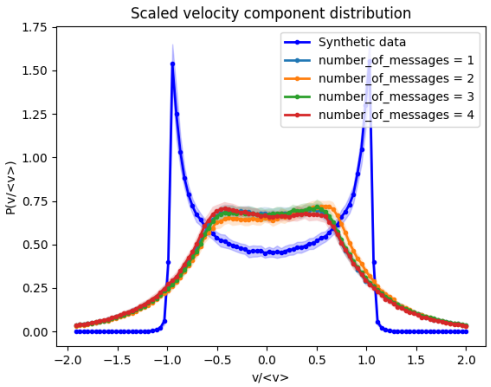
The number of messages, or number of GATv2 in our model, can also be tuned.



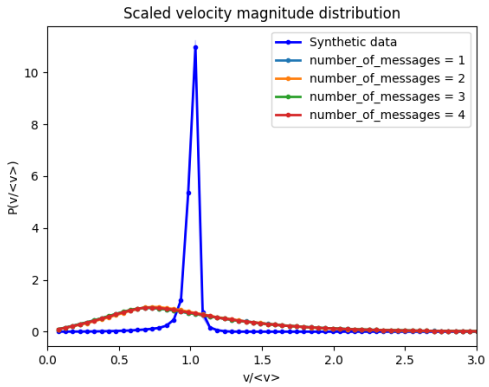
(a) Mean squared displacement



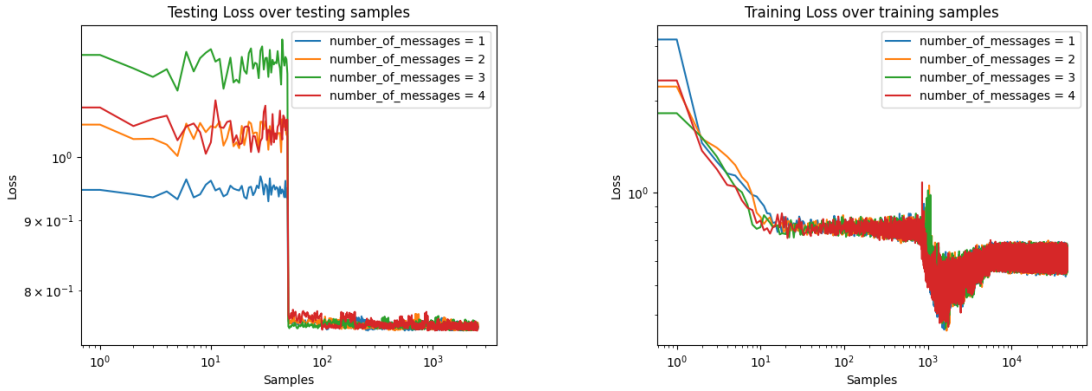
(b) Mean velocity



(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



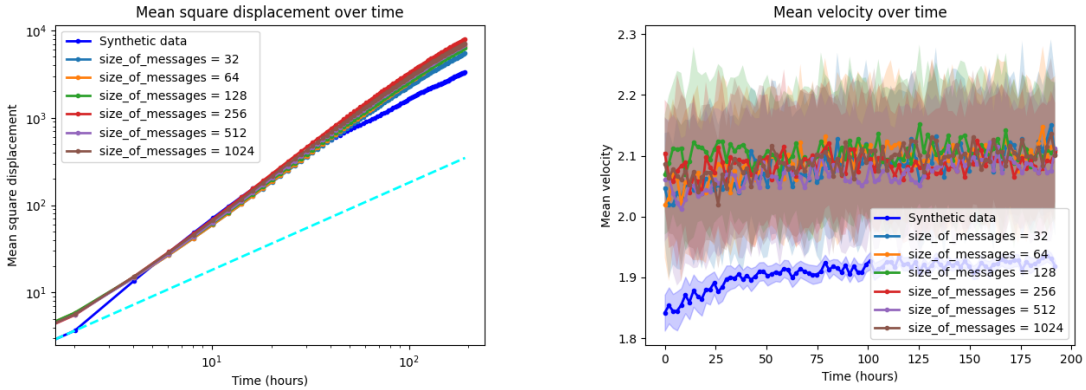
(e) Testing loss (f) Training loss

Figure 22: Tuning of the number of messages

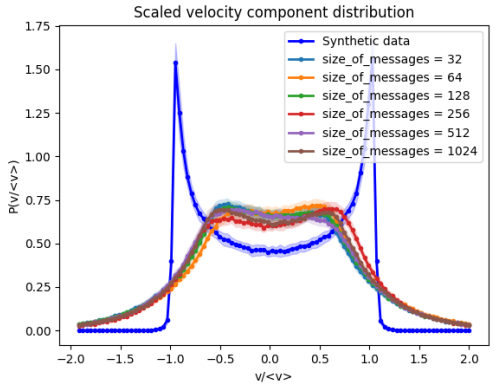
The influence of this parameter seem small, there is some little offset noticeable in the SVCD but they do not indicate better or worse results.

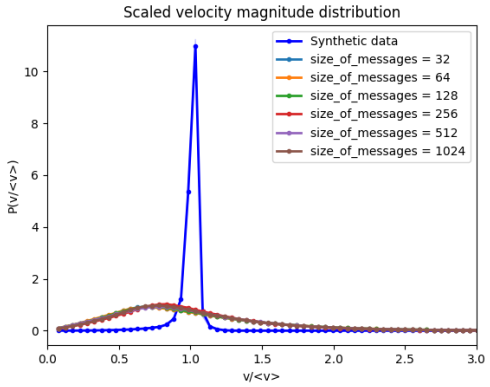
A.2.4 Size of messages

Most of the model relies on a single parameter to determine its width, and this number corresponds to the size of the messages sent, which also has to be set.

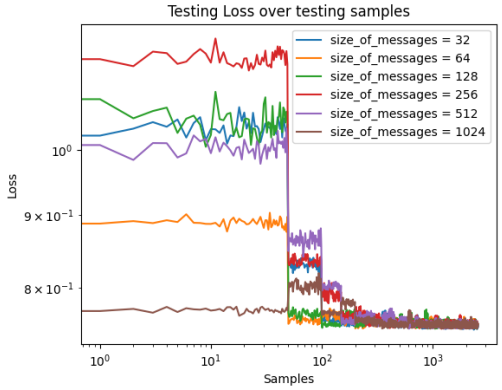


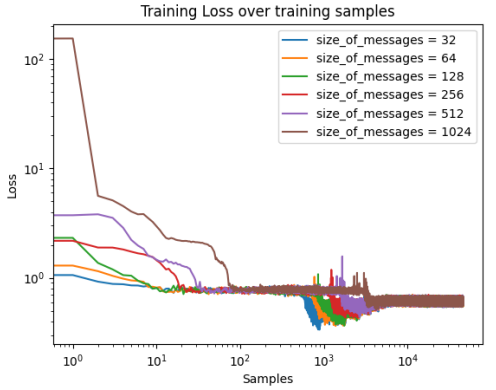
(a) Mean squared displacement (b) Mean velocity





(c) Scaled velocity component distribution (d) Scaled velocity magnitude distribution





(e) Testing loss (f) Training loss

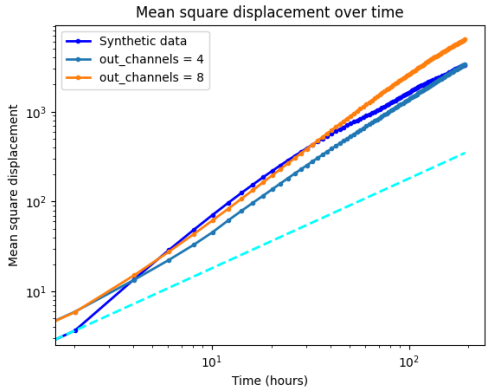
Figure 23: Tuning of the size of messages

Once again, the SVCD varies the most but picking a single winner seems impossible.

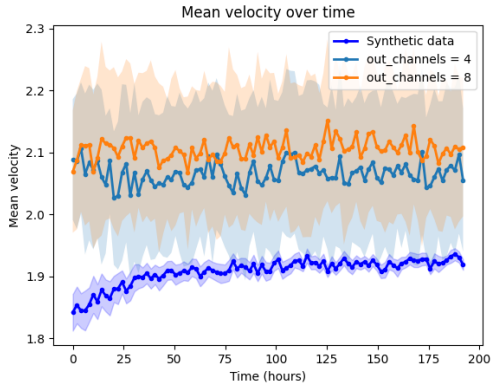
The 'breakthrough' appears later in the training loss and is less deep, though they all end up at the same loss.

A.2.5 Number of channels

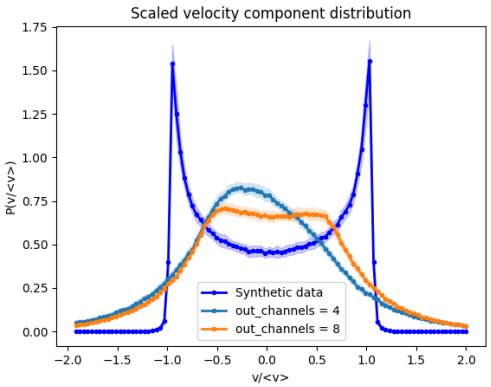
We can choose to either produce only the change in movement, $out\_channels = 4$, or also produce the change in speed, $out\_channels = 8$.



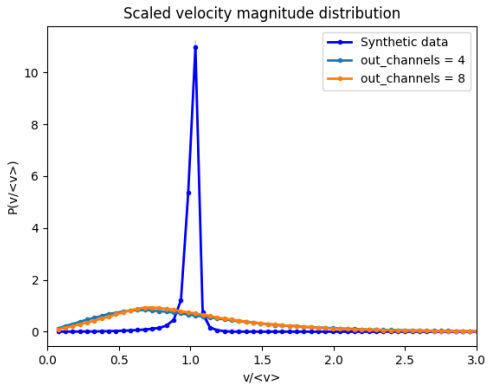
(a) Mean squared displacement



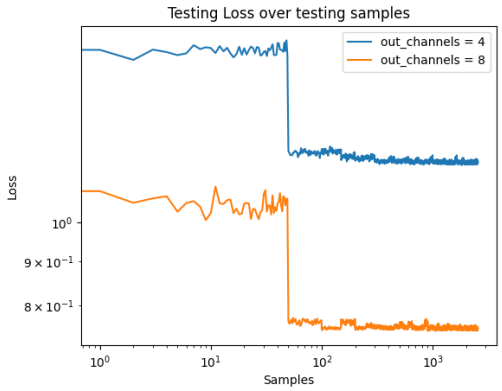
(b) Mean velocity



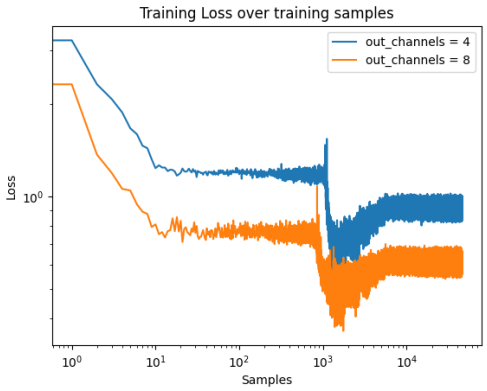
(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Testing loss



(f) Training loss

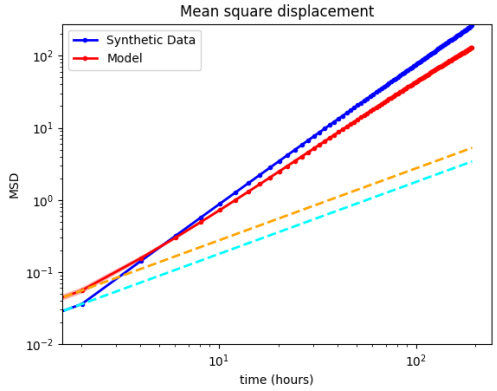
Figure 24: Tuning of the number of output

Predicting the velocities seem to have a major importance on all criterions including the losses, which had not happened with the other parameters.

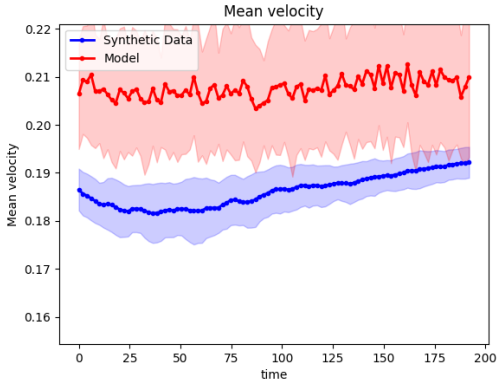
The mean velocity is a bit closer to the ground truth but every other metric indicates 8 output channels to be optimal.

A.3 Synthetic data - Performances

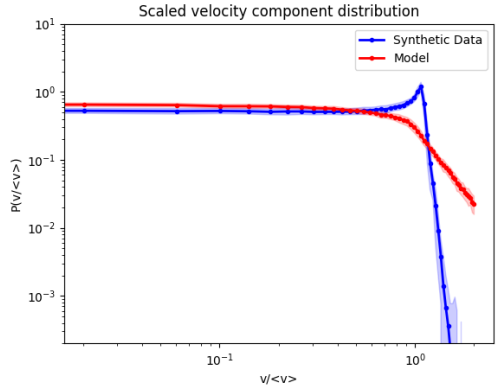
A.3.0.1 high tau, low velocity



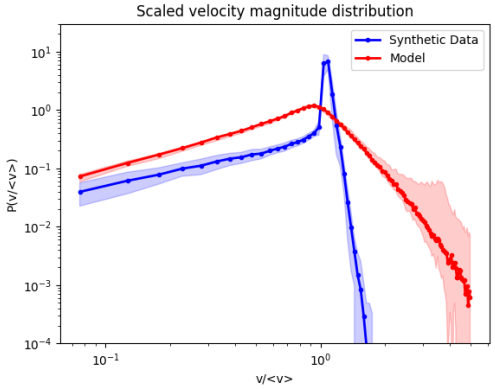
(a) Mean squared displacement



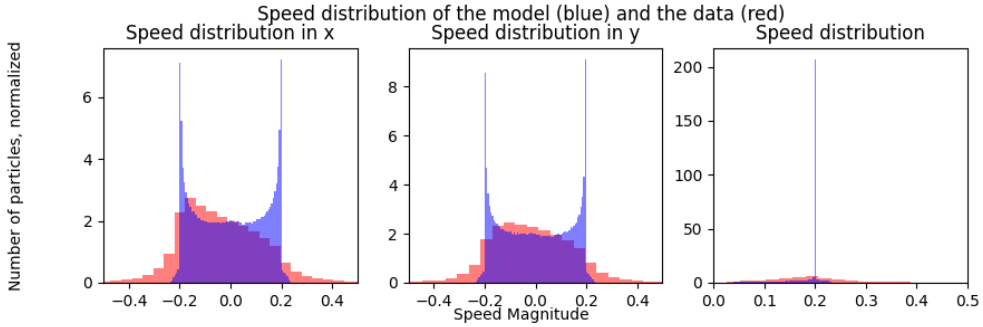
(b) Mean velocity



(c) Scaled velocity component distribution



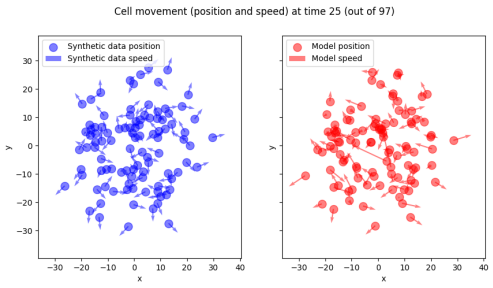
(d) Scaled velocity magnitude distribution



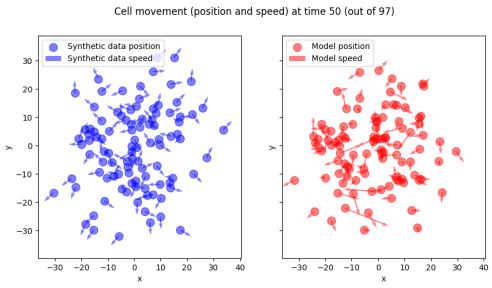
(e) Velocity distribution, component (x,y) and magnitude

Figure 25: Results in the validation data set high tau low velocity, tuned parameters

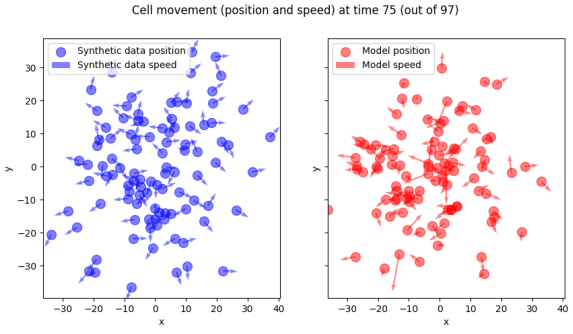
The lower velocity seem to be harder to simulate. Even though the summary statistics are satisfactory (except for SVCD in fig. 25c), the speed distribution does not seem to be the same for the separated axis on fig. 25e.



(a) Frame 25



(b) Frame 50

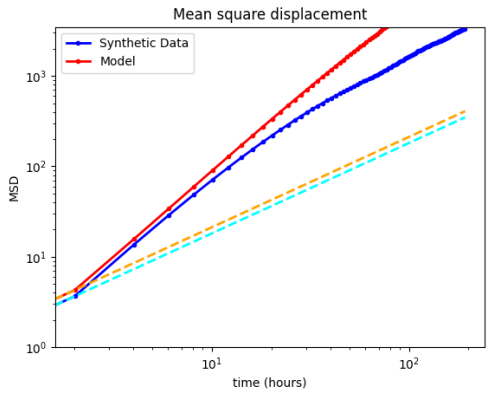


(c) Frame 75

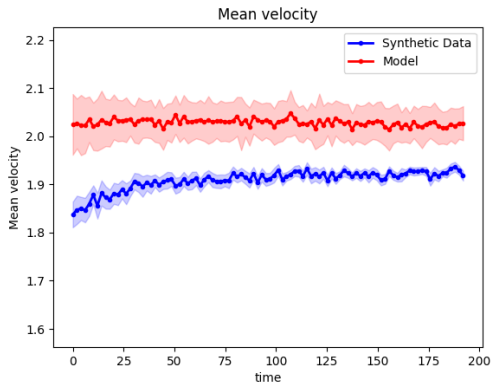
Figure 26: Comparison of the trajectories for a sample in the validation data set high tau low velocity

While there seems to be more clustering in the model's predictions, it is hard to rule them as incorrect or the opposite. The major difference seems to lie in the speed vectors, especially in fig. 26a in the middle, where is situated a huge speed vector.

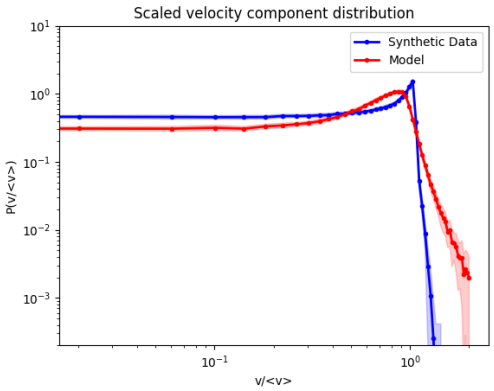
A.3.0.2 low tau, high velocity



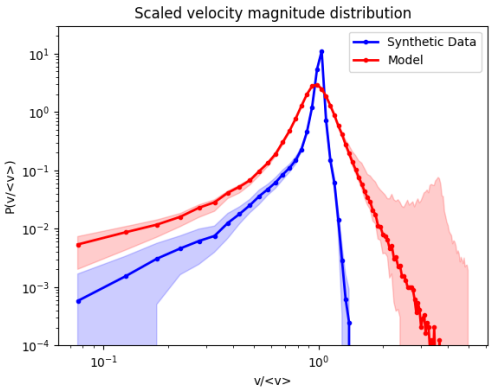
(a) Mean squared displacement



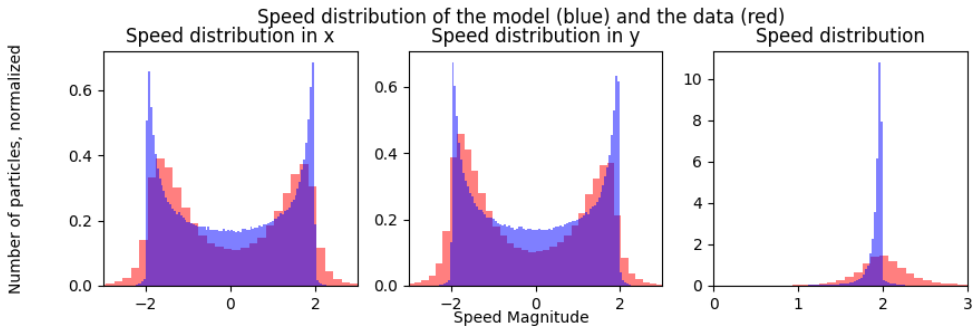
(b) Mean velocity



(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution

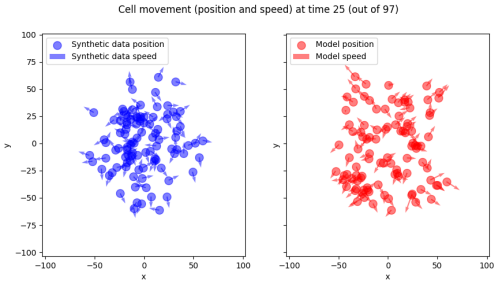


(e) Velocity distribution, component (x,y) and magnitude

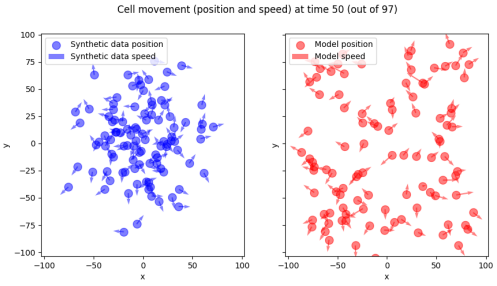
Figure 27: Results in the validation data set low tau high velocity, tuned parameters

All summary statistics seem to stay close to the ground truth in the case of *lt hv*. The most different seem to be in fig. 27b where can be seen the model overshooting the speed.

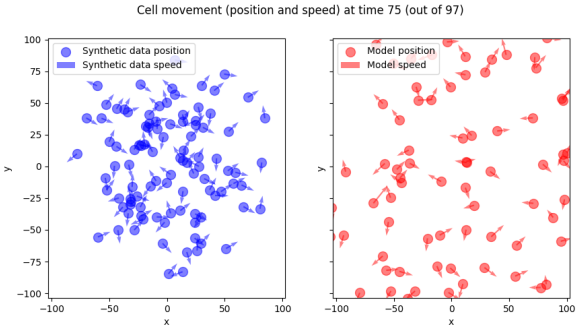
Contrary to *ht lv*, the shape of the speed distribution in each axis on fig. 27e is well represented by the model.



(a) Frame 25



(b) Frame 50

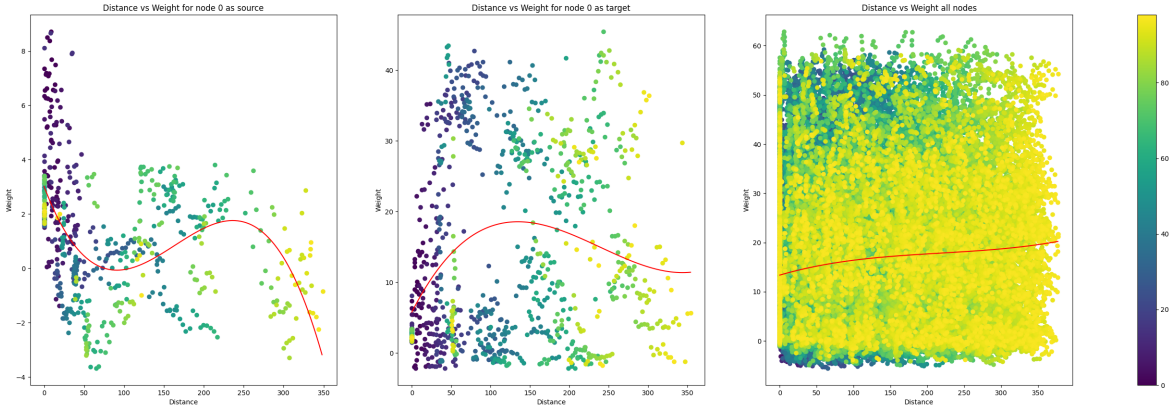


(c) Frame 75

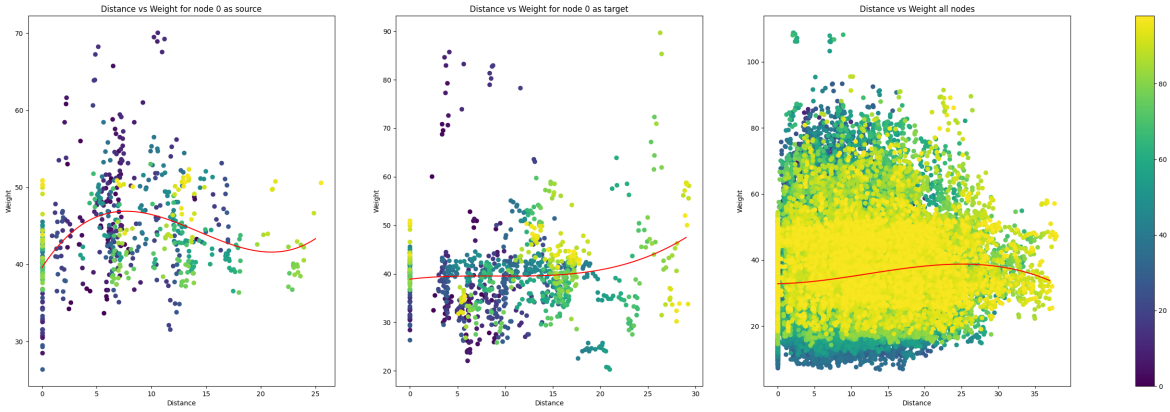
Figure 28: Comparison of the trajectories for a sample in the validation data set low tau high velocity

Despite the accuracy of the summary statistics, it seems the model for *lt hv* does not keep the cells bunched up together. This is especially noticeable in fig. 28c where they go out of frame, which is bounded by the minimum and maximum positions values of the group of cells in the synthetic data sample.

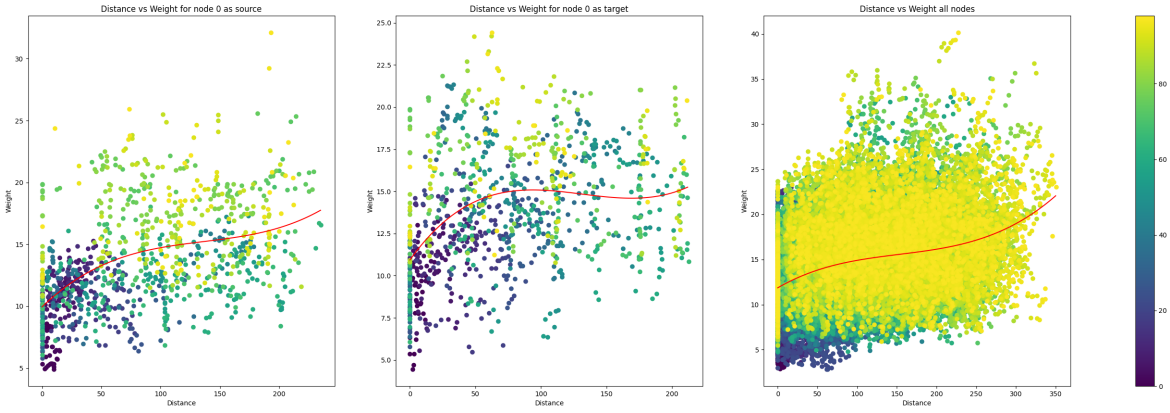
A.4 Synthetic data - Edge decision



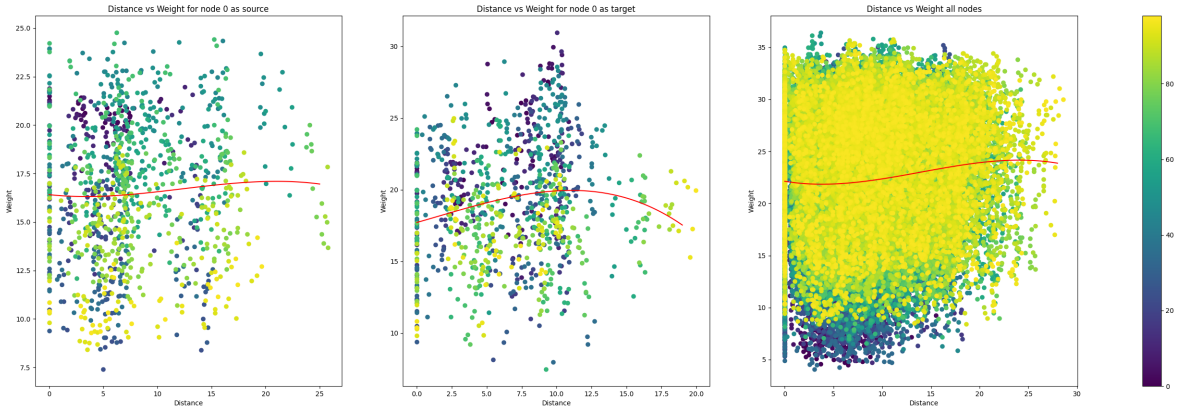
(a) ht hv


Distance vs Weight for node 0 as source
Distance vs Weight for node 0 as target
Distance vs Weight all nodes
Weight
Distance

(b) ht lv


Distance vs Weight for node 0 as source
Distance vs Weight for node 0 as target
Distance vs Weight all nodes
Weight
Distance

(c) lt hv



(d) lt lv

Figure 29: Attention weight over distance between cells
Single cell as source (left) Single cell as target (middle) All edges confounded (right)
Time step represented as colors from 0 (purple) to 47 (yellow)

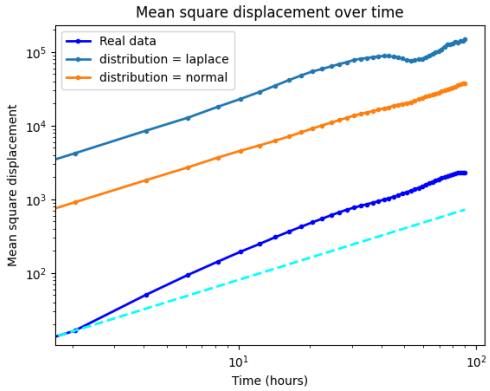
Apart from fig. 29c, there seems to be little to no correlation between the weights and the distances.

Compared to the real case, the evolution of a singular edge over time is much more apparent since there is less variation. One can follow a dot though time, which create those vertical streak patterns.

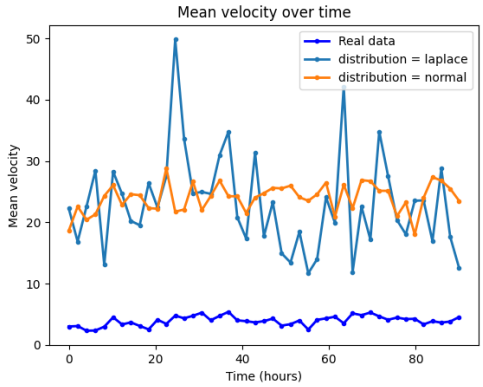
A.5 Real data - Tuning

A.5.1 Distribution

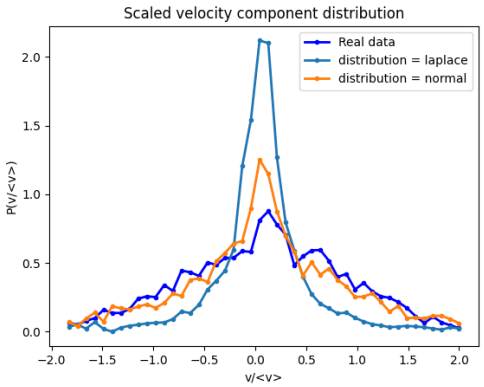
Once again, the losses are not directly comparable, though they can be useful to assess the well being of the training process.



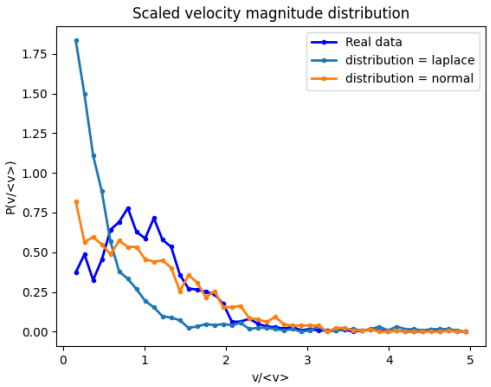
(a) Mean squared displacement



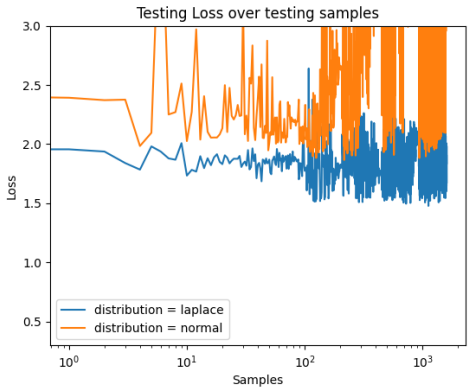
(b) Mean velocity



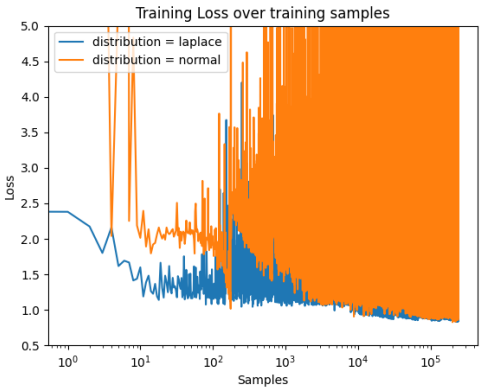
(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Testing loss



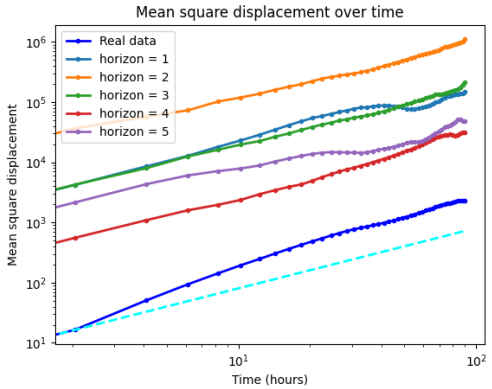
(f) Training loss

Figure 30: Tuning of the sampling distribution for the real data set

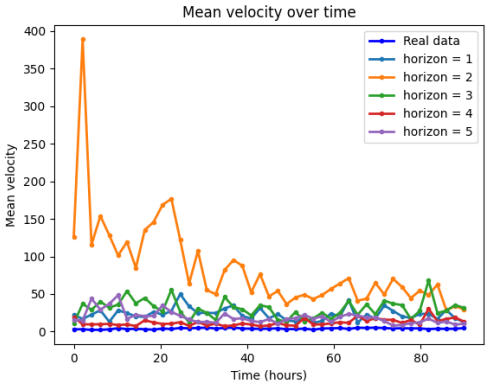
The tendency from the synthetic data seems to have reversed. On both the SVCD and SVMD, the normal distribution seems to fit better. The MSD also is a lot closer to the true value.

For both distributions though it seems the training is unstable and the more the model is trained the more variance increase for both the training and testing samples. This is a little more controlled for laplace.

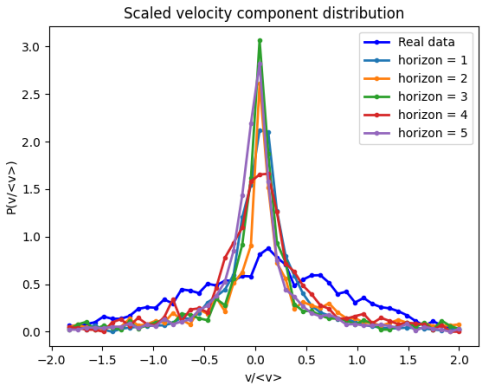
A.5.2 Horizon



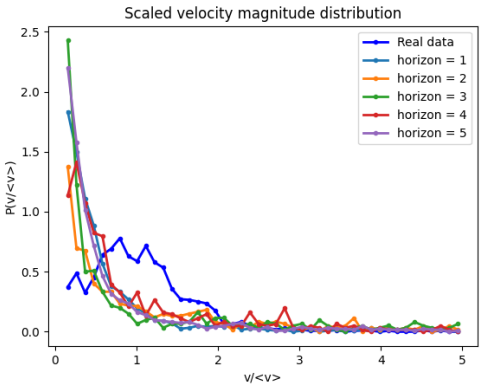
(a) Mean squared displacement



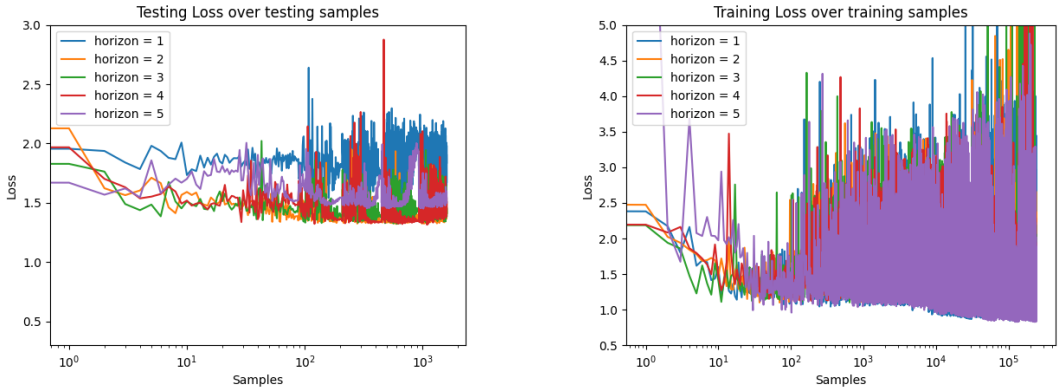
(b) Mean velocity



(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution

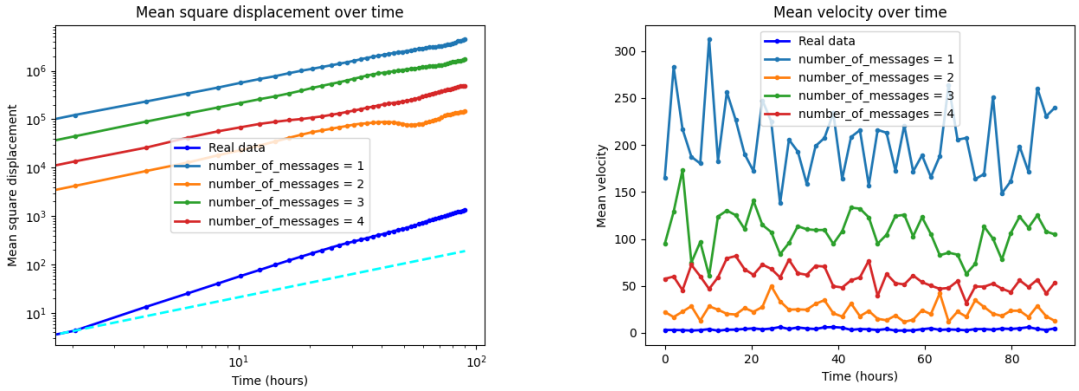


(e) Testing loss (f) Training loss

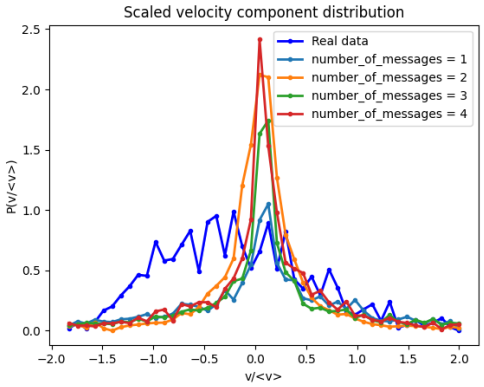
Figure 31: Tuning of the horizon for the real data set

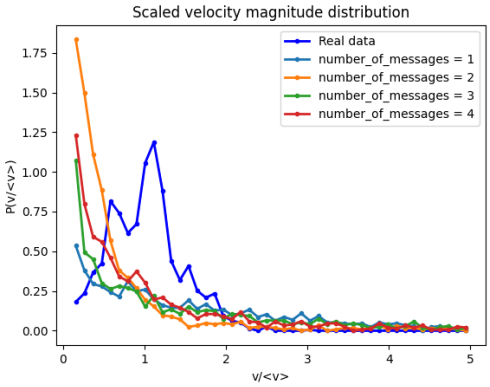
Once again, the results differ a lot from the synthetic data and having an horizon of 4 leads to systematically better results under all metrics.

A.5.3 Number of messages

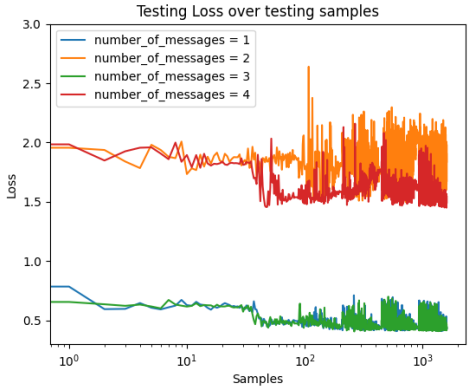


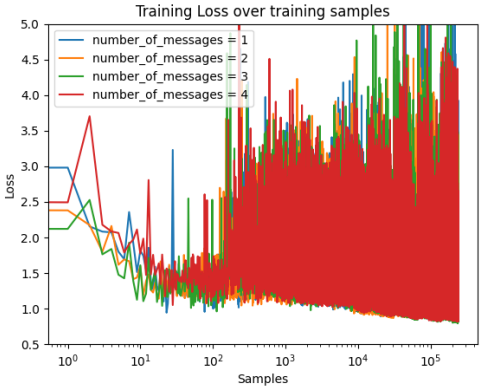
(a) Mean squared displacement (b) Mean velocity





(c) Scaled velocity component distribution (d) Scaled velocity magnitude distribution





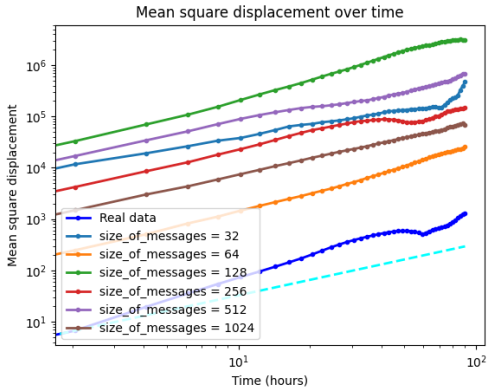
(e) Testing loss (f) Training loss

Figure 32: Tuning of the number of messages for the real data set

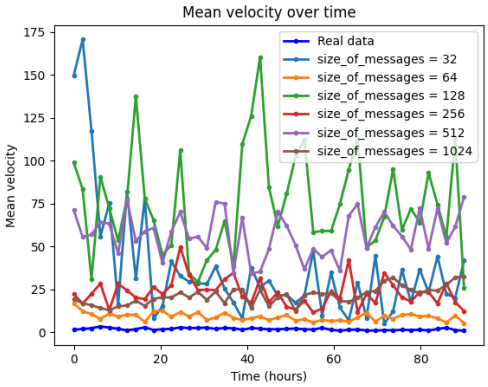
For the number of messages, it seems the SVCD and SVMD are better for the value of 1, the MSD and mean velocity are closer when the value is 2.

Strangely, the results in the testing loss are highly dependant on the initialization of the model.

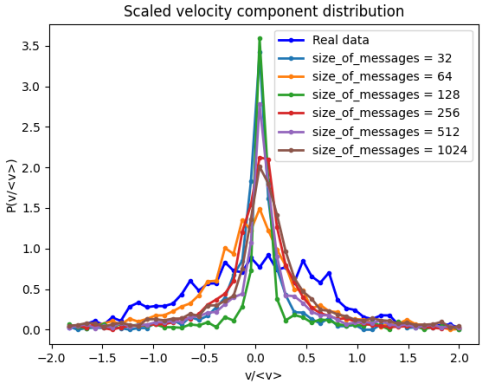
A.5.4 Size of messages



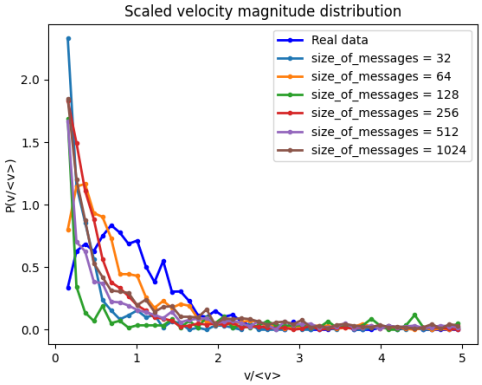
(a) Mean squared displacement



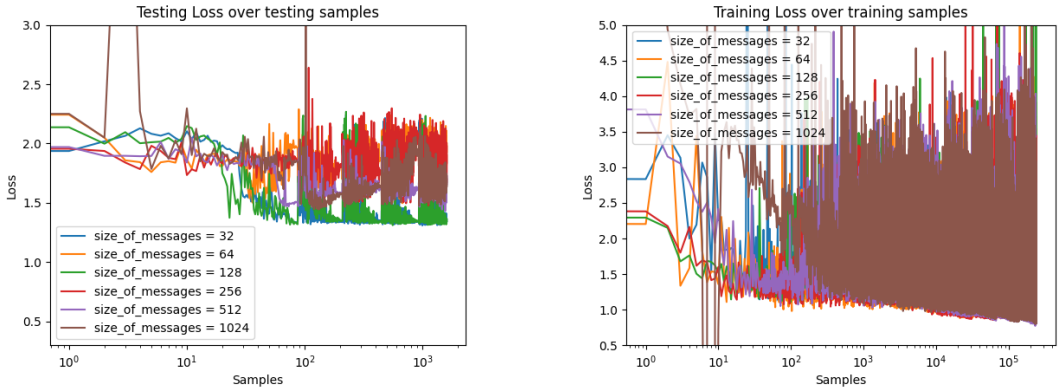
(b) Mean velocity



(c) Scaled velocity component distribution



(d) Scaled velocity magnitude distribution



(e) Testing loss (f) Training loss

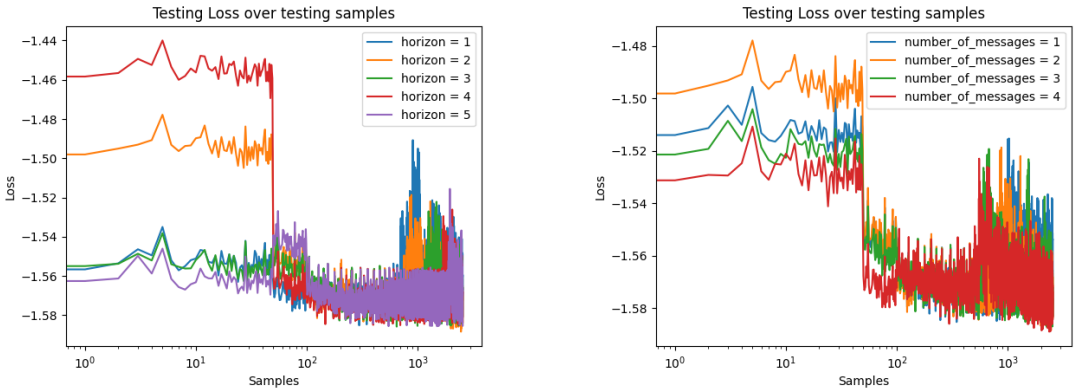
Figure 33: Tuning of the size of messages for the real data set

The testing losses end up better for $size = 32$ and $size = 128$ although all other metrics seem to indicate $size = 64$ comes on top.

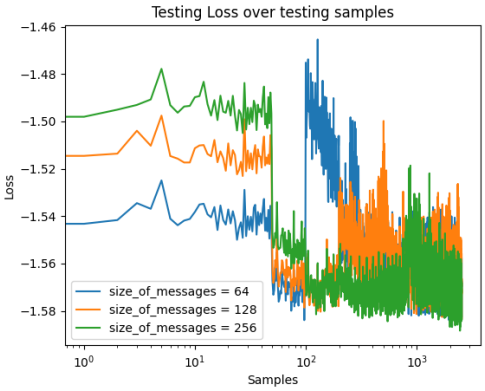
A.6 Tuning on "low tau low velocity"

Since the number of channels has been found as a clear winner, together with laplace distribution, it felt unneeded to tune it for other settings.

It seems the learning is a lot slower for *lt lv* in particular, increasing training time may slightly change the outcome. However, even doubling it lead to no noticeable improvement, the testing loss hovers -1.572 also.



(a) Tuning for horizon (b) Tuning for number of messages



(c) Tuning for size of messages

Figure 34: Tuning of lt lv

Given fig. 34, it may be the case that a greater number of channel and a greater horizon could lead to better performances, though it might be very slim.

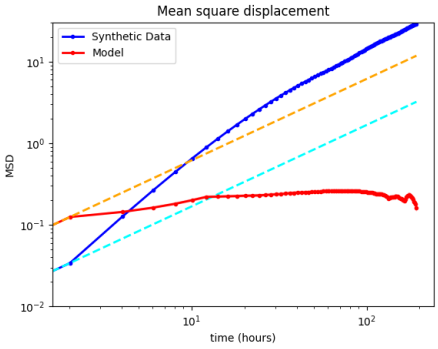
A.7 Many to Many results

In fig. 35 appear the experiment of using the model in a sequence to sequence, many to many fashion.

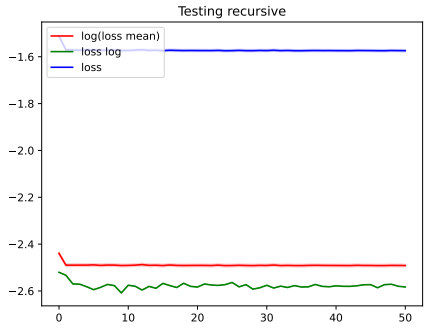
For the same number of samples, this method requires about 4 times the training time and yields poor results in terms of summary statistics.

The testing loss is around the same value of -1.572 as for the other methods, which is remarkable given that all cells barely move and seem to stay in place.

Perhaps it would lead to better performances for other settings where it would be forced to move, though a method that is even more susceptible to the setting is less preferable for the purpose of this thesis.



(a) Mean square displacement



(b) Average over 50 testing samples of the testing loss per epoch ($\sim$ 900 training samples)

Figure 35: Many to many, *lt lv* setting, $horizon = 6$