ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

A STUDY OF TDA METHODS FOR SCRNA VELOCITY

LABORATORY FOR TOPOLOGY AND NEUROSCIENCE

Jade Therras, supervised by Kelly Maggs

Contents

1 introduction **1**

2 Biological background : scRNA & RNA Velocity **3**
2.1 scRNA . . . 3
2.2 RNA velocity . . . 4
2.3 Computing RNA velocity in high dimension . . . 4
2.4 The difficulties with RNA velocity . . . 6

3 The UMAP algorithm **7**
3.1 what is the UMAP algorithm ? . . . 7
3.2 Weighted undirected graph construction . . . 7
3.3 spectral embedding and layout algorithm . . . 10
3.4 optimization of the graph layout . . . 10
3.5 implementation of the optimization algorithm . . . 14

4 Study of the influence of parameters in UMAP **16**
4.1 Parameters in UMAP . . . 16
4.2 The number of neighbors K - weighted directed graph . . . 16
4.3 Gradient descent parameters . . . 17
4.4 forces parameters . . . 19
4.5 Other useful parameters . . . 21

5 The scVelo embedding algorithm **22**
5.1 what is the ScVelo algorithm ? . . . 22
5.2 Markov chain . . . 22
5.3 The velocity graph . . . 22
5.4 scVelo Markov chain . . . 24
5.5 low-dimensional velocity embedding . . . 25

6 Study of the influence of parameters in the scvelo algorithm **26**
6.1 Parameters in ScVelo . . . 26
6.2 The number of neighbor K - KNN-graph . . . 26
6.3 number of PC to find the K neighbors . . . 27
6.4 the sensitivity parameter . . . 28

7 Improving the velocity embelling on UMAP **29**
7.1 The main embedding problem on UMAP . . . 29
7.2 learnable UMAP . . . 29
7.3 Encoding and Decoding Point Clouds . . . 31
7.3.1 learnable UMAP vs UMAP . . . 31
7.3.2 PCA vs Parametric UMAP . . . 38

7.4 Encoding and Decoding Velocity Vectors 43
7.4.1 Jacobian encoding and decoding methods 43
7.4.2 jacobian method on PCA vs learnable UMAP 45
7.4.3 jacobian method vs scVelo method on PCA 49
7.4.4 jacobian method vs scvelo method on UMAP 52
7.5 conclusion 53

8 general conclusion **55**

Abstract

Reducing and visualizing high dimensional data is a complex challenge, yet is an essential element of research in computational biology. In this paper, we will examine a particular type of high dimensional data called RNA velocity, which consists of a point cloud of single cell RNA expression values along with RNA velocity vectors. Visualizing RNA velocity in low dimensions requires a method to embed both the point cloud and the velocity vector field.

UMAP is the standard method for embedding the point cloud, and for the vector field, the predominant algorithm is the ScVelo algorithm. These methods experience two main problems. Firstly, the two embeddings depend on many arbitrary parameters, where it is difficult to get the best combination for visualization. Secondly, we have no way to quantify the distortion of the data by the embedding. Accordingly, this report presents how the RNA velocity embedding works, and make suggestions for its improvement.

In this paper, we explore the influence of parameters in UMAP and the ScVelo algorithm in both biological and synthetic datasets. In particular, we provide examples where a small change in parameters can reverse the vector field embedding. As a comparison, we tested learnable UMAP, a version of UMAP with learnable parameters, and find that it outperforms UMAP in low-dimension synthetic point cloud examples.

To quantify the distortion and test the accuracy of our methods, we choose to embed the data back from low to high dimensions. The ability to reverse the embedding provides a quantification of the loss of information and will be poor in quality in distorted regions. Further, we use the Jacobian of the inverse function from learnable UMAP to embed velocities back into high dimensional space.

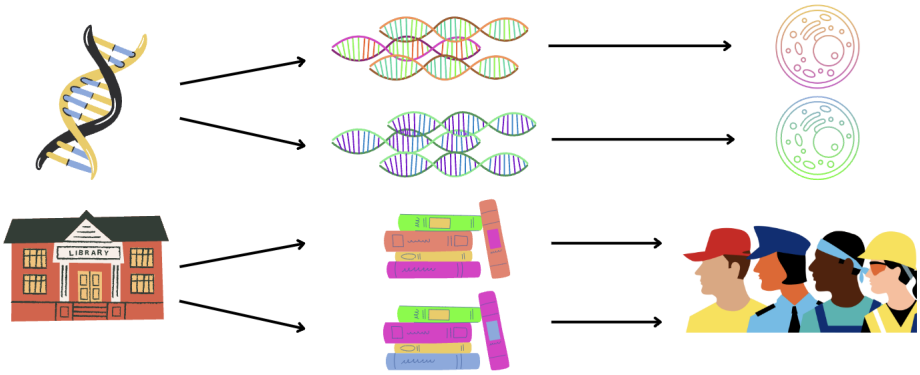
In the future, we envision that our method could be used to detect highly distorted regions of the vector field by current embedding methods.

Keywords: RNA velocity, high dimensional data embedding, UMAP, learnable UMAP, ScVelo.

PART 1

INTRODUCTION

A lot of studies require exploring what happens in a cell. In a multicellular organism such as a Human, cells have a similar DNA but use it in various manners. DNA is like a dictionary, where each cell can find the word needed for its functioning. Thus DNA is transduced to produce RNA, which is matured and traduced to produce protein. Therefore, RNA-sequencing is a method to characterize genetic activity.



[1]

In the past, measuring the genetic activity of single cells was difficult because a cell wasn't providing enough RNA to be able to sequence it. Experience can be done in a mix of a cell, resulting in an average expression of each cell. Unfortunately, even in the same cell type, each cell has a particular expression. This constatation is especially important when studying cellular differentiation. Cells come from the same cell type and evolve in different branches. Single-cell RNA sequencing – or scRNA seq – has been developed to overcome this situation. This method can distinguish which initial cell RNA comes from.

This represents a lot of data for each cell and finding which individual is close to another is a huge challenge. We need to find a method to keep the topology of the data in the high dimensional space to low dimensional space, typically $\mathbb{R}^2$.

Uniform Manifold Approximation and Projection technique – or UMAP – is a dimension reduction method, such as PCA or t-SNE. [2] Is goal is to reduce the dimension of a set of data while keeping is high dimension topology.

When we process cells for single-cell sequencing, the cell is killed so we can't observe the evolution of the cell. ScRNA velocity allows predicting the future of a cell, by distinguishing between mature

and immature RNA after sequencing. Since this velocity is also in the high dimension space, we need a method to compute it's low dimensional embedding without losing the correct direction of the vector field. ScVelo algorithm can compute the vector field (for dynamical or steady-state) and translate it into the low dimension space, [3].

Instead of using a neighbors graph from high to low dimension, we wanted to relate on a mapping of all the high dimensions by a function. learnable UMAP minimizes the same objective function as UMAP but learns the relationship between the data and embedding using a neural network, rather than learning the embeddings directly. This provides an explicit function from high to low dimensions. We designed and tested a new embedding method based on the gradient of the deterministic function from learnable UMAP, called the Jacobian gradient method. We also provided another algorithm that approximates the result of the Jacobian gradient method by using the parametric function to embed the starting and the ending point of the velocity vector.

To quantify the distortion and test the accuracy of our methods, we choose to embed the data back from low to high dimensions. The ability to reverse the embedding provides a quantification of the loss of information and will be poor in quality in distorted regions. We used the inverse learnable UMAP to embed back the point cloud. This method can either learn a neural network that minimizes the reconstruction loss, the parametric decoder, or use the same method as UMAP to reconstruct the data, the non-parametric decoder. Finally, we used the gradient of the inverse function from learnable UMAP, as for the Jacobian method, to embed back velocities.

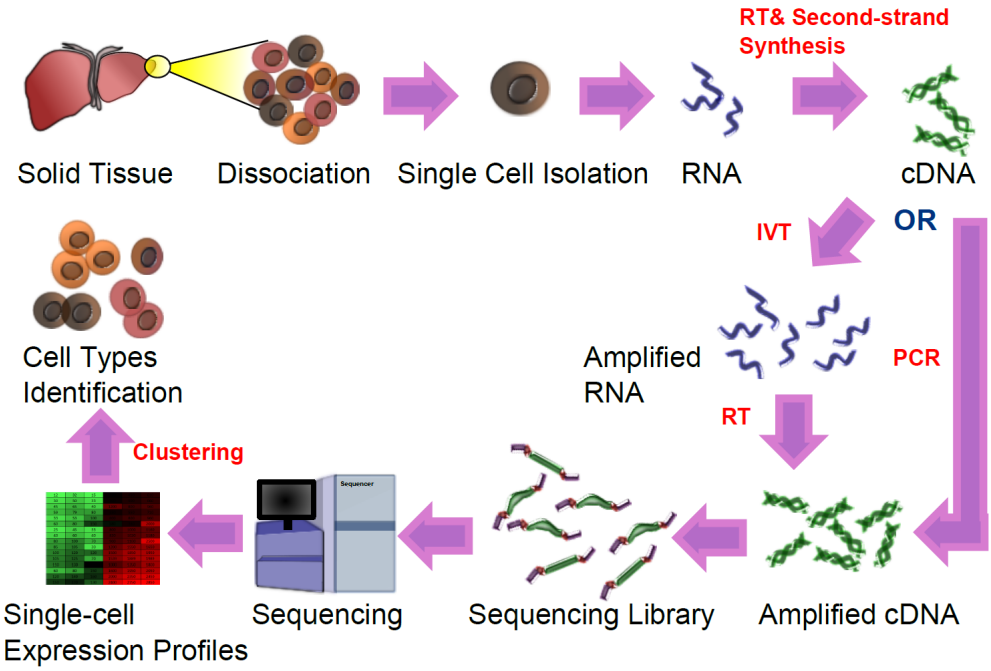
All the code used in this study can be find on github [4]

Part 2

Biological background : scRNA & RNA Velocity

2.1 scRNA

As said in the introduction, single cells RNA sequencing is a method to study individually the RNA expression in a cell.



[5]

The process begins with a pool of cells or tissue. Cells are dissociated and isolated. A solution with a lysis buffer and a specific marker is added. The cell lyses, cDNA with a cell-specific sequence is produced from the resulting RNA pool. (cDNA is reverse transcripted RNA with DNA bases ATGC). All the cDNA obtained is mixed together, amplified for example with PCR, and sequenced. Using the specific marker, the sequencing is identified for each cell. spliced and unspliced RNA (ie. mature and immature RNA) are also distinguished.

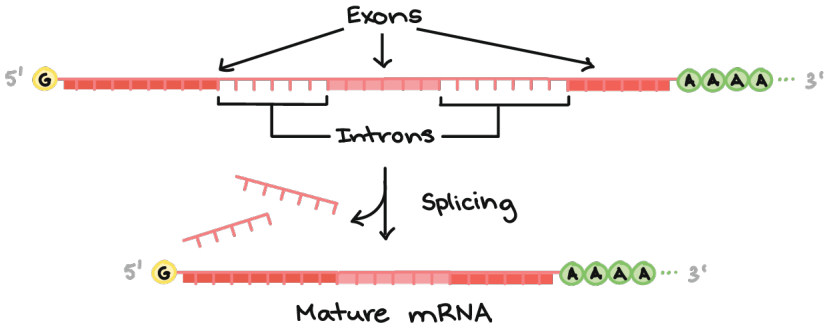
This produces the original dataset. Each cell is a point, each gene is a dimension. The position of a cell is determined by the current genetic expression.

Given the number of spliced and the number of unspliced RNA for one gene, how can we predict the evolution of the genetic expression?

2.2 RNA VELOCITY

A lot of RNA molecules are produced in the nuclei of a cell. mRNA is a part of it, intended to be the template for proteins.

Just after his transcription, mRNA is unspliced. The molecule undergoes some post-translational modifications, such as capping, splicing, and polyadenylation, to produce a mature spliced mRNA. These two types of mRNA can be differentiated. For example, by detection of the cap or the poly-A tail.



[6]

Since the unmatured RNA pool should in the future become the mature RNA pool (by approximation), we can compare the amount of unmatured and matured RNA for a specific gene to measure its evolution. In other words, if we have few matured RNA from a gene but a lot of unmatured RNA, we can suppose that the associate protein concentration will increase in the future of the cell. This consideration ultimately provides a "time scale" of the cell. By doing this process for all genes, we can "predict" the future state of a cell. [3]

Now, we can associate a vector to each cell, with the start point as the current state and the endpoint as the "future" state. With vector is called RNA velocity.

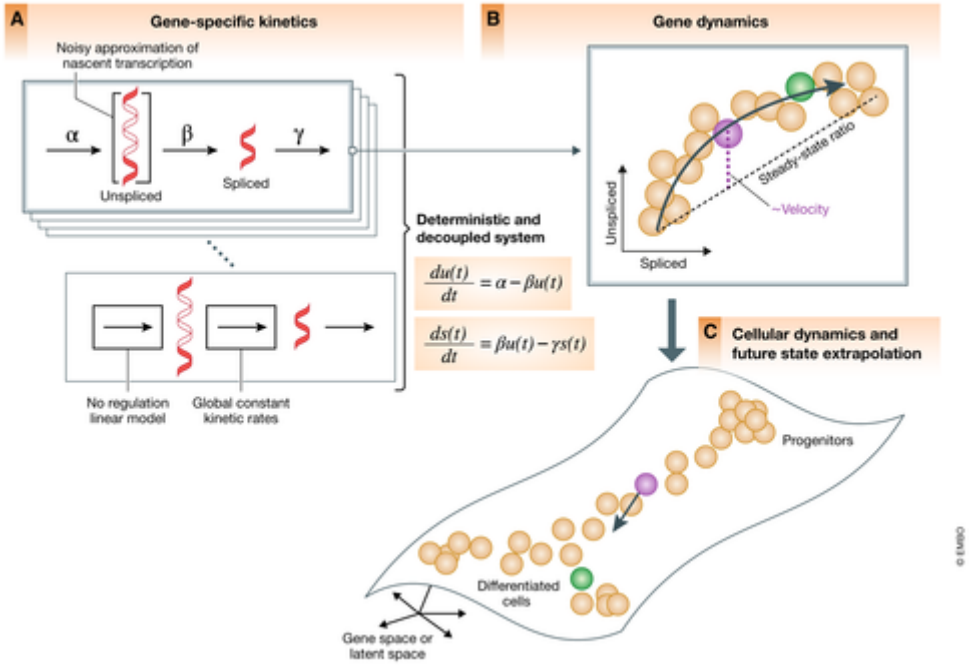
2.3 COMPUTING RNA VELOCITY IN HIGH DIMENSION

ScVelo is able to compute the velocity for each cell by distinguishing between unspliced pre-mRNAs and mature spliced mRNAs.

Knowing this principle, scVelo computes an approximation of the future of the cell. The velocities can be computed with three different models. [7]

- the deterministic model, using the deviation of the observed ratio from its steady-state ratio (or steady-state residuals). This method supposes a common time for going from unmatured to matured RNA.
- the stochastic model, using first and second-order moments. This method uses the Markov chain to estimate the probability of going from unmatured to matured RNA for a specific gene, taking into account its variation.

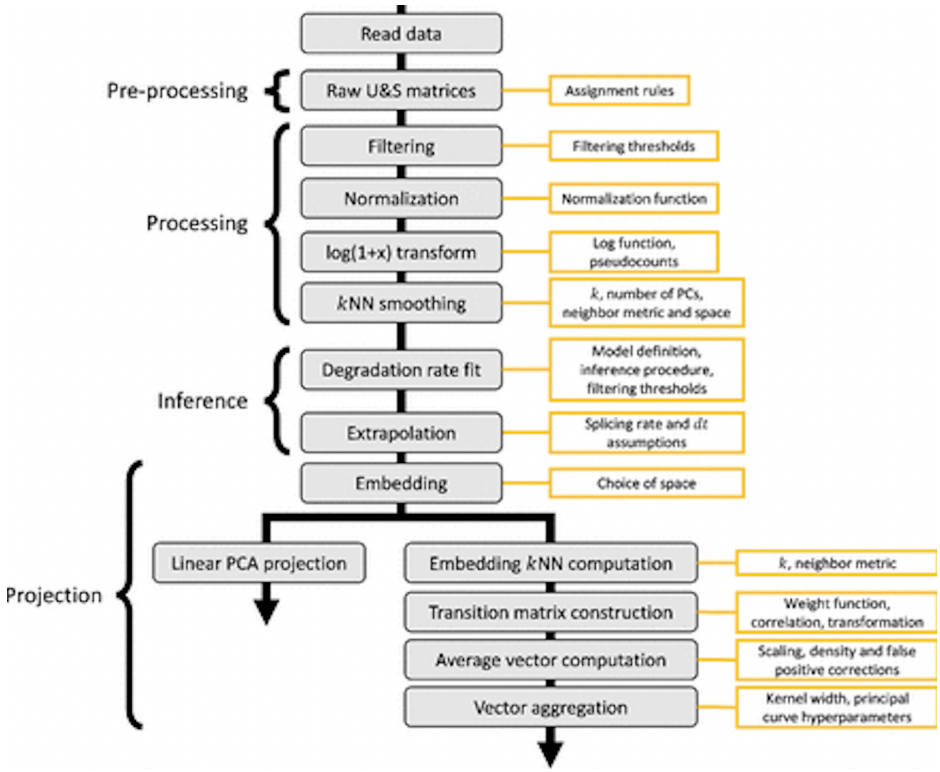
- the dynamical model using a likelihood-based framework. This method computes the full splicing dynamics for each gene and the unspliced/spliced phase trajectory, to estimate the RNA velocity.



[8]

Finally, in the high dimensional space, each cell is associated with its velocity vector.

2.4 THE DIFFICULTIES WITH RNA VELOCITY



[9]

Before mapping RNA velocity, the data is processed in a lot of ways that implicate a lot of choice of parameters and methods. Moreover, some genes present a lot of spiced RNA but only a few become mature RNA. In addition, the spicing process can change and one unspliced RNA can produce a different sequence of mature RNA in variable quantity.

The embedding process is part of this huge challenge of making RNA velocity meaningful.

PART 3

THE UMAP ALGORITHM

3.1 WHAT IS THE UMAP ALGORITHM ?

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique based on K-nearest-neighbors graph learning. Similarly to t-SNE, it can be used for visualization but also for general non-linear dimension reduction.

The algorithm is founded on three assumptions about the data. [10]

- There exists a Riemannian manifold on which the data would be uniformly distributed.
- The Riemannian metric is locally constant (or can be approximated as such).
- The underlying manifold of interest is locally connected

Preserving the topological structure of this manifold is the primary goal, the embellishing method search for the closest equivalent projection in low dimension.

Let's define for now m the dimension of the high dimension space, n the dimension of the low dimension space, U the number of points in the data (cells), and i and j in $[0, U]$, some index.

3.2 WEIGHTED UNDIRECTED GRAPH CONSTRUCTION

At the beginning, we have a point cloud in high dimension. Since we want to keep a feeling of this structure in low dimension, we need a way to measure and store the distance between data points. UMAP use a weighted directed k-nearest-neighbors graph. The aims of this technicle is, for each point, to store which point are is K nearest neighbors, with a weight for each point. The weight represent the high dimensional distance : the nearest have the largest weight, the farest the lower weight, are it's like if all the other have a zero weight. This graph give a feeling of the high dimensional structure, and can be use after to try to find the most similar low dimension embedding in terms of neighbourhoud.

The construction begins by determining the K nearest neighbors for each point in the high m dimension space. K is the first hyperparameter.

It's not because two point are in the K closest neighbors of a point that they are at the same distance to this point. The closest one in the high dimension needs to have a bigger attraction on this point to be also the closest in the low dimension, and the influence of one point needs to decrease with distance in the high dimension.

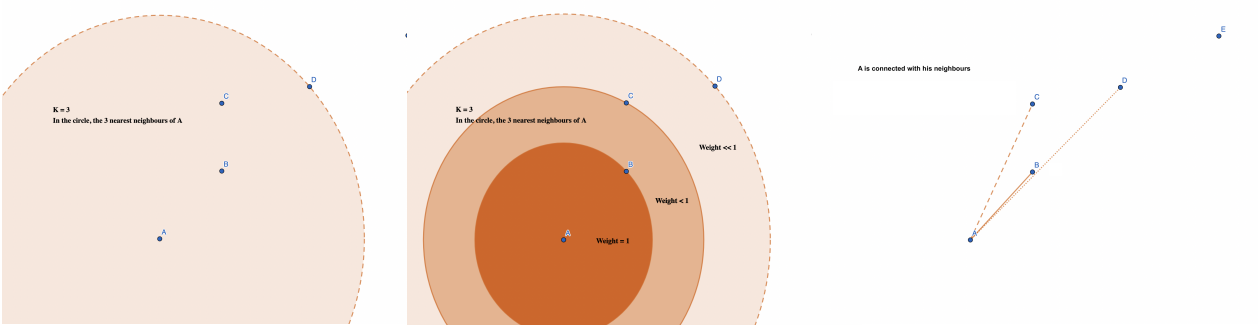
As said before, each point has weight to model this. Let's define the distance between x_i and it's fist neighbour ρ_i, and a normalisation factor σ_i

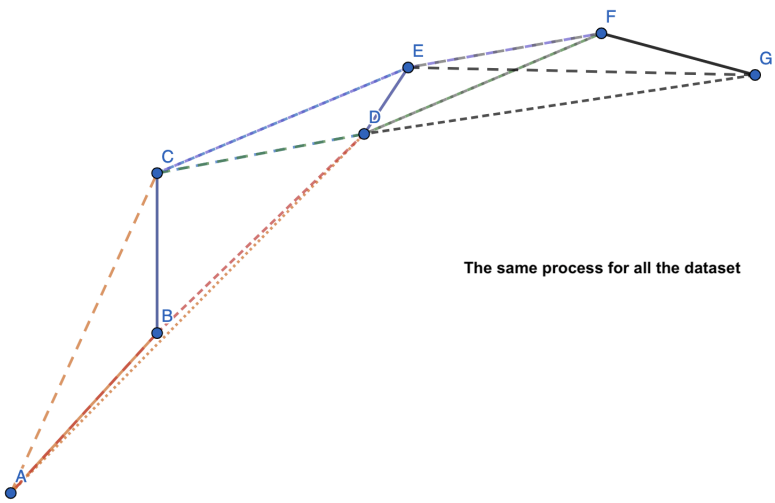
$$\rho_i = min(d(x_i, x_j)|1 \leqslant j \leqslant k, d(x_i, x_j > 0)$$
$$\sum_k exp(\frac{max(0, d(x_i, x_j)\rho_i)}{\sigma_i}) = log_2(k)$$

The weight function is defined as below. The closest point weights 1, and the weight decreases exponentially with the distance in the high dimension space. The normalization factor ensures that some of all the weight for one point is constant.

$$w((x_i, x_j)) = exp(\frac{max(0, d(x_i, x_j)\rho_i)}{\sigma_i})$$

To represent all points with is personal neighbourhood in one graph, a weighted directed graph $G = (V, E, w)$ is define. The vertices of the graph are the set of points. The edges connect each point with is K's nearest neighbors founded before.





[11]

For now, the weighted graph G is directed, meaning that the probability of an edge from vertex i to vertex j isn't the same that the probability of an edge from vertex j to vertex i. For example, in the drawing, we see that the edge from E to C (blue) and the edge from C to E (purple) haven't the same intensity (not the same value). Before working in the low dimension space the graph G needs to be converted into an undirected graph. In a drawing, they will be only one edge between two point, and the lenght represent the probability that at least one of the two edge exist.

Define A the adjacency matrix of G, containing all the weight. The coefficient A_{ij} is the weight $w((x_i, x_j))$. This number represent the probability of a edge from vertex i to vertex j.

Conceptually, $a_{ij} := P(j|i)$.

Define B the adjacency matrix of the undirected graph $\tilde{G}$. The value B_{ij} needs to represent the probability of an edge between i and j, independently of if it's from i to j or from j to i. This consideration imposed $b_{ij} = b_{ji}$, the matrix is symmetric. Therefore, matrix A is transformed into a symetric matrix.

The matrix B is computed as below

$$B = A + A^T - A \circ A^T$$

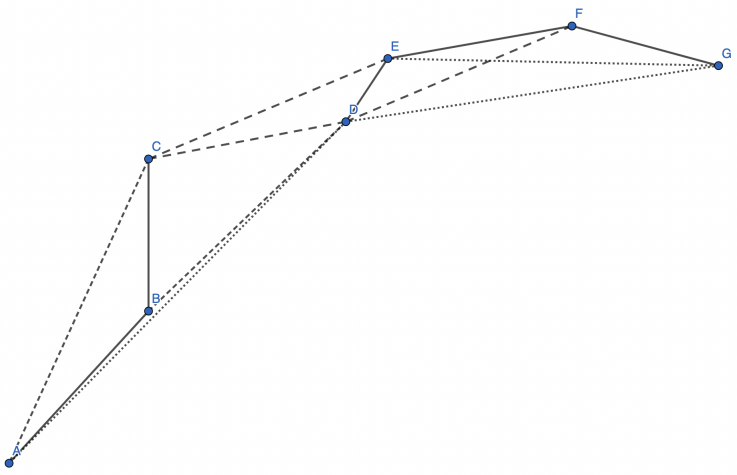
The matrix is symetric,

$$B^T = (A + A^T - A \circ A^T)^T = A^T + A - A^T \circ A = A + A^T - A \circ A^T = B$$

In addition, according to the law of probability,

$$b_{ij} = a_{ij} + a_{ji} - a_{ij} * a_{ji} -> P(j|i) + P(i|j) - P(\{j|i\} \cap \{i|j\}) = P(\{j|i\} \cup \{i|j\}).$$

With U as defined before, the number of points, B is a UxU symmetric matrix. This matrix is used as the adjacency matrix of the undirected weighted graph $\tilde{G}$.[12]



[11]

3.3 SPECTRAL EMBEDDING AND LAYOUT ALGORITHM

The matrix B gives a summing of the geometry of the high dimension space. This matrix will be used to compute the initial embedding of the data in low dimension space : the spectral embedding. The final embedding is then computed by iterating from the spectral embedding, in order to optimize it.

The algorithm takes the dimension of the low dimension space n and the weighted graph $\tilde{G}$. The weighted adjacency matrix B is computed with the 1-skeleton of the weighted graph. The degree matrix D of $\tilde{G}$ is defined as the diagonal matrix containing the degree of each vertex. The degree of a vertex is the number of "connections" with another vertex. In practice, each values on the diagonal of D is computed as the sum of the corresponding row of B (or the column, since B is symmetric) [13]

The Laplacian matrix L is the difference between the degree matrix and the adjacency matrix. This matrix contains the value of the degree on the diagonal and the negatives values of the adjacency matrix. Since matrix B is symmetric, L is also symmetric. This is also a consequence of the undirectonality of the graph.

Note that the algorithm used the symmetric, normalized Laplacian matrix [14], define as :

$$L = D^{1/2}(D - B)D^{1/2}$$

As a first approximation, points connected in the graph need to stay the closest as possible. Thus, we want to minimize the equation

$$\sum_{i,j} ||(Y_i - Y_j)||^2 b_{ij}$$

where Y_j is the position of the vertex j in the low dimension space, a vector of dimension $nx1$. Define also Y, the matrix with all the data points as columns, with dimension nxU.

$$\sum_{i,j} ||(Y_i - Y_j)||^2 b_{ij} = \sum_i Y_i^2 b_{ii} + \sum_j Y_j^2 b_{jj} - 2\sum_{i,j} Y_i Y_j b_{ij} = trace(Y^T L Y)$$

Thus minimizing $\sum_{i,j} ||(Y_i - Y_j)||^2 b_{ij}$ is equivalent to minimizing $trace(Y^T L Y)$.

A solution is given by the eigenvectors with the minimum eigenvalues of $LY = \lambda * DY$. In order to prevent the solution to collapse into a subspace of dimension less than n, impose the orthogonality $Y^T DY = I$ The final equation to minime give

$$Y_{init} = argmin_{Y^T DY = I} trace(Y^T L Y)$$

Finally, the spectral embedding is computed as the n first eigenvectors of the Laplacian matrix. The order of the eigenvectors is computed according to the eigenvalues, where the first eigenvector have the smallest non zero eigenvalue. [13]

For example, to reduce the data into $\mathbb{R}^2$ the first and second eigenvectors will be used.

This projection gives the first representation in low dimension space, ready for optimization. [12]

3.4 OPTIMIZATION OF THE GRAPH LAYOUT

The UMAP algorithm aims to optimize the representation of the data in the low dimension space, using the undirected weighted graph. Conceptually, the weighted graph gives a representation of distances in

the high dimensional space. A high weight between two vertices mains that they are close in the high dimension space, and vice versa.

The high dimensional representation and the low dimensional representation have share the same 0-simplices. We are comparing the two vectors of probabilities indexed by the 1-simplices, as a measure of similarity. Since they is only two state (the simplex exist or not), they are bernoulli variables. Cross entropy is used.

Cross entropy is a measure of similarity between graphs and can be used as a means of comparison between high and low dimensional representations. The cross-entropy can be computed for all edges to compare the proximity between the two embeddings.

$$\sum w_h(e)log(\frac{w_h(e)}{w_l(e)}) + (1 - w_h(e))log(\frac{1 - w_h(e)}{1 - w_l(e)})$$

with e an edge, $w_h(e)$ the weight in the high dimensional representation and $w_l(e)$ the weight in the low dimensional representation.

A set of attractive and repulsive forces can be derived from the edge-wise cross-entropy between the weighted graph G, and a current solution in the low dimension. Applying these forces is equivalent to trying to minimize the entropy, and can be done using the gradient descent method, as it will be explained after.

In the formula of the cross entropy before, the first term is at the origin of the attractive force, the second term is at the origin of the repulsive force.

The repulsive and attractive forces are applied for each pair of vertices. The formula for each force between two vertices contains a set of hyper-parameters and depends on two measures: the distance in the high dimension space, represented by the weight, and the Euclidian distance in the low dimension space.

For the attractive force

$$F_A(ij) = \frac{-2ab||y_i - y_j||_2^{2(b-1)}}{1 + ||y_i - y_j||_2^2} w((x_i, x_j))(y_i - y_j)$$

Where a and b are positive hyperparameters.

The attractive force increases proportionally with the weight. Additionally, if the two vertices are close in the low dimension space, the attractive force tends to be zero. If they are far, the numerator increases faster than the denominator, and the force grows negatively.

For the repulsive force

$$F_R(ij) = \frac{2b}{(\epsilon + ||y_i - y_j||_2^2)(1 + a||y_i - y_j||_2^{2b})}(1 - w((x_i, x_j)))(y_i - y_j)$$

Where ϵ is a small number to prevent division by zero.

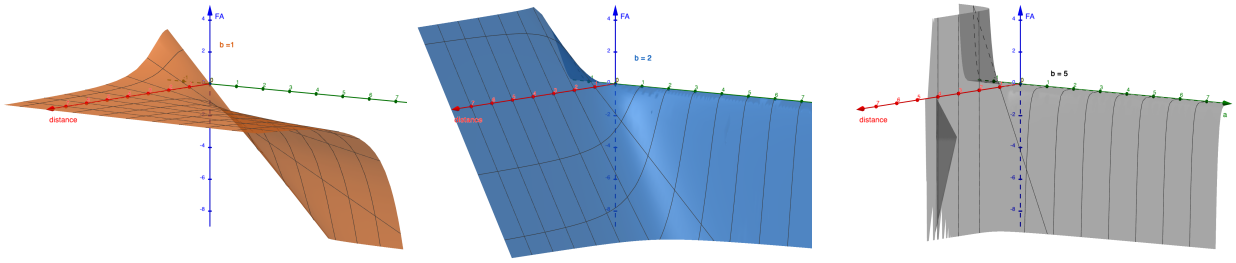
The repulsive force decreases proportionally with the weight. If the two vertices are close in the low dimension space, the denominator tends to be zero, and the repulsive force grows to infinity. If they are far, the denominator grows and the force decrease fastly.

Here is a demonstration of the repulsive and attractive force, as functions of the hyper-parameter a and b. This graph shows the evolution of the part independent of the forces independant the weight (as if the weight was fixed)

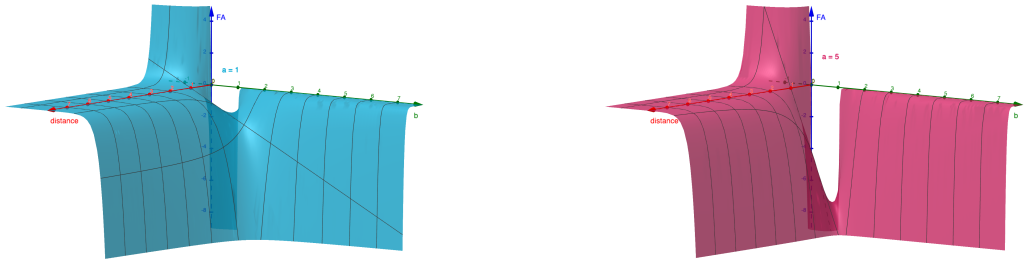
The x axis is the distance between the two vertices, for a different set of parameters. the z axis is the strenght of the force (attractive is negative, repulsive is positive) the y axis can be a or b. The other one is fixed at different values.

For the attractive force

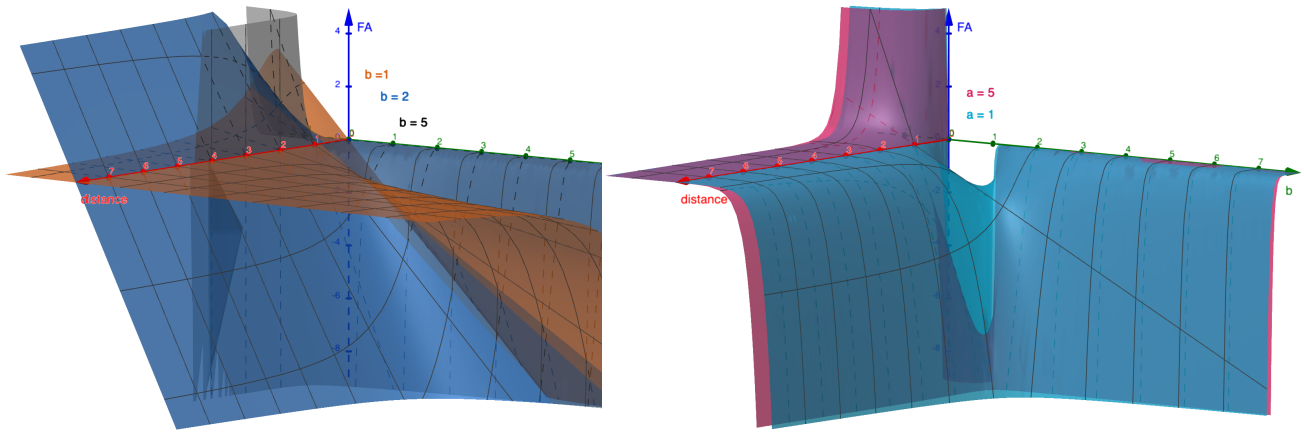
with b fixed at value 1, 2 and 5, a change



with a fixed at value 1 and 5, b change

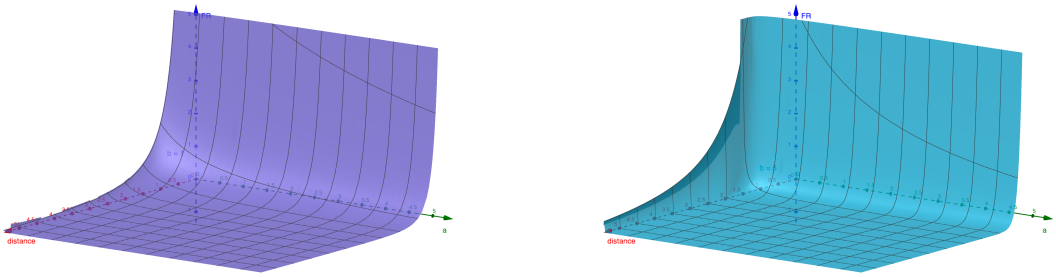


with all in the same graph

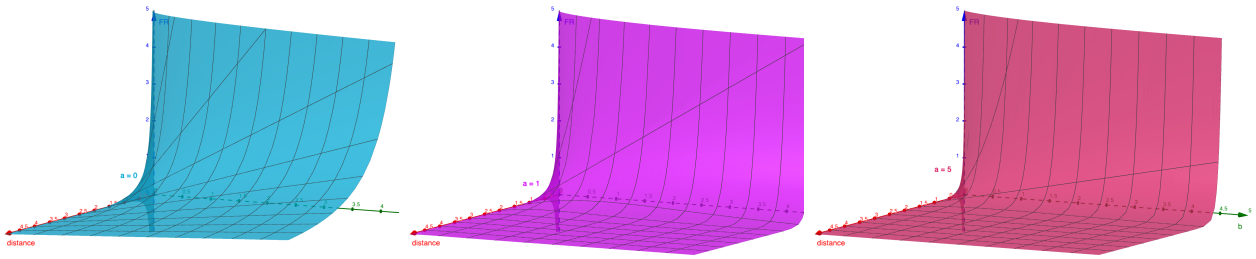


For the repulsive force

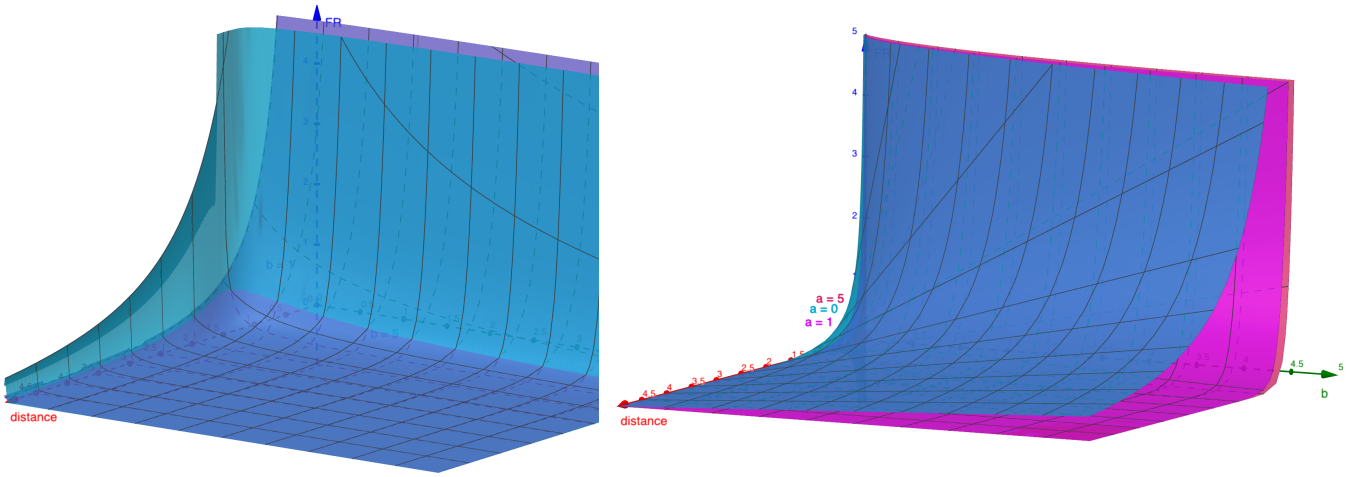
with b fixed at value 1 and 5, a change



with a fixed at value 0, 1 and 5, b change



with all in the same graph



[11]

For the two type of forces, parameter b changes the evolution of the force, which decreases proportionally. Parameter a control the sharpness of this evolution.

For the attractive force

if $b = 1$ the attraction increase when the distance decrease. When the distance is zero, the attraction force reach a constant value of $-2 * a$ If the distance increase, the attraction tend to zero. Then, the attraction force act at low distance and decrease with the distance. If $1 < b < 2$, the attraction is zero when the two point are superposed, and increase with the distance. When two point are far, the attraction force reach a minimum of $-2ab$ (the attraction is maximum). Then, the attraction work better for big distance, and act the same way for all point with a high distance than a treashold. Finally, if $b > 2$, the attraction stay zero for superposed point but increase continuously with the distance. Then, the attraction is even bigger for

far point. As said before, parameter a control the sharpeness of the force. If a increase, the slope of the curve increase.

For the repulsive force

The repulsion always increase to infinity when the distance approch zero, and decrease to zero for far point. Parameter b makes again the evolution of the force more sharp. With a small b, change from high repulsion to zero is more smooth. With a high b, the curve make an elbow. In addition, the parameter b shift the curve, applying even more repulsion between close points. Parameter a also act on the sharpeness of the curve. note that if a is zero and b is big, the slope of the curve decrease.

Finally, note that if a and b are zero there is no attraction and repulsion force. If only b is zero, the result is the same but if only a is zero, there is a repulsion force and no attraction force.

The algorithm tends to find the configuration that minimizes most all the forces. This representation is the final output of the UMAP algorithm. [12]

3.5 IMPLEMENTATION OF THE OPTIMIZATION ALGORITHM

The optimization algorithm starts from the spectral embedding solution and applies forces as described above. In practice, stochastic gradient descent is applied. Stochastic gradient descent is a machine learning method that shifts gradually the solution in the direction of the optimization state.

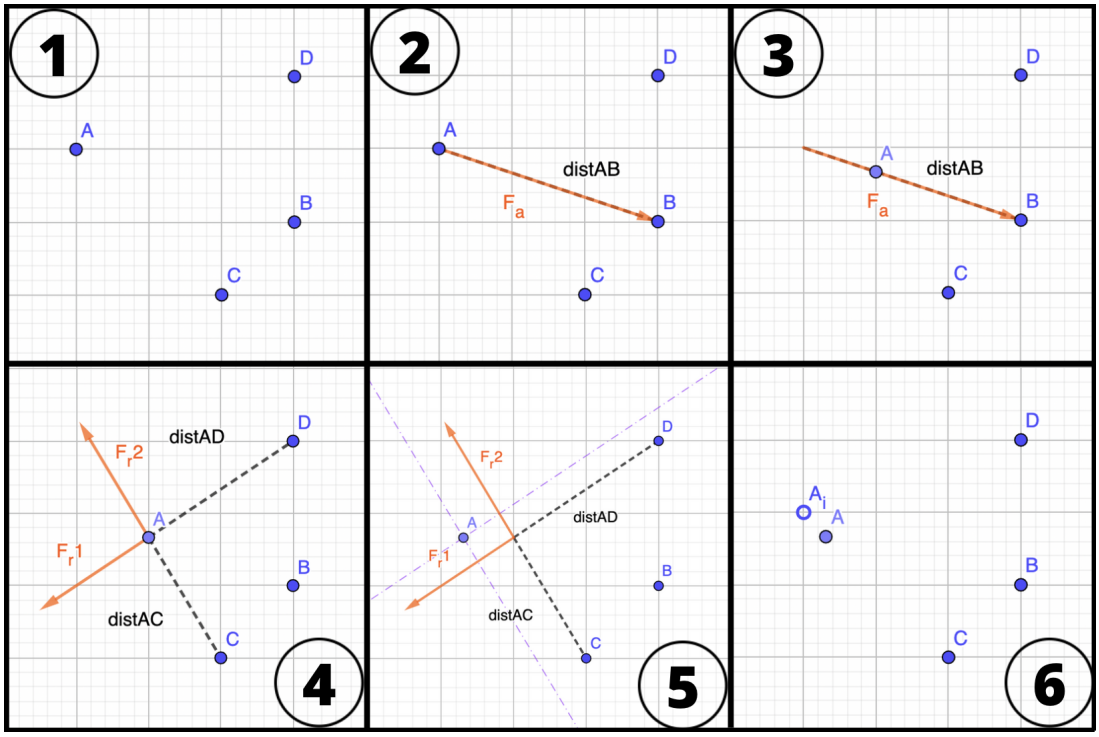
The algorithm is applied step by step. In each step, attractive and repulsive forces are applied to each couple of vertices of the current representation. It's easy to visualize that if we have only two points and an attractive force between them, we would like to make them closer in the next step.

At each step, it's possible to move a little bit at every point in a limited area around the current state. The sum of all forces determines if the movement is favorable or not. This way it's possible to map the region around the current state and find the direction of the minimum of forces. If the distance traveled in each step is enough small, and if the number of steps is enough high, a local optimization solution can be reached. Additionally, the attractive and repulsive forces are slowly decreasing at each step. This solution can be locally the best one but not globally as there are several local solutions. The initialization is important here.

The algorithm takes the graph to optimize, a minimum distance between the vertex (to prevent collapse), and the number of steps for the gradient descent. There are two hyperparameters.

At each step, and for each couple of points A and B, the position of point a is updated with probability p. first, the position of the point is shifted according to the attractive force between A and B. Secondly, a random set of points from Y is selected, and point A is shifted according to the repulsive force between A and each point of the set. The force is scaled by a constant at each step, the algorithm converges to a solution.

The final graph H is the representation of the data in the low dimension.[12]



[11]

1. initial graph (in a random step)
2. the attractive force is applyed on the vertice AB
3. the point A move according to the attractive force
4. the repulsive forces is applyed on a random set of vertices AX
5. the point A move according to repulsive forces
6. resulting graph after processing on one vertice

This process is done on each couple of vertice with probability p, on each step.

PART 4

STUDY OF THE INFLUENCE OF PARAMETERS IN UMAP

4.1 PARAMETERS IN UMAP

Unlike the ScVelo algorithm, the UMAP algorithm isn't deterministic. The final solution is optimized with machine learning, by a stochastic algorithm. This means that two outputs, obtained with the same initial data and parameters, may be different. Therefore, a result isn't reproducible.

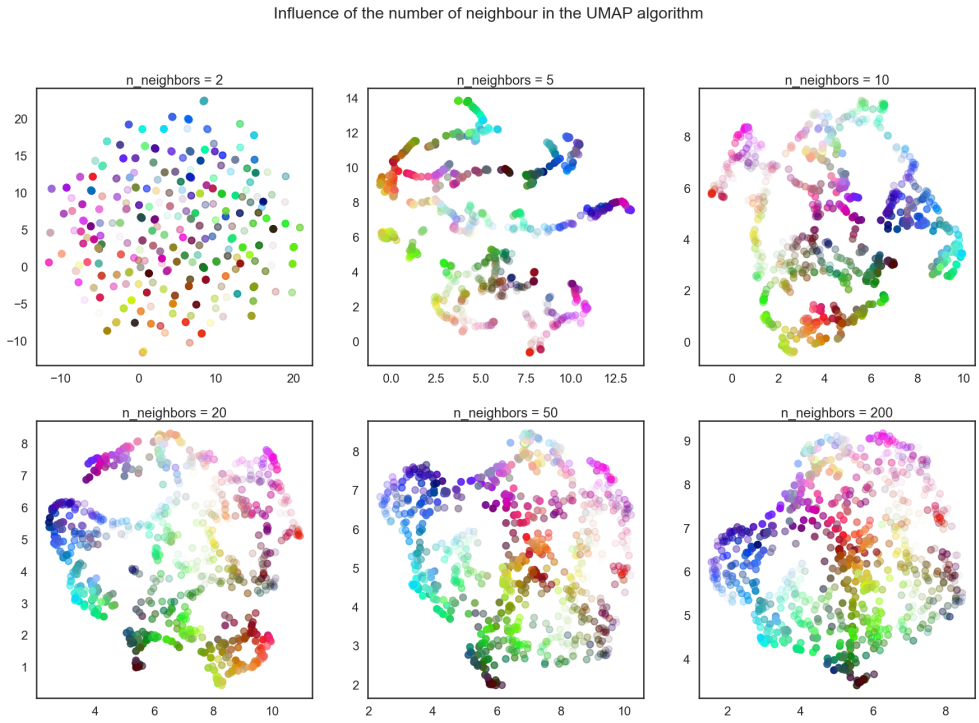
Moreover, the algorithm is influenced by a range of parameters. Some are for the gradient descent algorithm, while others influence directly the expression of forces or the embedding method.

In this section, we tried to understand the influence of the choice of some important parameters in the final representation of the data, conceptually and in practice.

The website of the UMAP algorithm has already done an exploration of some of these parameters. For our purpose, we will use the same data generation and visualization process.[10] The dataset is formed with a random color point. The coordinates of each point are (R, G, B, translucency). Each point has a color and an intensity. The high dimension is 4 and the low dimension is 2. Therefore, a good representation in two dimensions will arrange the point by color and intensity. [10]

4.2 THE NUMBER OF NEIGHBORS K - WEIGHTED DIRECTED GRAPH

In the first step of the UMAP algorithm, the parameter K is used to construct the initial graph. UMAP estimates the structure of the data in high dimensions by looking at the neighborhood of each point.



As we can see, a low K forces the algorithm to look only at the closest neighbors. The final representation will inform about close relationships but lose the global structure of the data.

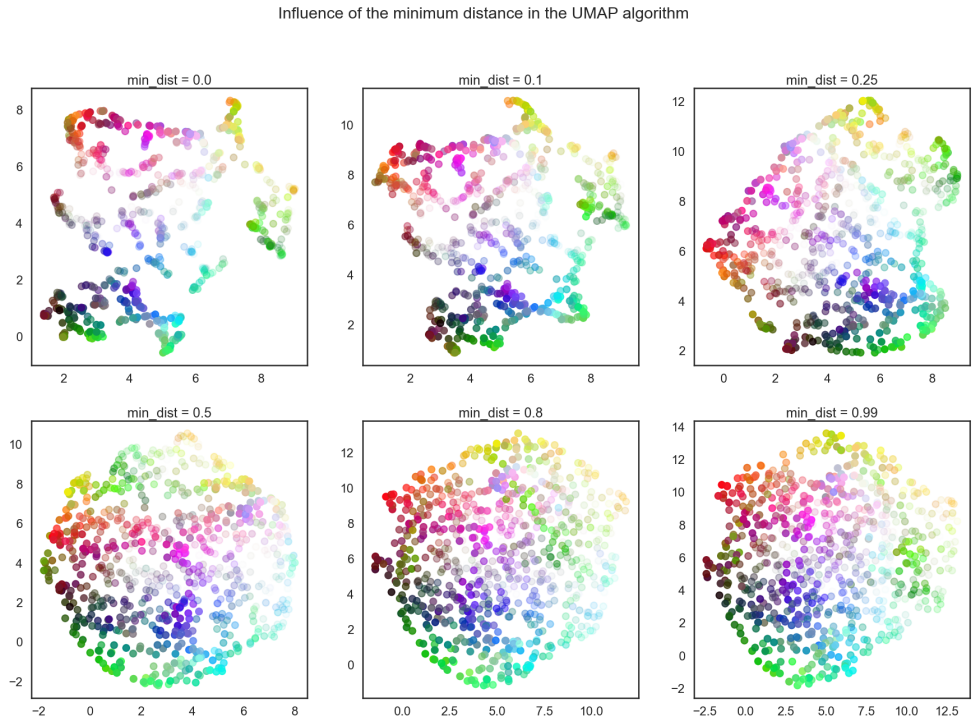
$K = 2$ totally fails to cluster the data. With $K = 5$ close relationships are represented by a "stick" structure, but the algorithm fails to determine the relationship between the different groups. For example, the representation shows a cluster of light blue with a good classification of colors, and a group of dark blue, but no show a relation between them.

A too high K gives the global structure of the data but loses local relationships.

$K = 20$ gives a good representation of the data. With a superior value, the overall structure is well determined.

4.3 GRADIENT DESCENT PARAMETERS

The optimization of the representation is done with stochastic gradient descent. The minimum distance and the number of epochs are important parameters.

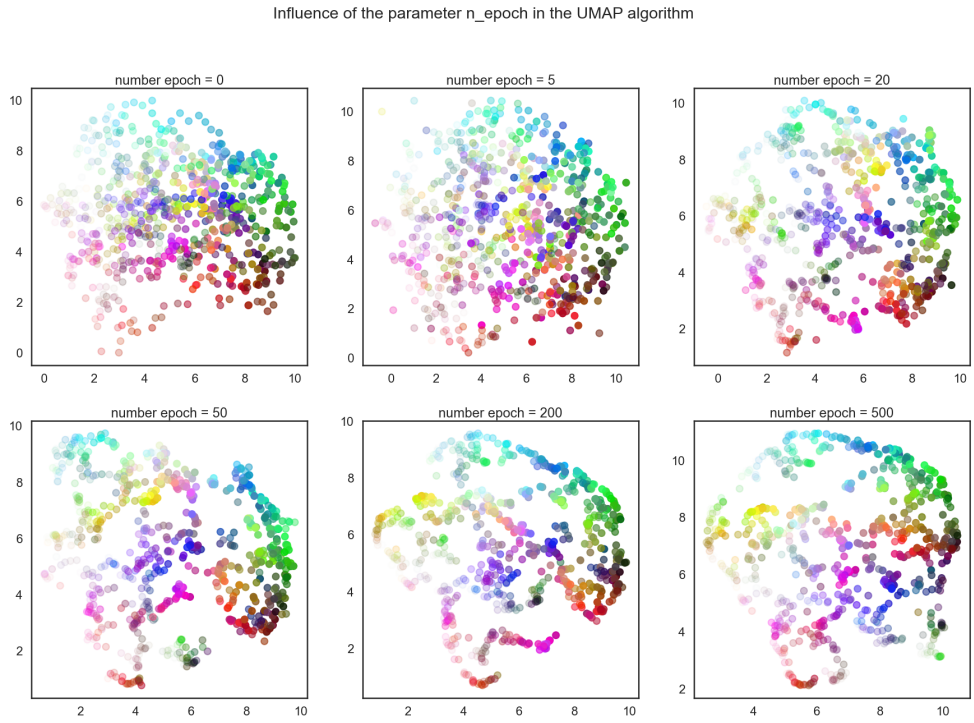


The minimum distance constraint how close data points can be in the low dimension. This parameter provides the data from collapsing.

A low or zero minimum distance keeps the topological structure of the high dimension and permits clustering. A high minimum distance constraint the algorithm and helps to keep the overall structure instead of the local relationship.

As seen, with no minimal distance distinct and unrelated clusters of color are formed. When the parameter increase, a global structure is formed. As data points can't be packed together, the algorithm is forced to deal with other parts of the set, and find a global structure that minimizes the loss without clustering.

Moreover, the gradient descent is processed step by step, moving gradually each datapoint. The number of epochs represents the number of steps of the algorithm before it stops.



The method tries to optimize the cross-entropy, and move in the direction of local minima. The number of epochs represent how many step the algorithm can do.

If the number of epochs is too low, the algorithm has no time to optimize the representation and give a result close to the initialization graph. In the extreme case $n = 0$, the result is the graph obtained from the spectral embedding layout algorithm. As seen, the initialization graph gives an overall but poor structure of the data. With only 5 epochs the representation is more structured.

As the number of epochs increases the algorithm has time to converge to a local minimum. With 50 epochs, the result is quite good with this dataset.

Conceptually the number of epochs can't be too high. The precision increase to one of the local minima and when this state has reached the representation don't move anymore. (The only problem is overfitting the data) Technically, a big epoch requires a lot of computational power and takes a lot of time. In addition, when close to the optimal state the representation does not change a lot. With 200 epochs and higher, the algorithm gives a good representation of the data and stays similar.

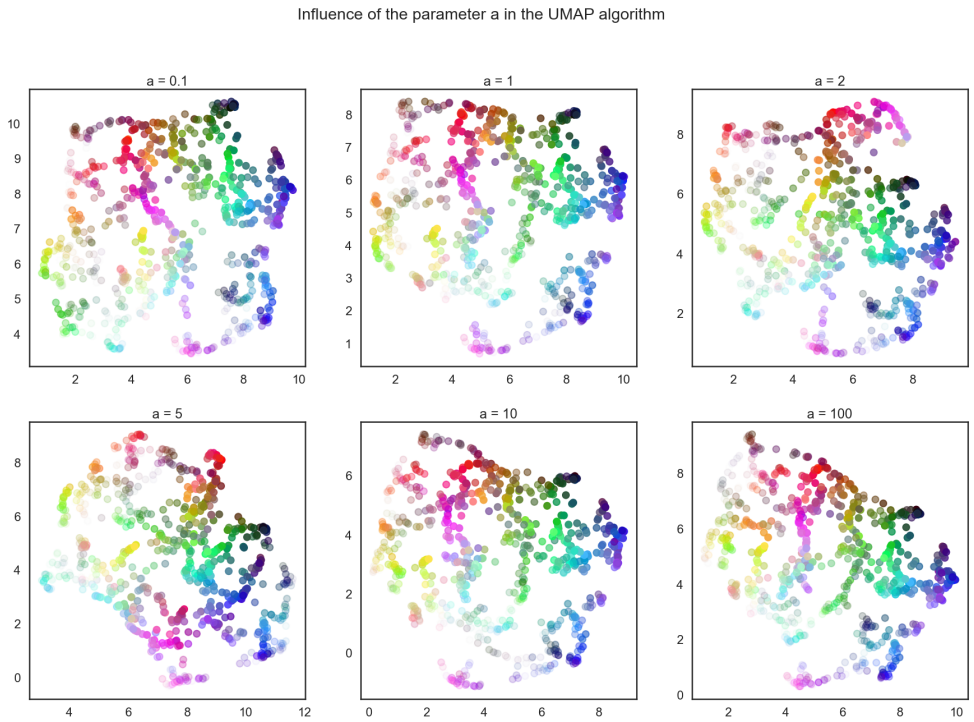
4.4 FORCES PARAMETERS

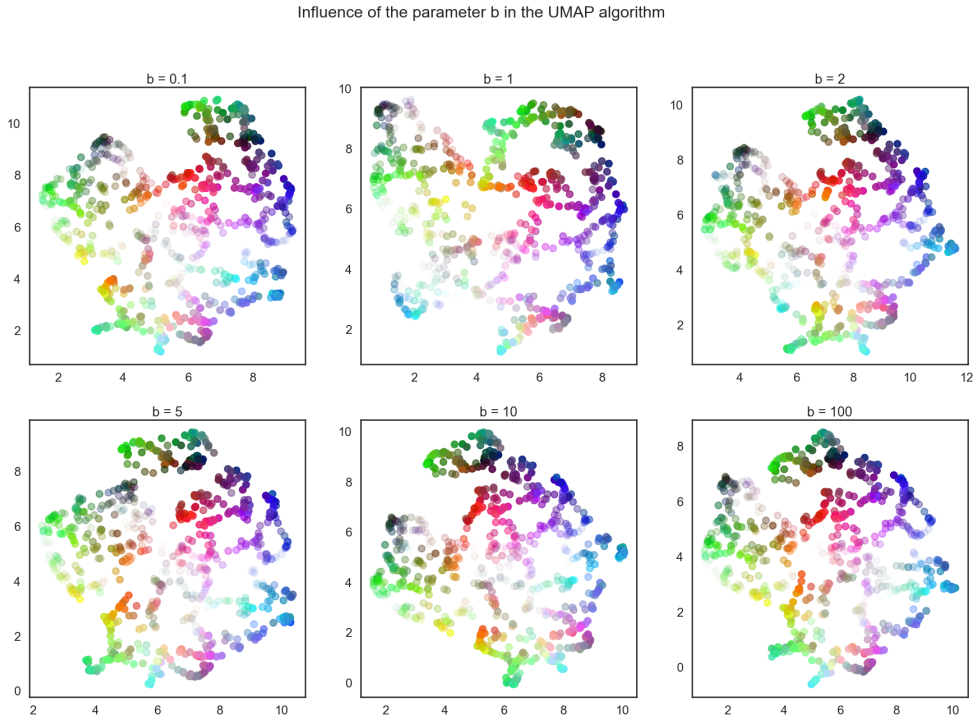
The gradient descent uses a set of forces derivated from cross-entropy. The cross-entropy function includes two hyperparameters that can be modulated.

Parameter a makes force globally bigger.

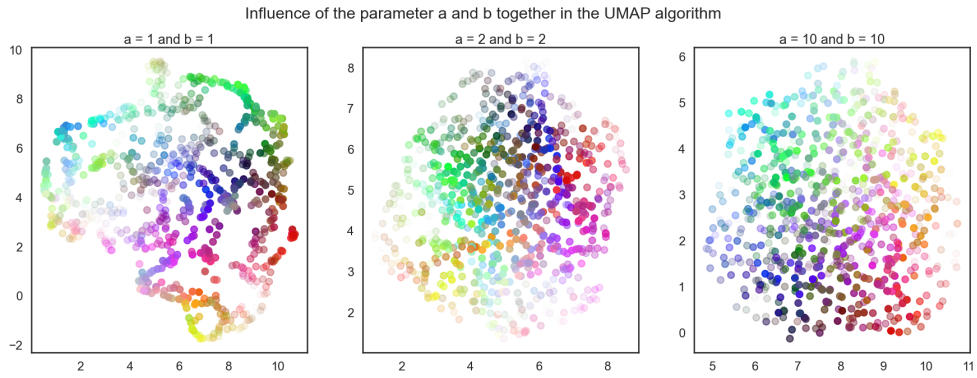
Parameter b makes the force sharper. The forces are not only globally bigger but a little change in the distance between the two points will lead to a big change in force.

This two parameter have been studied conceptually in the last chapter. Here we will test there influence in practice.





Independently, a and b change does not seem to make a big difference in this dataset.



When a and b increase together this difference is clear. Low a and b lead to more clustering and favorise the local topology of the data. High a and b give the overall structure of the data, but a too high value (10 for the two) loses a lot of precise relationships.

4.5 OTHER USEFUL PARAMETERS

A lot of parameters influence the UMAP mapping.

Obviously the final number of dimensions, named "n components" is crucial. This number is determined dependently on the final use of the UMAP projection. It's usually 1, 2, or 3, to be able to plot and visualize the data. Less often the reduction isn't for visualization but just to reduce the weight of the data. The final n component needs to be determined in consequence. For example, use the smallest n where 99 percent of the data is clearly represented (but UMAP is more used for visualization, PCA is usually the one for this purpose)

Another useful parameter is the metric. If not precise, the program state for euclidian metric. Other metrics can be interesting, especially for keeping the topology of the high dimension or we already expect a structure for the high dimension.

Finally, learnable UMAP will be discussed in this paper. This method optimizes the same force as normal UMAP but has the particularity to provide a function from high to low dimension. It's possible to replace UMAP with learnable UMAP and obtain a different result.

PART 5

THE SCVELO EMBEDDING ALGORITHM

5.1 WHAT IS THE SCVELO ALGORITHM ?

The UMAP algorithm permits the reduction of the data's dimension, from high dimensional space to low dimensional space.

However, the velocity vector is initially also in high dimension. The ScVelo embedding algorithm aims to transplant the velocity field into the low dimensional space, to be able to interpret it on the data. As for the data, the difficulty is to keep the topology of the high dimension space.

For example, two cells that were close in the high dimension space need to be close in the low dimension space, and if the velocity vector of one cell point to another in the high dimension space, this has to be the case also in the low dimension space.

5.2 MARKOV CHAIN

In probability theory, the Markov chain is a stochastic model with various states, where the probability of going from an actual state to another (or staying in this state) at a certain time depends only on the actual state.

In discrete-time there it's a sequence of random variables from a countable set S called the state space of the chain, that can be characterise by

$$P(X_{n+1} = X_i | X_n = X_j) \quad X_i, X_j \in S$$

defining for pair of state and time n. The probability of moving can be homogenous in time (independent of n) or not.

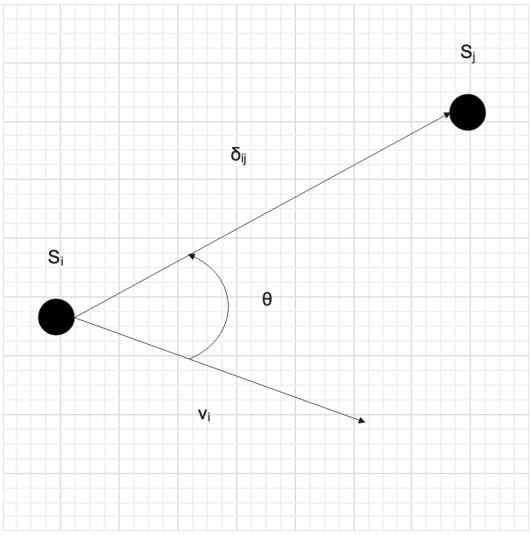
At a step n, we can define the transition matrix as a mxm matrix, with m the number of states, regrouping all probabilities of moving from the state a to state b.

5.3 THE VELOCITY GRAPH

The velocity graph in the high dimensional space gives the similarities between the direction of evolution of cell i and the actual cell j. To reduce the dimension of the space, the velocity is used as a sensor of the probability that each cell moves from the actual state to another state, represented by other cells.

In other words, we can compute the probability for a cell to differentiate in state-like neighborhood cells in the function of the direction of its velocity vector.

- Choose the number of cells to include in cell-to-cell transition pool
- Determine a measure of correlation between cells
- transform into probability to construct the markov chain

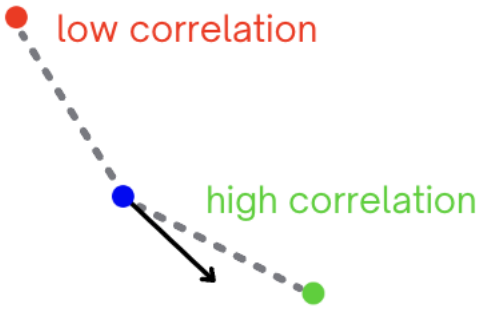


n is the number of cell, i,j $k \in$[0,n].S_i represent the position of a cell i in the high m dimension space and v_i is the velocity vector of cell i, two vector of length m. Finally $\delta_{ij} = S_j - S_i$, the vector of distance from cell i to cell j.

Define the measure of similarity between the direction of evolution of cell i and the actual cell j :

$$\pi_{ij} = Lcos(\delta_{ij}, v_i)$$

The result is between -1 and 1. If $\pi_{ij} = 1$ the velocity of cell i predict a evolution in the direction of cell j (colinearity and same direction of v_i and δ_{ij}), and if $\pi_{ij} = -1$ the cell i evoluate in the opposite direction (colinearity and opposite direction of v_i and δ_{ij}).

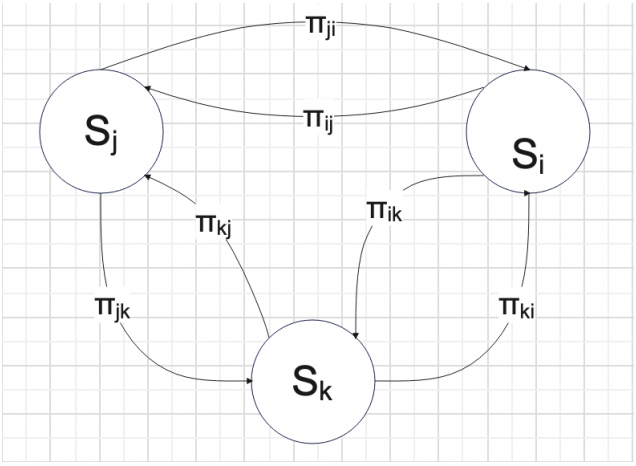


Not only the direction but the distance is important in the possibility of a cell-to-cell transition.

for exemple, we can find the same value for π_{ij} and π_{ik}, but $|\delta_{ij}| << |\delta_{ik}|$. We can suppose that the probability for cell i to do a transition into cell k his lower than the probability to do a transition into cell j.

The velocity graph is a KNN-graph, constructed with the similarity measure π computed on the K first neighbors of each cell.

The number of neighbors K is, therefore, an important parameter.



5.4 SCVELO MARKOV CHAIN

To construct a Markov chain, the values on the velocity graph π need to be transformed into probabilities: the sum of all possible events for one cell needs to be one and all the values need to be positives.

A exponential kernel is applied.

$$\tilde{\pi}_{ij} = \frac{1}{z_i} exp(\frac{\pi_{ij}}{\sigma_i})$$

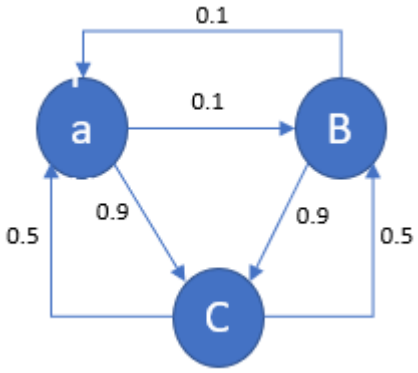
the exponential function turn all value positive and

$$z_i = \sum_i exp(\frac{\pi_{ij}}{\sigma_i})$$

is the normalisation factor.

σ_i is a sensibility parameter, computed individually for each cell. More σ_i is high more the difference between values in the velocity graph is attenuated, which prevents probabilities to be too extreme.

At this step, the velocities can be represented by the Markov chain, as probabilities to go into one or another cell from a particular cell. To be able to interpret it with the data in the low dimensional space, the reverse process has been done. From the Markov chain, velocity vectors in low dimensional are computed.



$$P = \begin{bmatrix} 0 & 0.1 & 0.9 \\ 0.1 & 0.0 & 0.9 \\ 0.5 & 0.5 & 0.0 \end{bmatrix}$$

5.5 LOW-DIMENSIONAL VELOCITY EMBEDDING

After reduction of dimension with a algorithm such as UMAP, $\tilde{S}_i$ represent the position of a cell i in the low (typically R^2) dimension space.

Define the normalised direction vector from cell i to cell j.

$$\tilde{\delta}_{ij} = \frac{\tilde{S}_j - \tilde{S}_i}{||\tilde{S}_j - \tilde{S}_i||}$$

The velocity vector for a cell is constructed as a linear combination of the direction of its neighbors, with the probabilities of the Markov chain used as weight. More the probability is high the corresponding direction will influence the final velocity vector.

$$\hat{v}_i = \sum_j \tilde{\pi}_{ij} \tilde{\delta}_{ij}$$

The velocity field has been reported in the low dimension space.

PART 6

STUDY OF THE INFLUENCE OF PARAMETERS IN THE SCVELO ALGORITHM

6.1 PARAMETERS IN SCVELO

The ScVelo algorithm is deterministic, meaning that with the same condition (parameters and dataset) we find always the same solution.

However, the algorithm is influenced by a range of parameters. In this section, we tried to understand the influence of the choice of parameters in the final representation of the data, conceptually and in practice.[7]

6.2 THE NUMBER OF NEIGHBOR K - KNN-GRAPH

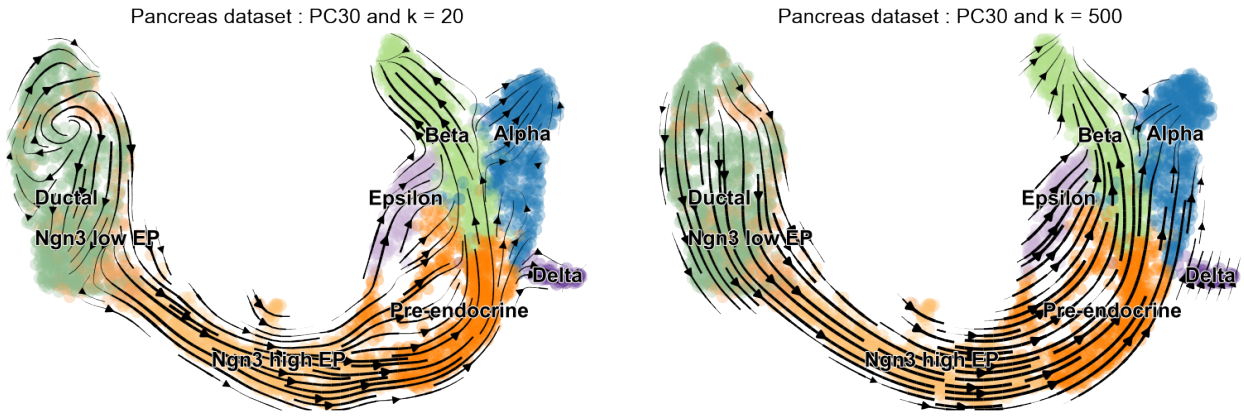
The number of neighbors chosen for the algorithm is a capital parameter. ScVelo, unlike UMAP, treats all neighbors in the same way without regard to their respective distance to the current point. A point is a neighbor or not, there is no distinction between the closest and the farthest neighbor.

If we want to catch the topology of the high dimensional space, we need to take into account, several neighbors. A too low number of K will fail in this task.

The default number is 30. Conceptually, in the extreme case $K = 1$, the cell can only stay in the current state or go to the nearest state.

A high number of K is required, but since the distance is not taken into account, a too big K will smooth the data. In addition, using a too big k will take a lot of time to run.

conceptually, in the extreme case $K =$ number of cells -1, the cell can transition to all other states. if the velocity vector in the high dimension point to a far cell, is that more probable for the current cell to do a transition to this cell or the closest state but a little de-centered?



comparaison between different number of neighbours, using the pancreas dataset from ScVelo [15]

With a too high k, the velocity field conserves the general movement, but the local complexity is lost. In particular, the circular movement is replaced by a uniform direction field. In this locality, the vector field change in the opposite direction.

6.3 NUMBER OF PC TO FIND THE K NEIGHBORS

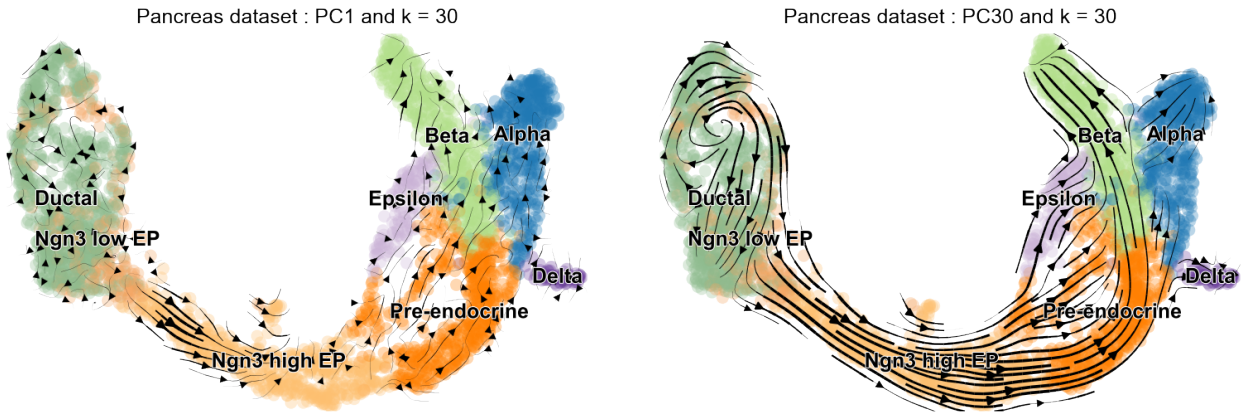
One important parameter in the algorithm is the number of PC taken into account to compute the neighborhood. Most of the time the dataset is big and noisy. A lot of information is not useful to catch the topology of the data (too low change or null value for example).

PCA is a determinist method of reduction of dimension, useful for denoising. Each PC dimension explains a part of the data dispersion, in a decreasing manner. In other words, dropping the last PCs will drop the noise on the data.

The K neighbors aren't determined on the initial data in high dimension but after applying a PCA. Only a few PC are kept, and the neighborhood is determined by this representation.

There is also here an important trade-off. With a too high PC number, a lot of not useful information is kept. The precision of the final representation stays the same but the computational cost increase.

With a too low PC number, important information is lost. Therefore, the neighborhood isn't computed correctly (the point are not the same as in high dimension space) and the final representation isn't consistent with the topology of the original data.



comparaison between different number of PCs, using the pancreas dataset from ScVelo [15]

6.4 THE SENSITIVITY PARAMETER

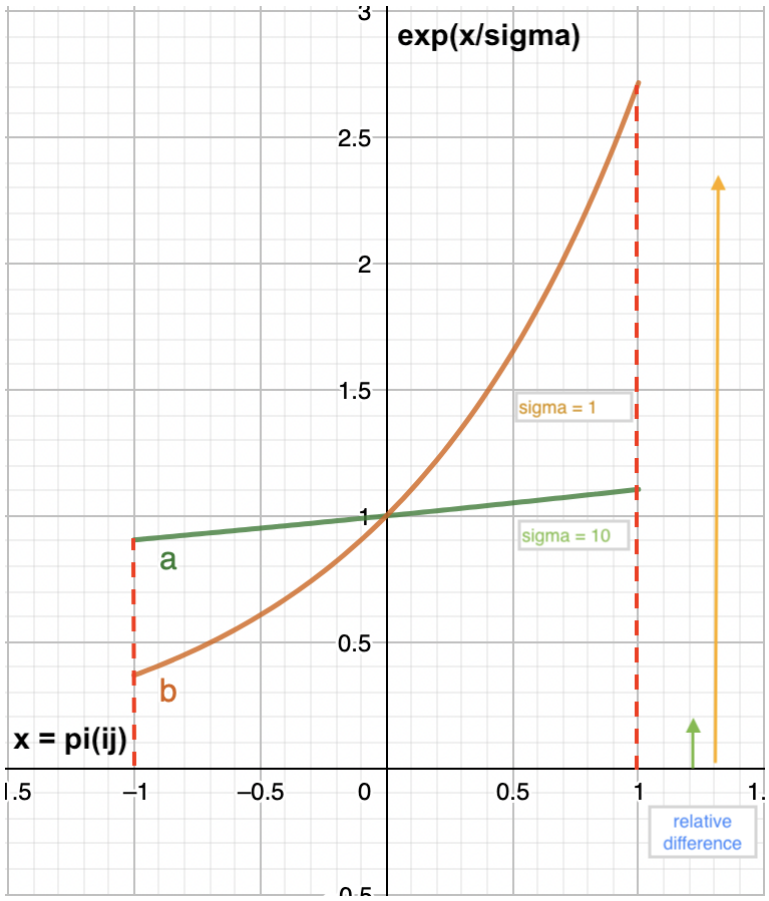
The sensitivity isn't a hyperparameter, since we can't choose it in the program. Moreover, it still influences largely the result. This parameter is optimized individually for each cell by the algorithm.

conceptually, with a high sigma probabilities are smoothed. There can be relatively low extreme probabilities.

With a low sigma, a little difference in distance can be translated into a big difference in probabilities.

The automatized optimization of this parameter prevent also from the high-density bias. In fact, if there is a region with a high density of neighbors, the computed velocity vector will be attracted in this direction even if each cell does not attract a lot individually.

Here a graph showing the influence of sigma in the forces.



PART 7

IMPROVING THE VELOCITY EMBELLING ON UMAP

7.1 THE MAIN EMBEDDING PROBLEM ON UMAP

Unlike ScVelo, UMAP is a non-deterministic algorithm. The final output is determined using machine learning, trying to preserve the structure of the high-dimensional data.

Moreover, UMAP does not generate a map from the entire high to low dimensional space. Therefore there is no function to translate another point, or that can be used to translate the RNA velocity profile, from a high to a low dimension using the normal UMAP algorithm. In other words, each data point in the high dimension has an equivalent in the low dimension but UMAP didn't build a representation of all the high dimensional space in the low dimensional space.

In contrast, PCA is a deterministic and mapping algorithm. With PCA it's possible to extract a function that maps the high dimensional space in the low dimensional space, which can be used also to move the RNA velocity profile. With UMAP, there is no map and no "correct" way to embed the velocity profile.

The ScVelo algorithm finds a solution by looking at the direction of each vector according to the neighbor's position in the high dimension and trying to keep this direction in the low dimension.

Unfortunately, there is no way to know how much the data is distorted by the UMAP projection. In addition, using a different number of neighbors can reverse the direction of the embedded vector field, as shown in the chapter about ScVelo parameters. This shows that there is a need to develop a method to determine which embedding is more reliable.

This part aims to find and test a way to map the vector field from a high to a low dimensional space that overpass this conceptual problem.

7.2 LEARNABLE UMAP

As said before, UMAP doesn't provide a function from high to low dimensions. A idea is to find a way to map all the high dimensions in low dimensions, not only the initial point cloud, using UMAP.

learnable UMAP is a method that optimizes the same loss function that UMAP does, but also provides a function from high to low dimension in the form of a neural network, the encoder Therefore, for a given dataset $X \in R^m$.

$$\mu_{n,m} : X \to R^n$$

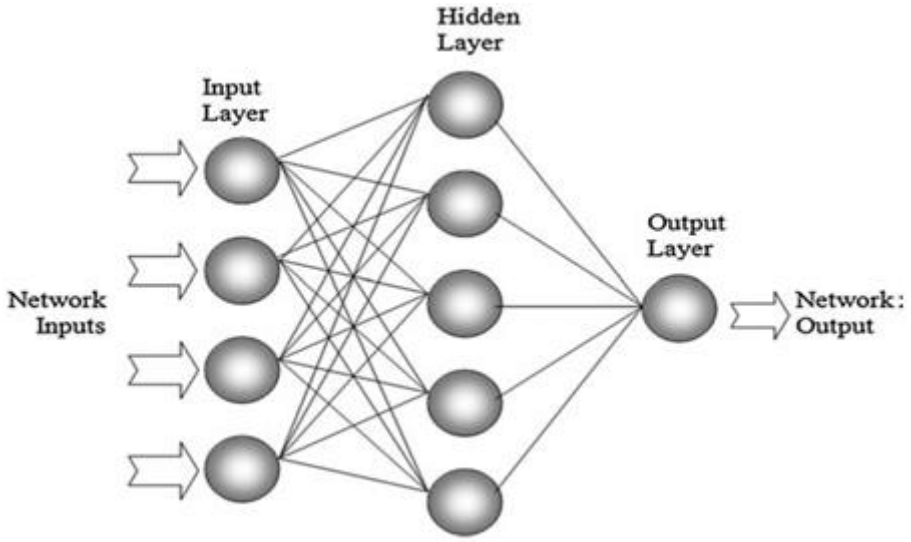
In addition learnable UMAP can provide a function from low to hign dimension in the same form, the decoder. Therefore, for a given embedding $Y \in R^n$.

$$\mu_{m,n} : Y \to R^m$$

[16]

neural network

Here learnable UMAP use by default a neural network, and an epoch of 10, with cross-entropy as a measure of fitness for his encoder and decoder. again by default, it's a 3-layer 100-neuron fully-connected neural network.



[1]

This type of neural network is made as in the figure. The number of neurons for the input layer is the high dimension, and the number of neurons for the output layer is the low dimension. There is 100 neuron in between, a neuron is connected with all neuron of the neighbors' layers.

https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a: :text=The

encoder

first of all, the low-dimension representation is computed using UMAP. A Neural network is trained with the high dimension data and the low dimension data to fit a function from high to low dimension. This function, called the encoder, links the high dimension to the low dimension. With this function, the velocity profile can be translated into the low dimension like for PCA.

decoder

To see how much the data have been distorted by the embedding, and how much information can be easily decoded, we wanted a way to go back from low to high dimension. Initially, we wanted to compute the inverse transform of the embedding using the inverse function of the learned encoder. It's not possible to invert a neural network, so we use a decoder.

learnable UMAP can also train a neural network that maps the low dimension in the high dimension and minimize reconstruction loss, called a decoder. After decoding, we calculate the reconstruction error.

As for the embedding part, we can either use learnable UMAP and his learned decoder or use a non-learnable method for the reverse embedding process. [17]

The non-learnable methods work the same way as UMAP, but from a low to a high dimension. Given a point in low dimension, it converts it into an approximation of the high dimensional representation that would have been embedded into such a location. Following the sklearn API, this is as simple to use as calling the inverse transform method of the trained model and passing it the set of test points that we want to convert into high-dimensional representations. sklearn API is the class and function reference of scikit-learn, a well-known machine learning library in python. The difference here is that the learnable decoder train a neural network, and the non-learnable method does not train a neural network but does the same process as UMAP. It constructs a KNN graph, computes weight, computes a first high-dimensional representation, and finally optimizes it using cross-entropy.

be careful that it works mostly for low-dimensional reverse embedding, the result is poor if the difference between the low and the high dimension is too big and it will be computationally costly. Typically, the implementation of UMAP returns a warning " "Inverse transform works best with low dimensional embeddings." Results may be poor, or this approach to inverse transforms may fail altogether! If you need a high dimensional latent space and inverse transform operations consider using an autoencoder." if we dimension is higher than 8.[17]

As we will see in several examples, the non-learnable method is very efficient in our synthetic dataset. To compute a good inverse transform, we can use this method.

Unfortunately, a problem appears with the reverse embedding of the vector field. As for the embedding of the vector field, we can approximate this by computing the inverse transform of two-point representing the vector. We can do this method with the non-learnable method. But if we want to apply the jacobian gradient algorithm to compute the inverse embedding of the vector field, we need the gradient from low to high dimension. For this purpose, we need a learned decoder.

As implemented, the learned decoder needs normalized data to compute correct reconstruction loss and is less efficient in our dataset. With more time we will be able to reverse properly using for example a personalized decoder, but avoid it in this thesis. Intuitively, decoding looks more complicated than encoding, because, for some high-dimensional structures like the torus example, two-point can be close in the low dimension but far in the high dimension. The decoder has to choose between the two positions in the high dimension.

7.3 ENCODING AND DECODING POINT CLOUDS

7.3.1 LEARNABLE UMAP VS UMAP

Firstly, we compare the performance of UMAP and learnable UMAP. To do so, synthetic data have been generated and tested. The synthetic data is generated in dimension 3, to be able to visualize it in the low and the high dimension.

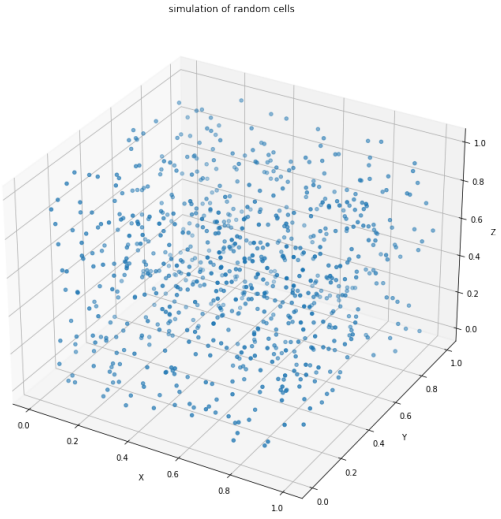
UMAP and learnable UMAP have been performed on random datasets, on more structured datasets like balls or torus, and finally on biological datasets.

Since learnable UMAP gives a function from high to low dimension, it's possible to find a reverse function from low to high. This function exists and is called a decoder, as explained earlier. If the high dimensional structure of the data is well represented in the low dimension, only a little information is lost. Therefore, the reconstruction of the data using the decoder should be good.

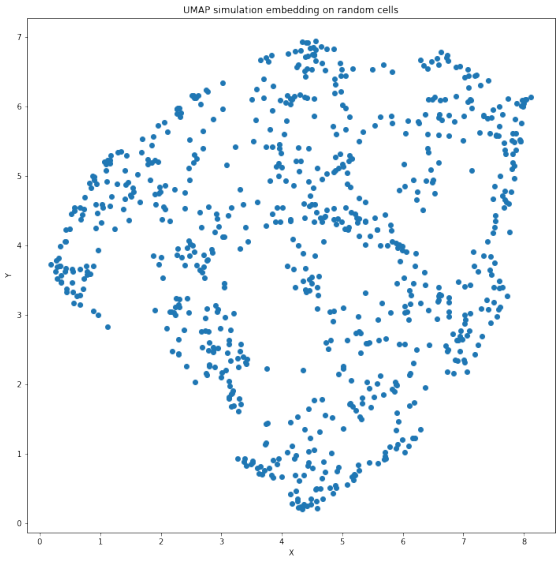
The non-learned decoder had been used on learnable UMAP mapping output as a verification of the

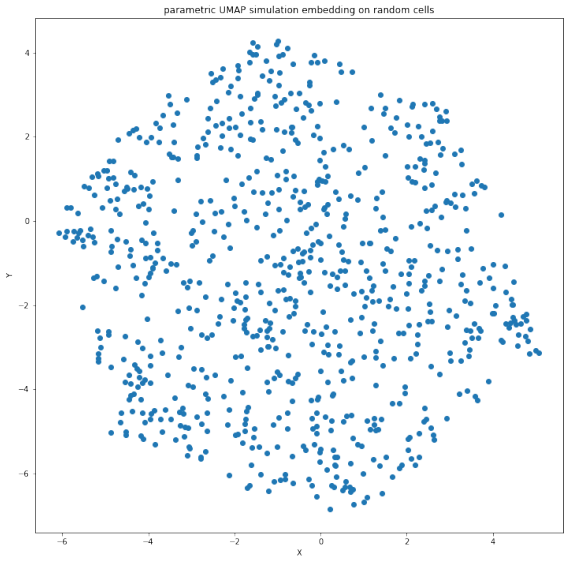
mapping quality.

Density preservation

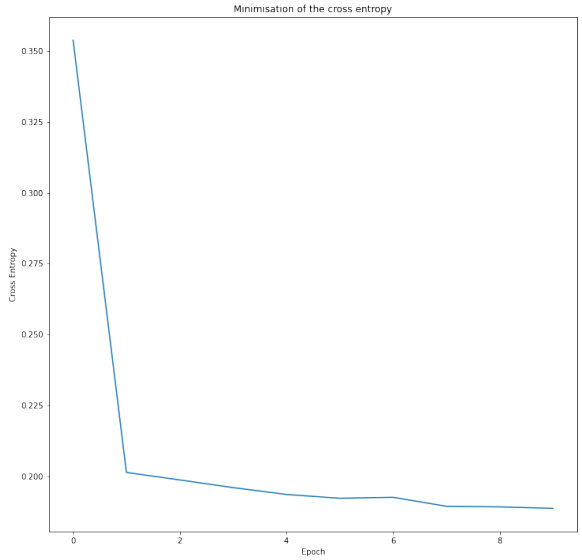


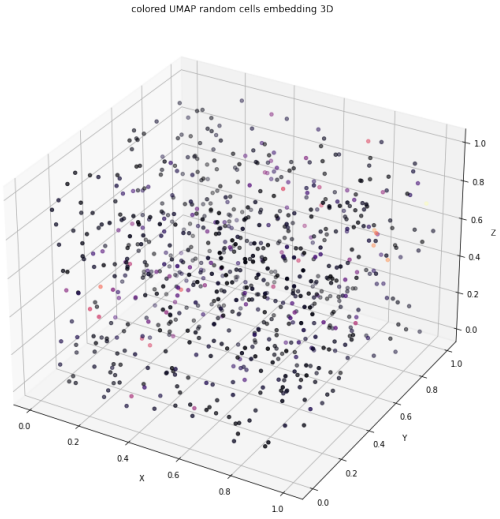
The first synthetic dataset is formed of 800 random cells in a cube of length 1. Therefore, UMAP and learnable UMAP is applied. (left normal map, right learnable UMAP

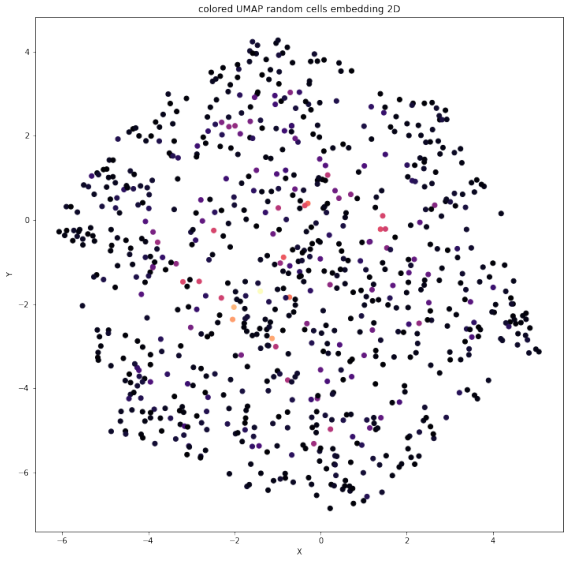




The learnable UMAP is seen to work better than UMAP on this synthetic data. indeed, learnable UMAP is less sensitive to little change in random data density and focuses on the general topology of the high dimension data. In addition, UMAP is optimized for clustering, but clustering in random data is nonsense. Visually, learnable UMAP embedding better reflects the high dimensional density.







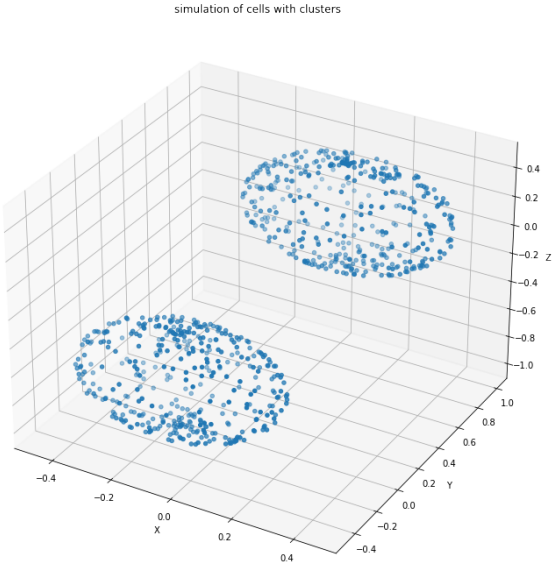

```
distance between cells
mean :  0.1257792426725281
max :  1.3222184742432734
min :  0.004107813377915406
```

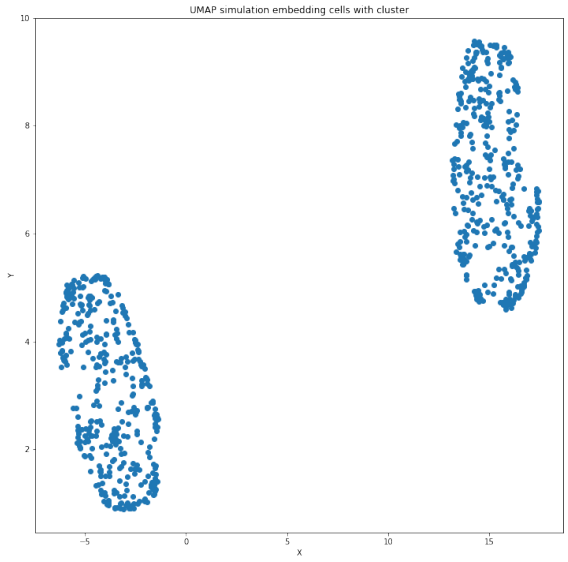
the final cross-entropy is quite low. For the last representation, the color of the point represents how far a point of the reconstructed data is from the original point, in the high dimension. We used the magma set of color, black represent low to null distance and a red point is far in comparison to other. The reconstruction is very good, as the mean distance of less than 0.0 The points in the center of the low dimension embedding seem to be mapped with more error, and are seen to be randomly distributed in the high dimension.

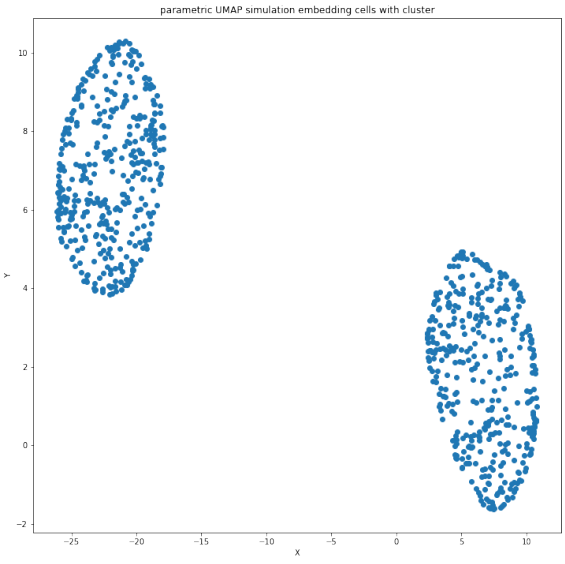
Since learnable UMAP was seen to work better because less clustered, the second synthetic dataset has been made to test how good the learnable UMAP is for clustering.

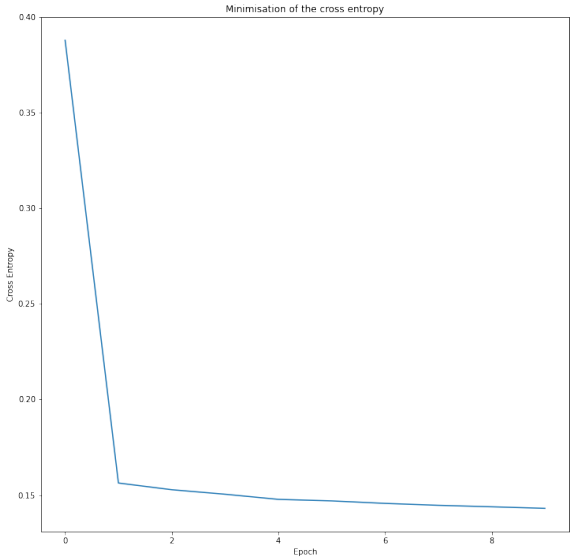
Here, UMAP and learnable UMAP make good predictions. Again learnable UMAP is seen to keep more fidelity to the high dimension topology.

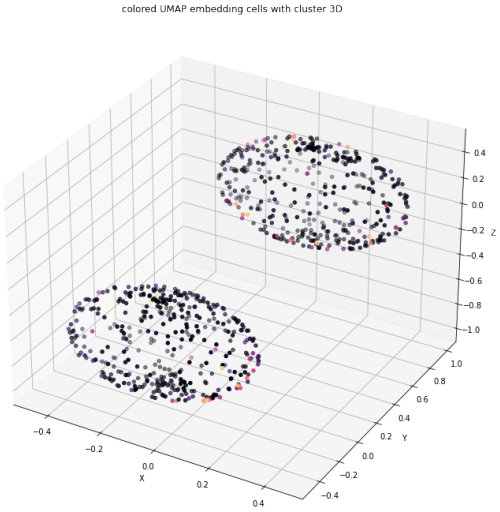
This show again that learnable UMAP preserves the density of the high dimensional data In addition, the reconstruction error is higher at the center of each cluster. In fact, the decoder has to decide if a point was at the top or the bottom of the eggs for each cluster. The reconstruction error is a great measure of the quality of the embedding.

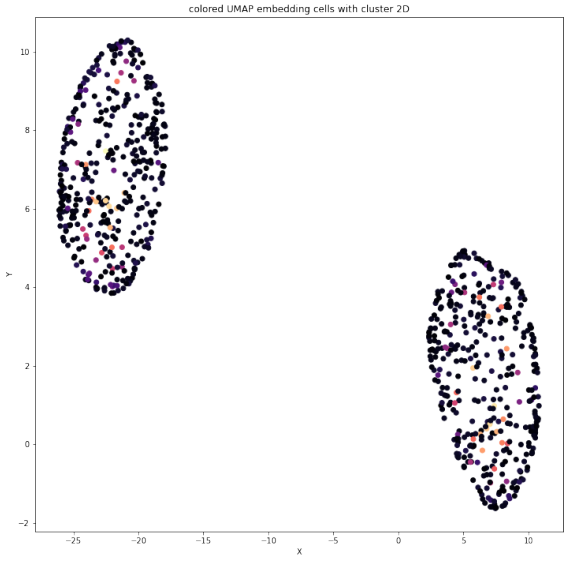













```
distance between cells
mean :  0.07008455887440554
max :  0.6562560693680142
min :  0.004046763112234922
```

Reconstruction error

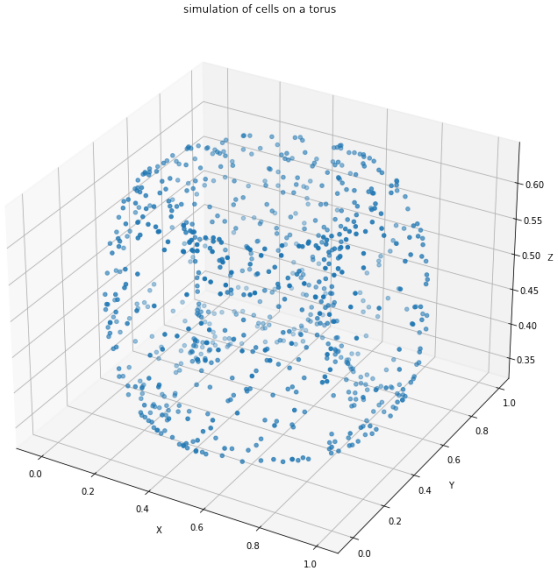
The synthetic data have been generated randomly on a torus. Again the representation is better for the learnable UMAP (less clustered). The reconstruction is quite perfect, probably because the function found by the neural network fits the tore parametrization, which is more predictable than the random dataset before.

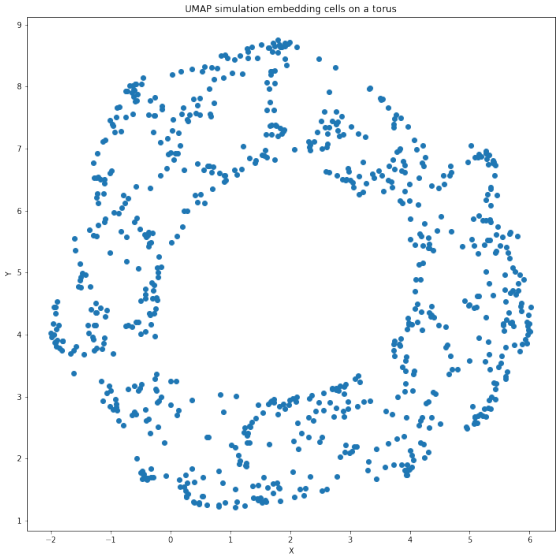
Have a look also at the reconstruction error on this dataset. On the 2D colored representation, we see that the highest reconstruction error appears at the interior of the ring. In fact, here the learnable UMAP embedding flat the z-axis. Therefore, points that were far away in the low dimension are embedded closely in the low dimension. When we want to reverse embed the dataset, the decoder has to decide if a point was initially in the high or in the low part of the torus. When the decoder fails, the point is translated far

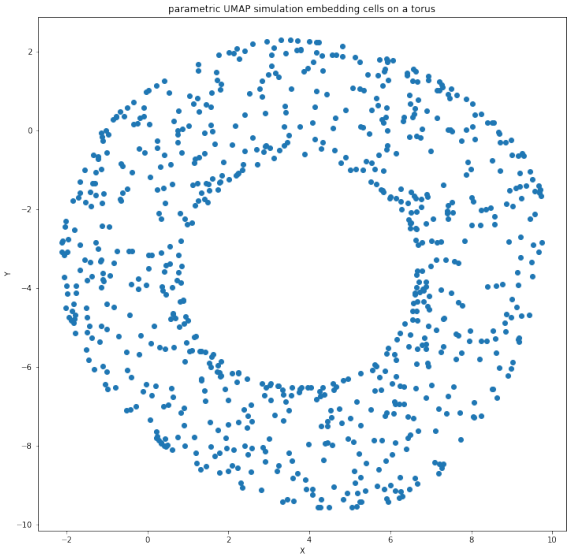
away from its initial position in the original dataset.

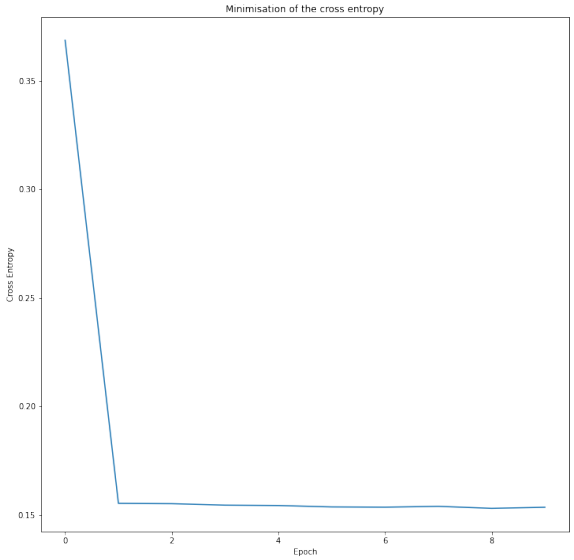
In this example, it's clear that we obtain a high reconstruction error in the part of the dataset that is not clearly distinct in the embedding. When we lose information, like the one on the z-axis, the reconstruction error increase.

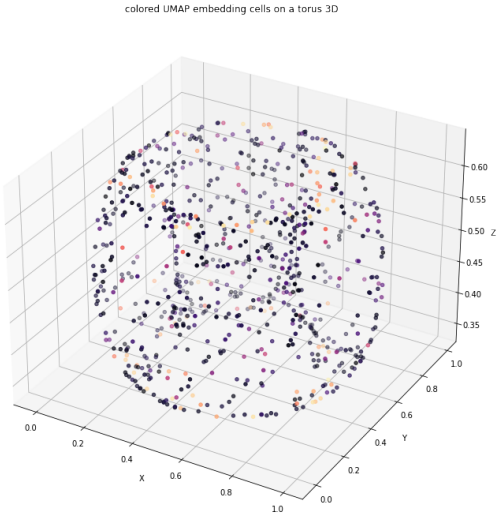
It is why we can use the reconstruction error as a measure of the quality of the embedding.

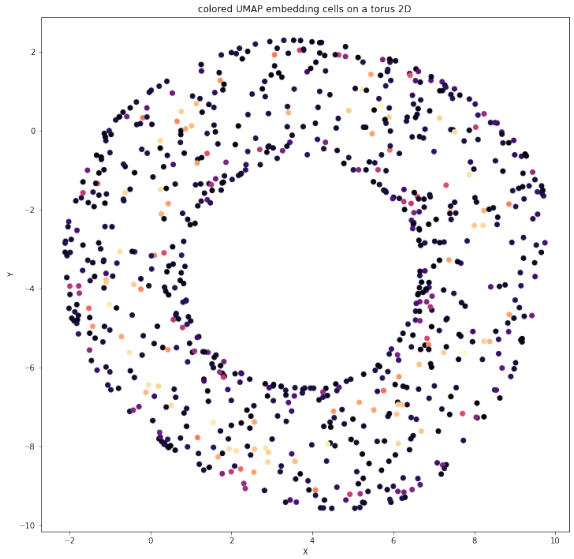










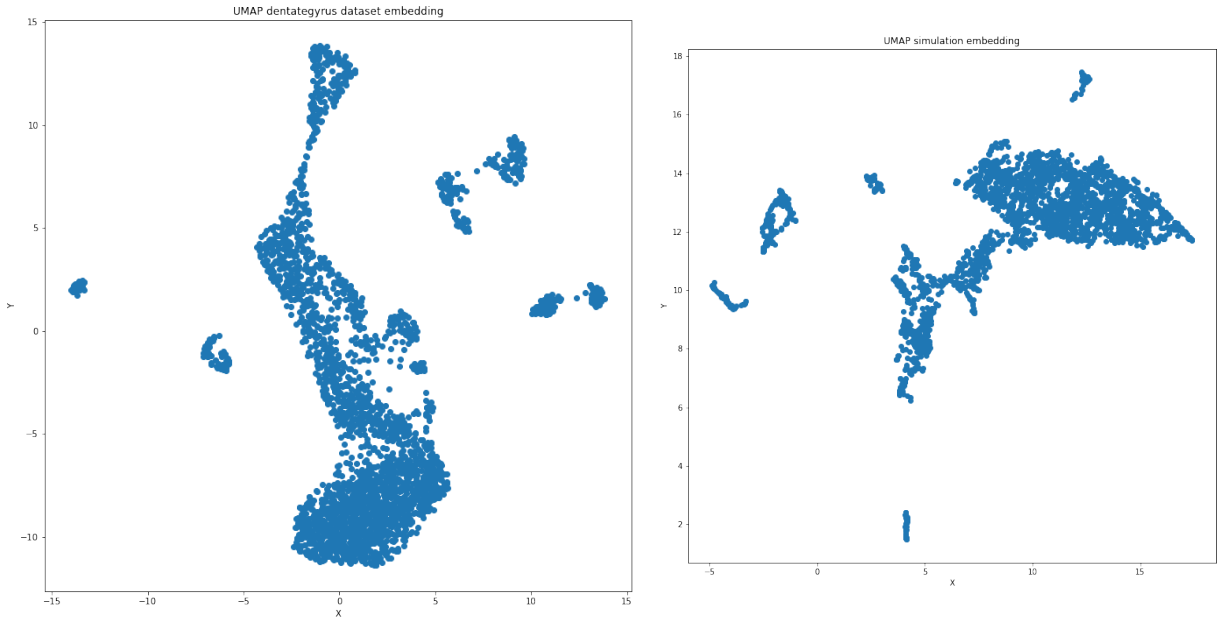



```
distance between cells
mean :  0.06668979448753189
max :  0.3122702892912692
min :  0.004711539665416945
```

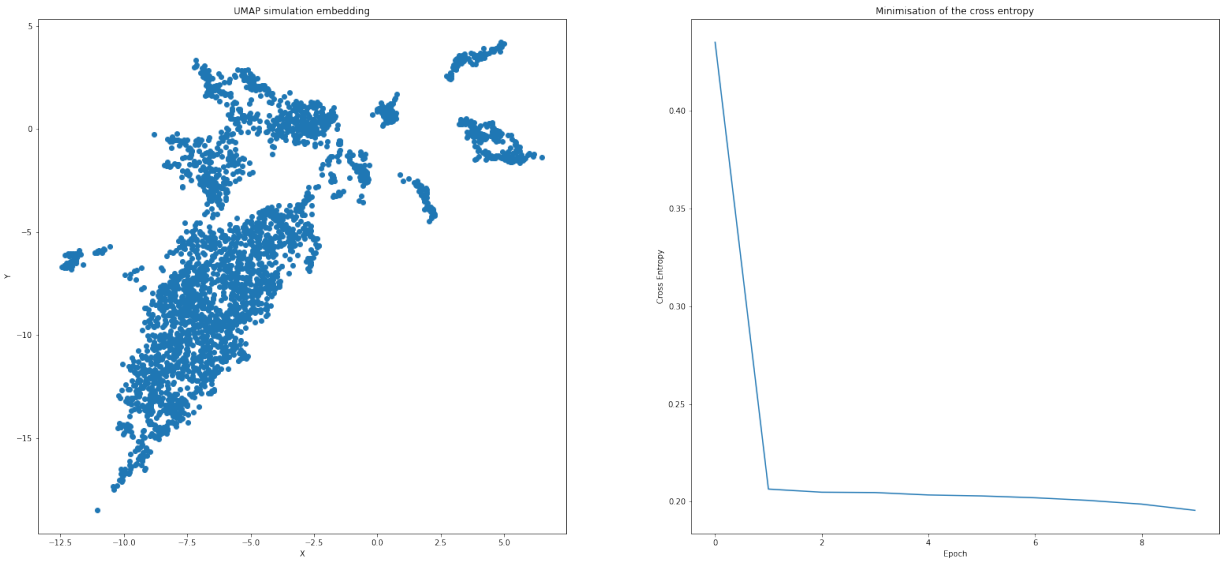
biological dataset

We wanted to test also how learnable UMAP vs UMAP performance on a bigger dataset, especially a biological one.

For this test, we use the Dentategyrus dataset from the ScVelo library. The UMAP projection has a cluster and is structured. Moreover, since this dataset is from ScVelo it's easy to also apply the color method for the embedding of the vector field comparison.



This dataset already contains a UMAP projection (left). We also make our own UMAP projection (right).



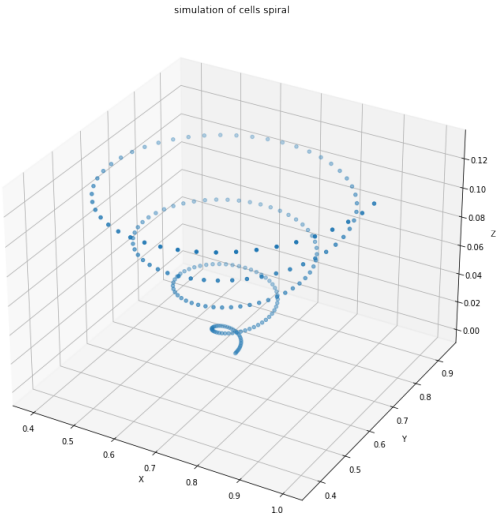
The result of learnable UMAP is good but takes a lot of time to be processed. (a least a couple of minutes for the ScVelo dataset to a couple of hours for the big biological dataset). In the same way, the non-learnable decoder is a very powerful tool but takes a long time to be compiled.

7.3.2 PCA VS PARAMETRIC UMAP

One aim of an embedding method is to make the visualization easy and meaningful. This means that important structures may be maintained in priority such as a cluster, group, or outlier.

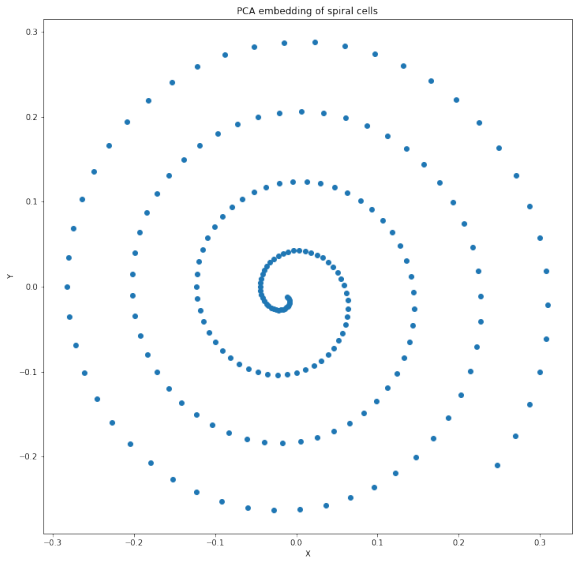
But another important aim is to "keep" as much information as possible in fewer dimensions. Therefore, a good embedding loses less information. We will be able to reconstruct a representation of the high dimensional structure with the low dimensional embedding.

A spiral is a complex structure and can be visually perfectly embedded. We use learnable UMAP to embed the spiral and reverse the embedding to see how precisely it's possible to reconstruct the original dataset.

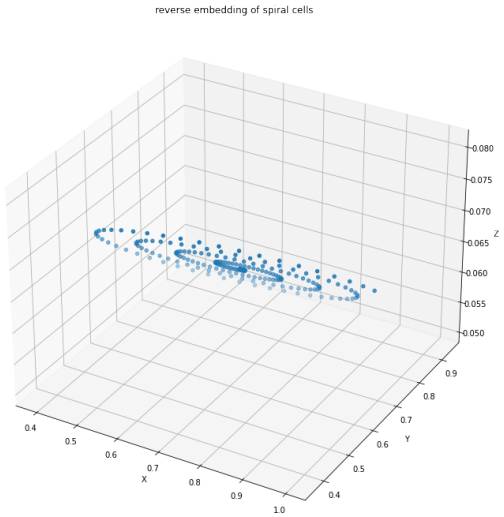


reconstruction on PCA

To see the problem that can face the decoding method, we first try to embed the data using PCA projection. The result was actually very good. The PCA method flatted the spiral.



When PCA is projected back in the high dimension, it forms a plan in the space. This is a poor reverse embedding.



This example with the PCA gives a first idea of why reconstruction can be a good way to quantify how good is the embedding, and of the problem that can appear. A sparrow is an easy example for PCA, it should be also easy for UMAP. Now we want to do the same process with UMAP projection.

reconstruction on UMAP

Up is an example of embedding with UMAP. down are embeddings with learnable UMAP. Here the

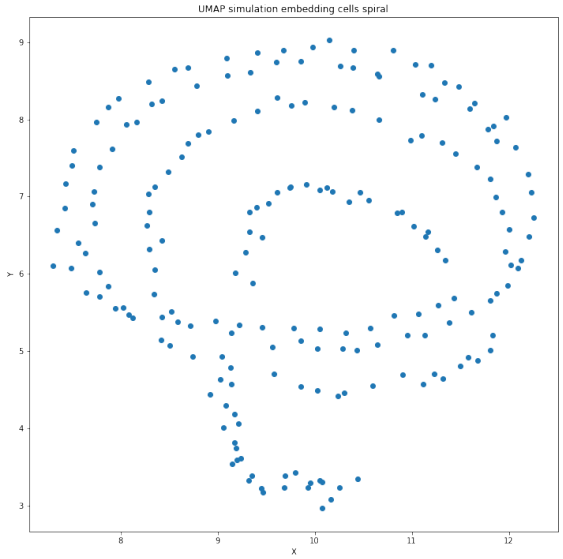
embedding with learnable UMAP is really good.

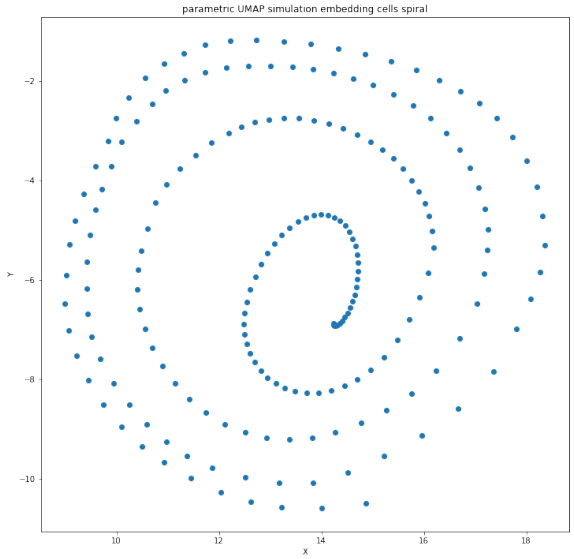
By running different spirals (more flat or elongate, small or big etc...) we observe two different types of embedding from learnable UMAP, representing probably two different minima of the cross-entropy.

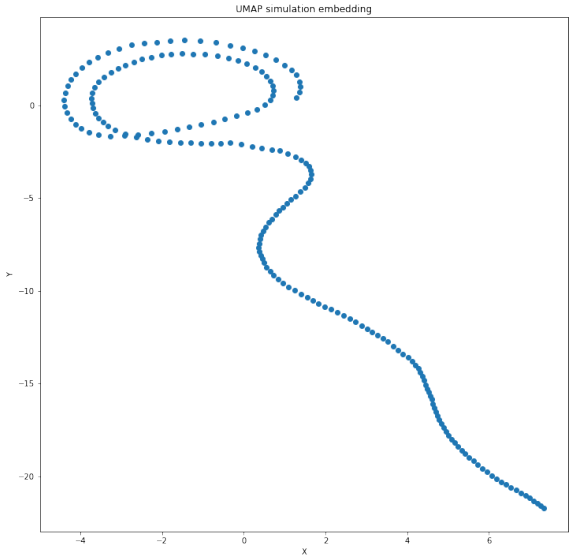
Sometimes, the method returns a spiral in 2 dimensions like the one below. The circular structure is maintained and the z-axis is flattened. We have a feeling of the z-axis by looking at the distance between two consequent points and the distance to the center of the structure.

Sometimes, the algorithm returns a "wave" embedding. The circular structure is lost and the distance on the z-axis is maintained. We have a feeling of the circular structure with the wave and the line, and by looking at the distance from the point to the line.

We can favorise one of the two structures by deforming the spiral on the Z-axis or the x-y plane. The method decides if it is better to lose the circular height component of the structure.



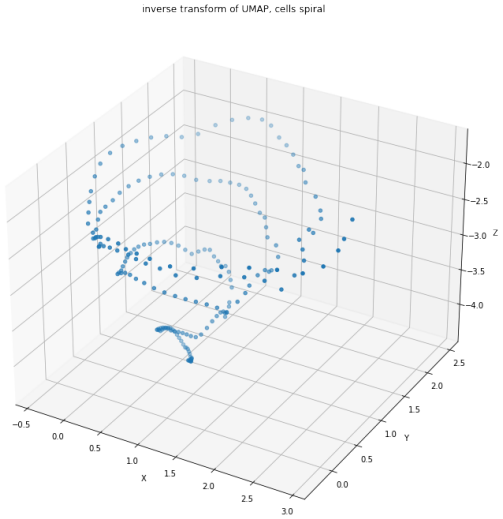




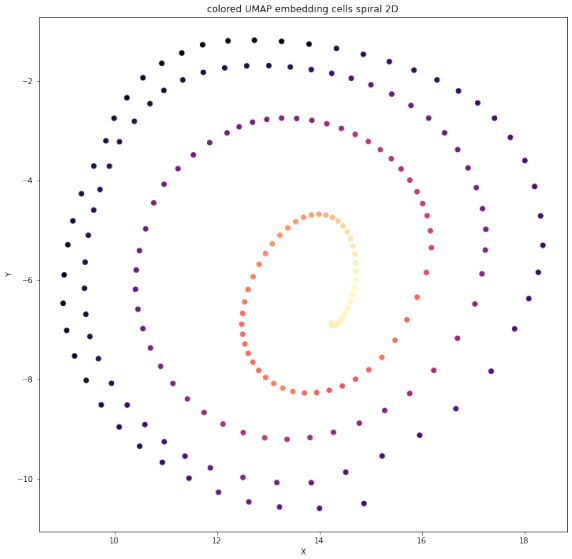
then we try to reconstruct the original dataset with the UMAP embedding and the learned function. Here is what we obtained with the circular embedding before and the original dataset. The reconstruction is quite good. We have a feeling of the circular structure (but not very circular at this point). The z component is also reconstructed but with less accuracy.

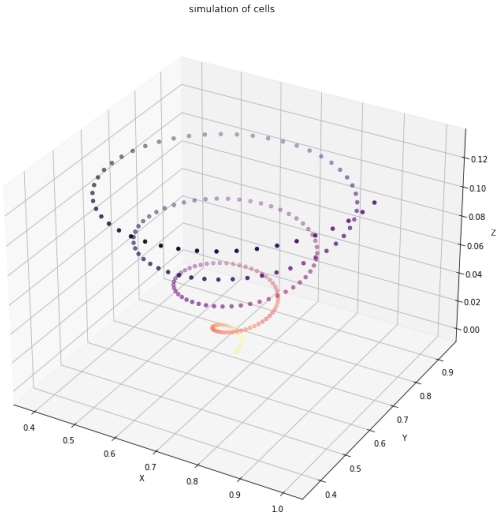
The color in the last graph represents the distance between the original point and the reconstructed one. As the distance increase, the color becomes lighter. A black point is perfectly reconstructed.

The most difficult part to reconstruct is seen to be the bottom part of the sparrow. However, the reconstruction error is biased because the scale of the original sparrow and the reconstructed sparrow is not the same. This is also why we have such a distance between reconstructed and original cells.




```
distance between cells
mean :  3.0943555915637764
max :  4.367446728548069
min :  2.1487076240338254
```





7.4 ENCODING AND DECODING VELOCITY VECTORS

7.4.1 JACOBIAN ENCODING AND DECODING METHODS

principle

Given methods that produce a map from high dimension m to low dimension n, for a given dataset $X \in \mathbb{R}^m$.

$$\mu : X \to \mathbb{R}^n$$

And the vector field in high dimension is like below

$$V_m : X \to \mathbb{R}^m \cong T_x\mathbb{R}^m$$

Now we want to map the vector field V in low dimension as μ

If there is a smooth function $f : \mathbb{R}^m \to \mathbb{R}^n$, the jacobian of this function

$$\mathbf{J}_{f(x)} : T_x\mathbb{R}^m \to T_{f(x)}\mathbb{R}^n$$

makes the link between the tangent space in high dimension and the tangent space in high dimension. therefore, the mapping of the vector field is make as below.

$$\begin{aligned} V_n :& \mu(x) \to R^n \cong T_{f(x)}\mathbb{R}^m \\ & \mu(x) \to \mathbf{J}_{f(x)} V_m(x) \end{aligned}$$

With the function, it's become possible to map from high dimension to low dimension any point even if it wasn't in the original high dimension dataset. It's possible to extract this function and calculate its jacobian.

For this study, we develop the case of a dataset originally in $\mathbb{R}^3$, that we want to map in $\mathbb{R}^2$. In this case, the function is defined as

$$\begin{aligned} f :&\mathbb{R}^3 \to \mathbb{R}^2 \\ &(x, y, z) \to (f_a(x, y, z), f_b(x, y, z)) = (a, b) \end{aligned}$$

The jacobian is defined as

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_a}{\partial x} & \frac{\partial f_a}{\partial y} & \frac{\partial f_a}{\partial z} \\ \frac{\partial f_b}{\partial x} & \frac{\partial f_b}{\partial y} & \frac{\partial f_b}{\partial z} \end{bmatrix}$$

and the vector field is

$$V_2 = \mathbf{J}_f V_3 = \begin{bmatrix} \frac{\partial f_a}{\partial x} & \frac{\partial f_a}{\partial y} & \frac{\partial f_a}{\partial z} \\ \frac{\partial f_b}{\partial x} & \frac{\partial f_b}{\partial y} & \frac{\partial f_b}{\partial z} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} V_a \\ V_b \end{bmatrix}$$

This method (from m dimension to 2 dimensions) has been implemented in python.

more generally, from the m dimension to n dimension. The function is define as

$$\begin{aligned} f :&\mathbb{R}^m \to \mathbb{R}^n \\ &(x_1, x_2, ..., x_m) \to (f_1(x_1, x_2, ..., x_m), ..., f_n(x_1, x_2, ..., x_m)) = (y_1, y_2, ..., y_n) \end{aligned}$$

The jacobian is a matrix of dimension nxm

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & ... & \frac{\partial f_1}{\partial x_m} \\ ... & ... & ... \\ \frac{\partial f_n}{\partial x_1} & ... & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

and finally, the embedded vector field in dimension m is computed as below

$$V_n = \mathbf{J}_f V_m = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & ... & \frac{\partial f_1}{\partial x_m} \\ ... & ... & ... \\ \frac{\partial f_n}{\partial x_1} & ... & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \begin{bmatrix} Vx_1 \\ ... \\ Vx_m \end{bmatrix} = \begin{bmatrix} Vy_1 \\ ... \\ Vy_n \end{bmatrix}$$

implementation

the function "velocity embedding", from the particular case where the embedding is computed from a high dimension to 2 dimensions, has been implemented and tested in python.

This method extracts the encoder of the trained learnable UMAP model. This encoder is a function computed as a neural network. The gradient in each cell point is computed from the encoder. Since we have an RNA velocity for each cell in the high dimension and the gradient of the encoder for each, the embedded velocity in the low dimension is easy to compute.

approximation

The jacobian method can take time to process because this requires computing the gradient in each point. For too large datasets or too big dimensions, this method can be useless. Perhaps the jacobian method does not takes much time in comparison with the computing of the learnable UMAP model.

An approximation of the vector field embedding in low dimension can be made by using only two-point as a linear approximation of the vector instead of the gradient.

Since the length of each vector in high dimension isn't significant, it can be useful to normalize the vector field before using this approximation.

the function "vector calculus" represents a vector by a starting point (the cell) and an endpoint (in the direction of the vector) in high dimension. The encoder is used to compute the embedding of these two-point and there represent the vector in low dimension.

This method is mathematically less robust, but takes way less time and gives, either on synthetic or in the biological dataset, similar results to the jacobian method.

for PCA

PCA is a determinist embedding method and consists of a smooth function from high to low dimension. If an embedding comes from a smooth function, after decoding both to the original space, the jacobian embedding is the "correct" embedding of the vector field.

PCA process a smooth linear transformation

$$y = T * x$$

withy the PCA components, x the high dimension data, and T the weight matrix or loadings. Therefore, the Jacobian in a point is equal to the matrix T at this point.

It's possible to extract this jacobian and use it to process the jacobian embedding of the vector field.

for UMAP

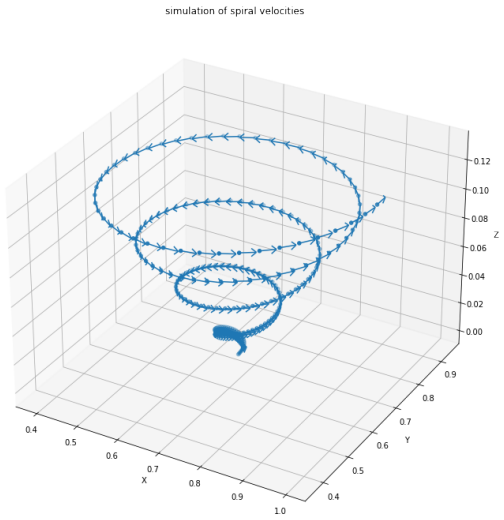
Reminder, learnable UMAP produces a map from high dimension m to low dimension n, the encoder. It is possible to apply the jacobian method with the encoder.

In the same way, it is possible to apply the jacobian method with the decoder.

Since the decoding of the vector field is based on the gradient of the decoder, if the decoder makes a bad reconstruction the reconstructed vector field will also be bad. We will see that the vector field seen well reconstructed according to the reconstructed point cloud.

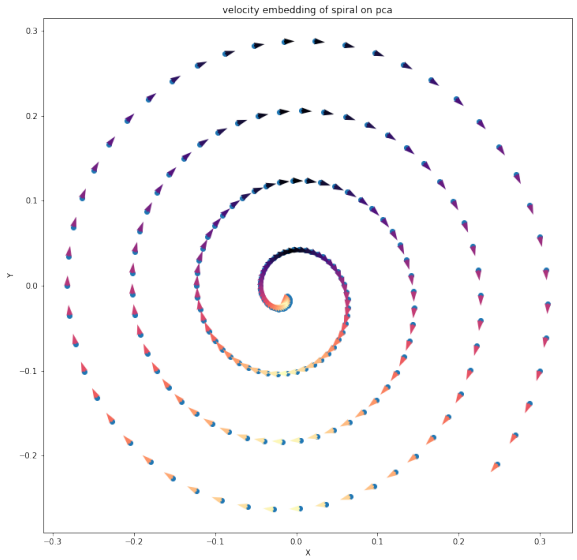
7.4.2 JACOBIAN METHOD ON PCA VS LEARNABLE UMAP

We will use the same spiral example as above, with a vector field on the spiral. We embedded it and try to inverse the embedding.

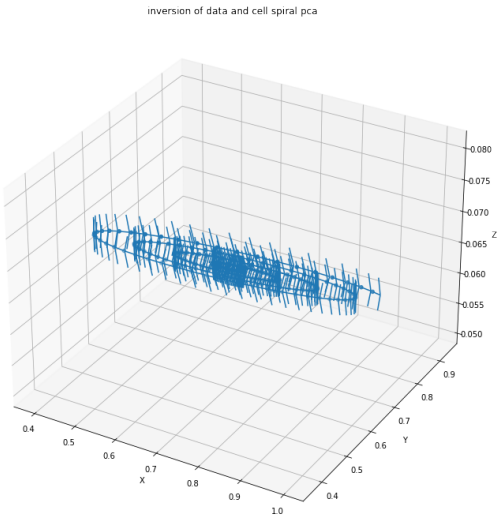


reconstruction on PCA

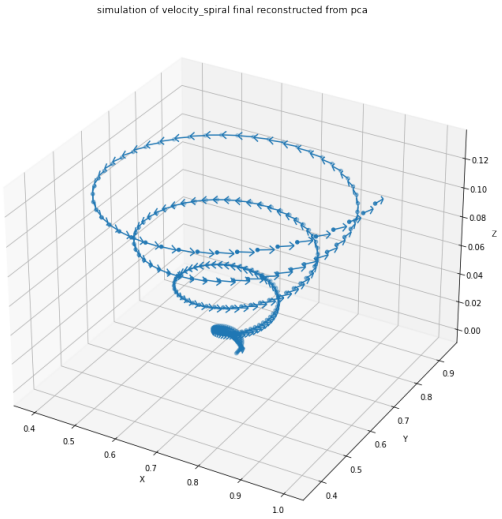
We embed the vector field on the result of the PCA using the jacobian method. The result was actually very good, the vector field follows closely the resulting trajectory.



Then we invert the vector field and plot it into the reverse embedding of the point cloud and obtain the following result.



However, when we plot the same vector field on the original dataset, the reconstruction is near to be perfect. In fact, the reconstructed vector field is at 1 degree in mean the same as the original one, but this work well so far only because the original vector field has no component in the z-direction. If there was a component in this direction, it will be flattened and the representation will be less good.



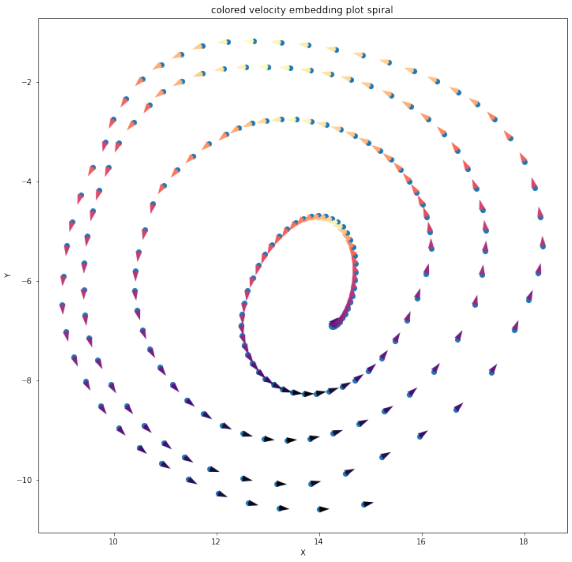
Now we want to do the same process with UMAP projection.

reconstruction on UMAP

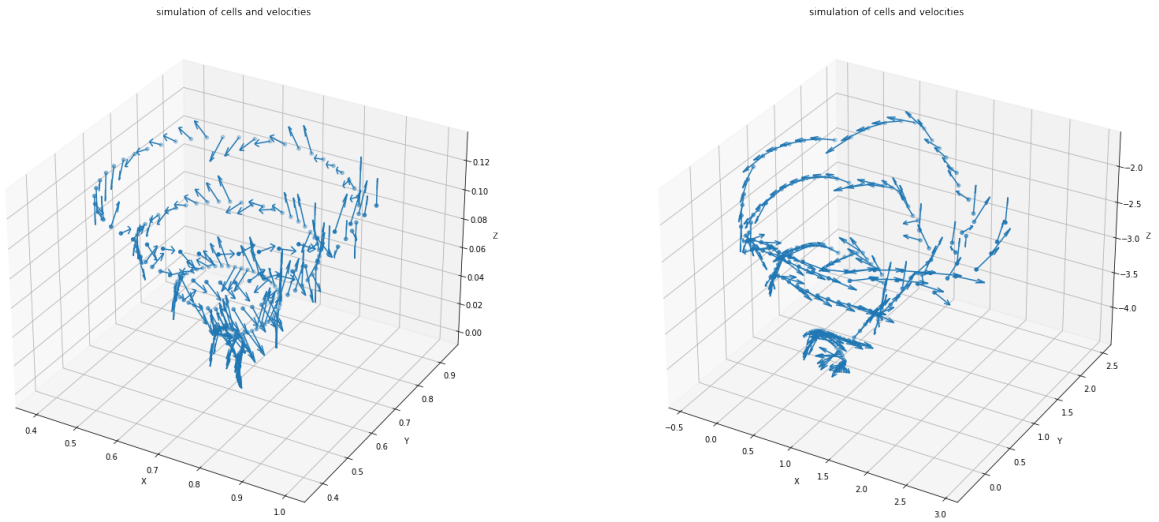
We embed the vector field with the jacobian method and the approximation point-to-point method. We

obtain approximately the same vector field embedding, only 2 degrees of difference in mean, between the approximation and the jacobian method. We will focus on the jacobian embedding.

In the following figure, the color of the vector represents the angle of the vector in two dimensions. The embedding of the vector field with the jacobian method is really good. The vector field follows the trajectory of the flat spiral.



Finally, we want to embed back the vector field from the low dimension to the high dimension. To do so we tried again with the jacobian method, using the gradient of the decoder, and the approximation method. Unfortunately, the approximation method didn't work well. We plot the result of the jacobian method on the original dataset (left) and on the reconstructed dataset (right).



When we plot the vector field on the original dataset, the inversion looks bad even if the embedding of the point cloud and of the velocity looks good. However, when we plot the same vector field on the reconstructed data, we find that it follows the trajectory of the spiral.

This means that the reverse embedding of the vector field isn't bad, but we have to take into account the poor reverse embedding of the point cloud.

In fact, since the learned decoder isn't really good, the gradient method using the jacobian of the decoder also is.

7.4.3 JACOBIAN METHOD VS SCVELO METHOD ON PCA

Principle

How can we quantify whether the ScVelo embedding method is good? To test the validity of the ScVelo method, we will use PCA. The velocity embedding from Scvelo is compared to the velocity embedding using the jacobian.

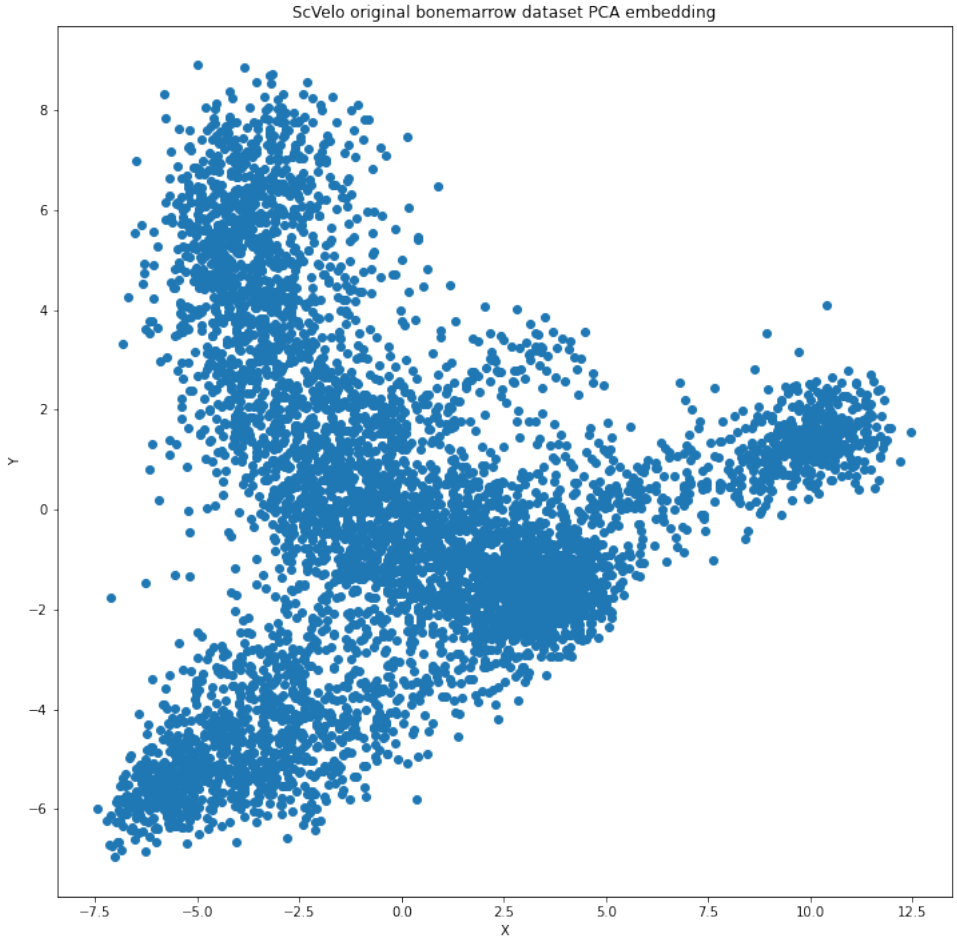
Since we assume that the length of the vector isn't important but the direction is, the comparison between the two embeddings must be based on angles.

implementation

We will use datasets from the ScVelo package for convenience. The PCA projection is given, and deterministic, unlike the UMAP projection.

The first example is the bonemarrow dataset, composed of 5780 cells in dimension n = 14319.

First, we embed the dataset in dimension 50 using PCA. We will use the result as high-dimensional data. The final embedding is made in dimension two.



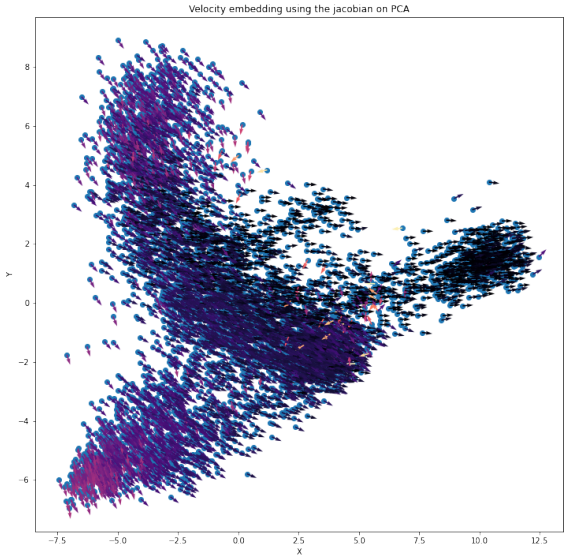
We compute the embedding using the jacobian method on the PCA projection. To do so, we extract the matrix of loadings, in dimension [2,n], and compute the embedding of the vector field.

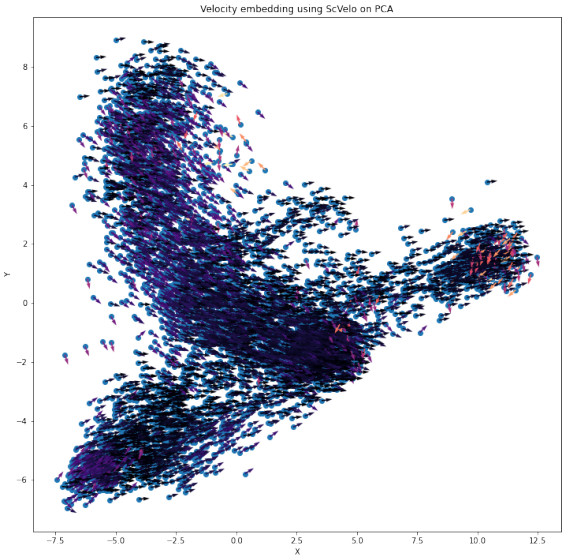
To compare the two vector fields, we compute the angles between each pair of embedded vectors and some statistics like the mean, the maximum, and the minimum of similarities.

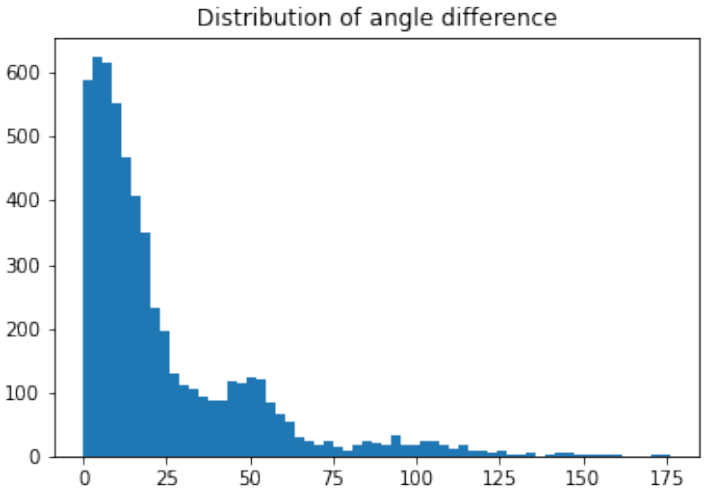
Finally, to test the embedding we reverse embed the vector field and the point cloud from dimension 2 back in dimension 50 and compare the result with the original vector field.

Results

At left is the embedding of the vector field using the jacobian method, and at right the embedding of the vector scvelo. The color of the vector relates to the angle. If two vectors have the same color, they have the same direction. This permit visualizing the global direction of the vector field and the outlier.








```
distance between vector (in degree)
mean :  24.60901468194343
median :  14.757233091062037
max :  175.94434181714922
min :  0.08187577464277193
```

The median difference in angle between scvelo and PCA embedding is 15 degrees. As we can see the distribution of similarity evaluate like an inverse log function.

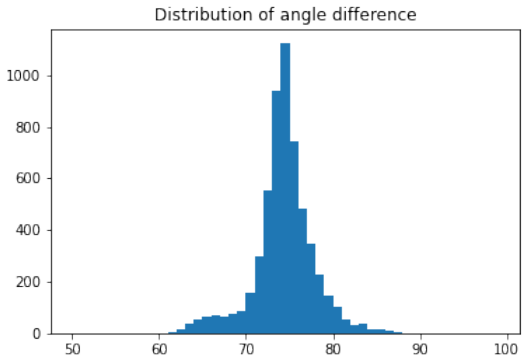
The majority of vectors have only a little change in angle between the two embeddings. Some have a total change (the maximum is close to being in the opposite direction !) but that is only a local outlier.

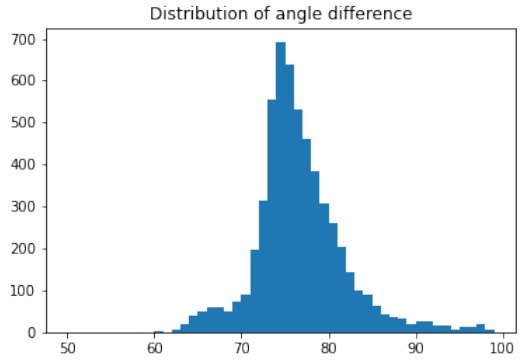
The global direction of the vector field is mostly the same for the two embeddings. It's seen that there are fewer points that have a different direction than the local field in the jacobian embedding. Since the similarity is also a measure of the validity of the vector field, that means that the embedding is quite good. and meaningful.

In some way the jacobian embedding since to be not meaningful in the extreme part of the embedding. For example at the left bottom, the vector field from the jacobian embedding does not follow the cells while the vector field from the scvelo embedding does. The Scvelo embedding method is optimized to keep the right direction regarding the neighborhood of the local cell. This prevents this king of the pattern when the vector field goes in nowhere.

The scvelo embedding is concerned by some difficulty regarding the distortion of the data from high to low dimensions. It is convenient to keep some consistency for the embedded vector field as ScVelo does, but sometimes when the high dimensional vector field is complex or contains some outliners and non-related cells, the vectors can be distorted.

It's consistent with what we observed by comparing the high and low numbers of neighbors for the kind graph used by scvelo. Sometimes, considering too many or few neighbors in a complex structure can invert the embedded vector field. We reverse embed the two embedding back in dimension 50 using the jacobian method and the decoder for PCA. A reminder of what we have observed in the last example: PCA will project back the vector field in the form of a plane in the high dimension, here 50. By definition, PCA embedding will return the best plane. We expect the reverse embedded vector field from the jacobian method to give a better result than the reverse embedded scvelo.





Here, the Jacobian method has in mean a difference of 74 degrees from the original vector field. The scvelo method has in mean a difference of 77 degrees. Here, the Jacobian method gives a better result. Moreover, if we compare only the first two dimensions, the jacobian vector field is reverse embedded perfectly, because PCA is a linear method. In these two dimensions, the difference in performance between the scvelo and the jacobian method is clear.

Now, we have to test our method in UMAP embedding, using learnable UMAP.

7.4.4 JACOBIAN METHOD VS SCVELO METHOD ON UMAP

We now want to project the data in low dimension using UMAP and embed the vector field.

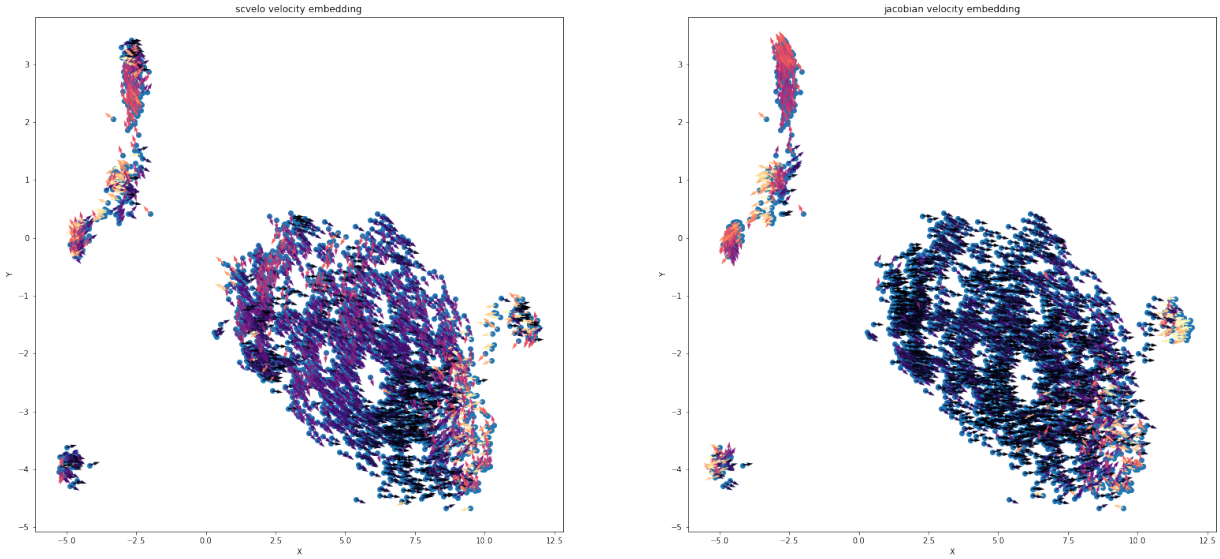
In the previous part, it's seen that learning a function from UMAP embedding is possible and give good result in comparison with non-learnable UMAP. This function makes the method of the jacobian possible.

In addition, the PCA ground truth experience gives a feeling that the jacobian and the ScVelo method gives a similar result.

biological dataset

We use the Dentategyrus dataset from sc velo for this example.

Firstly we used learnable UMAP to compute a projection in 2D from this dataset. Secondly, we embed the vector field using scvelo and the jacobian method on this projection.



This result is consistent with what we observed in the PCA projection. The two methods have the same general direction. Scvelo embedding show more small variation of the vector field where the jacobian method keeps the same direction.

7.5 CONCLUSION

In this part, we introduce learnable UMAP for the embedding of the point cloud.

We have shown that this method preserves density and structure, and provide an example where learnable UMAP and standard UMAP are coherent. We also show that the notion of reconstruction error makes sense and can be used as a measure of the goodness of the embedding.

Furthermore, we provide a method for the embedding of RNA velocity and a reverse method to test the distortion of the data.

The jacobian method works well on PCA projection, and also in UMAP projection. In addition, the jacobian and scvelo methods give coherent results. The vector field is well reverse embedded on the reverse embedded point cloud, finally, decoding can be used to measure how good the embedding is.

We also face some problems, especially on the decoding part :

- the decoder keeps the general structure of the high dimensional data but is proportionally distorted in low dimensions. For example, we encode a spiral from high to low dimension, and back from low to high dimension using the learned decoder (neural network). The reconstruction looks well as a spiral, but the center is not the same as the original one (shift of the dataset), the spiral is bigger or smaller (distortion), or the structure is turned (also a distortion). We need to work on the decoder to understand why this problem appears because it should map back the point cloud close to the original. Here it's seen that the decoder favorises the general structure of the high dimension but not the scale. This makes it difficult to compare the reconstruction and the vector field, we need a comparison method based on the structure and not on distance and angle (maybe covariance).
- the decoder is trained only with the original structure. So, if the vector field goes in a direction that isn't in the original structure (for example going far away from the structure of the spiral when all

the point tends to go to the center of the spiral), the decoder will be unable to understand this point and map it back on the surface (because it's looking more logical).

- finally, if the decoding of the point cloud is bad, the decoding of the vector field is also. With the sparrow, for example, the original vector field follows the path of the point cloud, making a spiral. The reconstructed vector field keeps the same property: it follows the point cloud.

  The decoding of the point cloud isn't really good, so the reconstructed vector field doesn't follow a spiral because the reconstructed point cloud doesn't.

PART 8

GENERAL CONCLUSION

In this paper, we explored RNA velocity embedding in many ways.

In the first step, we provide an overview of what RNA velocity is, the difficulty of embedding this type of data, and the current state of methods and knowledge.

UMAP is the current method for the embedding of the point cloud. We explain the functioning of this algorithm, based on KNN-graph and gradient descent to find a low dimensional representation that fits as much as possible the structure of the high dimensional dataset. We explored some of the most important parameters and showed how much these arbitrary choices can influence the result. Scvelo is the current method for the treatment of RNA velocity data. We show how it computes a high-dimensional vector field from a count of spiced and unspiced RNA, and explain the functioning of the embedding method. We explored also ScVelo parameters, and provide examples where a little change can return the opposite result.

In the second step, This study investigates two main points.

Firstly, the two embeddings depend on many arbitrary parameters, where it is difficult to get the best combination for visualization. Secondly, we have no way to quantify the distortion of the data by the embedding.

To solve these problems, we introduce learnable UMAP as a method to embed point cloud, learning the parameter and returning a function from the high to the low dimension as an encoder. We have shown that learnable UMAP preserves the density and the structure of the high dimension as well or better than standard UMAP. We also reverse embed the data using a learnable UMAP decoder to quantify the distortion and provide an example that proves the utility of reconstruction error as a measure of quality for an embedding. For the velocity field, we implemented and tested a method named the jacobian gradient method. We have shown that this algorithm returns coherent results in PCA and UMAP projection, as well as on synthetic and biological datasets. We compare it to the scvelo method and show that the jacobian method works better than the scvelo one on PCA projection. We face some problems with the decoding part. The decoder distorts the data and isn't accurate is the difference in the number of dimensions is too high. This also influences the decoding of the vector field. In fact, the decoding of velocities looks good regarding the decoding point cloud but follows the same distortion.

If the future, we can imagine a new personalized decoding method to overcome these difficulties. Finally, we envision that our method could be used to detect highly distorted regions of the vector field by current embedding methods.

BIBLIOGRAPHY

[1] *Fig. 3. Typical 3-layer fully connected network.* ResearchGate. URL: https://www.researchgate.net/figure/Typical-3-layer-fully-connected-network_fig2_315791883 (visited on 8th June 2022).

[2] *How UMAP Works — umap 0.5 documentation*. URL: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html (visited on 3rd Mar. 2022).

[3] Gioele La Manno et al. 'RNA velocity of single cells'. In: *Nature* 560.7719 (Aug. 2018). Number: 7719 Publisher: Nature Publishing Group, pp. 494–498. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0414-6. URL: https://www.nature.com/articles/s41586-018-0414-6 (visited on 2nd Mar. 2022).

[4] *Manage access · jadetherras/bachelor_project*. GitHub. URL: https://github.com/jadetherras/bachelor_project (visited on 8th May 2022).

[5] *A Brief Note on Single-Cell RNA-Seq Analysis*. Zhen He. 21st Feb. 2020. URL: https://zhenhe.github.io/blogs/2020/02/note-on-scrna-seq-analysis/ (visited on 8th June 2022).

[6] *Eukaryotic pre-mRNA processing | RNA splicing (article) | Khan Academy*. URL: https://www.khanacademy.org/_render (visited on 8th June 2022).

[7] *scVelo - RNA velocity generalized through dynamical modeling — scVelo 0.2.5.dev6+g1805ab4.d20210829 documentation*. URL: https://scvelo.readthedocs.io/ (visited on 2nd Mar. 2022).

[8] *rna velocity*. URL: https://www.embopress.org/cms/asset/e9f12612-f65c-4460-9b57-56feb5d6f6aa/msb202110282-fig-0001-m.jpg (visited on 8th June 2022).

[9] Gennady Gorin et al. *RNA velocity unraveled*. Section: New Results Type: article. bioRxiv, 13th Feb. 2022, p. 2022.02.12.480214. DOI: 10.1101/2022.02.12.480214. URL: https://www.biorxiv.org/content/10.1101/2022.02.12.480214v1 (visited on 9th June 2022).

[10] *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction — umap 0.5 documentation*. URL: https://umap-learn.readthedocs.io/en/latest/ (visited on 2nd Mar. 2022).

[11] *Graphing Calculator - GeoGebra*. URL: https://www.geogebra.org/graphing (visited on 12th Mar. 2022).

[12] Leland McInnes, John Healy and James Melville. 'UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction'. In: *arXiv:1802.03426 [cs, stat]* (17th Sept. 2020). arXiv: 1802.03426. URL: http://arxiv.org/abs/1802.03426 (visited on 2nd Mar. 2022).

[13] Mikhail Belkin and Partha Niyogi. 'Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering'. In: *Advances in Neural Information Processing Systems*. Vol. 14. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper/2001/hash/f106b7f99d2cb30c3db1c3cc0fde9ccb-Abstract.html (visited on 28th Mar. 2022).

[14] *Spectral Graph Theory , by Fan Chung*. URL: https://mathweb.ucsd.edu/~fan/research/revised.html (visited on 28th Mar. 2022).

[15] *Endocrine Pancreas — scVelo 0.2.5.dev6+g1805ab4.d20210829 documentation*. URL: https://scvelo.readthedocs.io/vignettes/Pancreas/ (visited on 23rd Mar. 2022).
[16] *Parametric (neural network) Embedding — umap 0.5 documentation*. URL: https://umap-learn.readthedocs.io/en/latest/parametric_umap.html (visited on 4th June 2022).
[17] *Inverse transforms — umap 0.5 documentation*. URL: https://umap-learn.readthedocs.io/en/latest/inverse_transform.html (visited on 9th June 2022).
[18] *Tools for Single Cell Genomics*. URL: https://satijalab.org/seurat/ (visited on 2nd Mar. 2022).
[19] *Simplicial set*. In: *Wikipedia*. Page Version ID: 1067003244. 21st Jan. 2022. URL: https://en.wikipedia.org/w/index.php?title=Simplicial_set&oldid=1067003244 (visited on 3rd Mar. 2022).
[20] *Simplicial complex*. In: *Wikipedia*. Page Version ID: 1070796321. 9th Feb. 2022. URL: https://en.wikipedia.org/w/index.php?title=Simplicial_complex&oldid=1070796321 (visited on 3rd Mar. 2022).
[21] *Markov chain*. In: *Wikipedia*. Page Version ID: 1071684499. 13th Feb. 2022. URL: https://en.wikipedia.org/w/index.php?title=Markov_chain&oldid=1071684499 (visited on 9th Mar. 2022).
[22] *Stochastic process*. In: *Wikipedia*. Page Version ID: 1075991213. 8th Mar. 2022. URL: https://en.wikipedia.org/w/index.php?title=Stochastic_process&oldid=1075991213 (visited on 9th Mar. 2022).
[23] Normalized Nerd. *Markov Chains Clearly Explained! Part - 1*. 25th Oct. 2020. URL: https://www.youtube.com/watch?v=i3AkTO9HLXo (visited on 9th Mar. 2022).
[24] Y. Koren. 'Drawing graphs by eigenvectors: theory and practice'. In: *Computers & Mathematics with Applications* 49.11 (1st June 2005), pp. 1867–1888. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2004.08.015. URL: https://www.sciencedirect.com/science/article/pii/S089812210500204X (visited on 23rd Mar. 2022).
[25] *How To Create a Neural Network In Python – With And Without Keras*. ActiveState. URL: https://www.activestate.com/resources/quick-reads/how-to-create-a-neural-network-in-python-with-and-without-keras/ (visited on 23rd Apr. 2022).
[26] *Quickstart — pca pca documentation*. URL: https://erdogant.github.io/pca/pages/html/Examples.html# (visited on 16th May 2022).
[27] *GeoGebra - the world's favorite, free math tools used by over 100 million students and teachers*. GeoGebra. URL: https://www.geogebra.org/ (visited on 4th June 2022).
[28] Alireza Majidi et al. 'Estimating compaction parameters of marl soils using multi-layer perceptron neural networks'. In: 20 (1st Jan. 2014), pp. 170–198.
[29] *msb202110282-fig-0001-m.jpg (2128×1433)*. URL: https://www.embopress.org/cms/asset/e9f12612-f65c-4460-9b57-56feb5d6f6aa/msb202110282-fig-0001-m.jpg (visited on 8th June 2022).