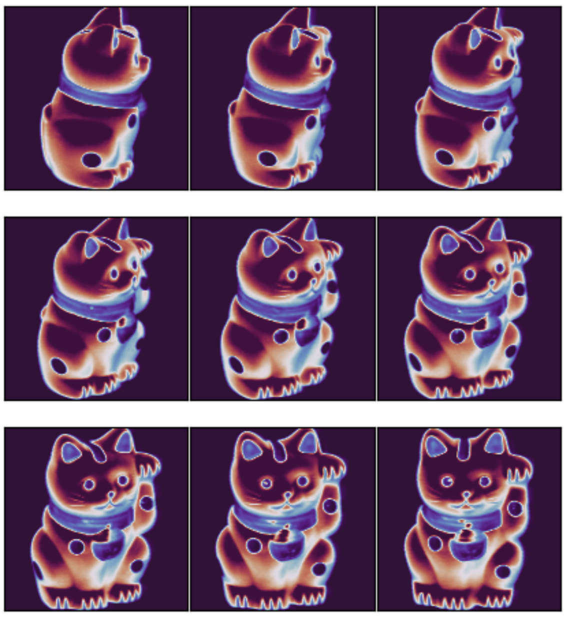
# Laplacian Eigenmap and UMAP algorithms

University of Bologna, Applied Physics
Complex Networks course

Federico Magnani



October 2021

## Contents

## Introduction

In this report the Laplacian Eigenmap and the UMAP algorithms are briefly introduced, recalling their theory and focusing then on their practical implementation. The two algorithms are compared and tested in the task of visualization, on three different datasets: the swiss roll, the mnist and the coil 20. The reader should be familiar to these dataset, that will not be described. A brief insight is made on the topic of the Heat Kernel.

A particular focus is always given to the concepts related to graph theory and if possible the discussions are kept at this level of interpretation. If not possible, the graph analogy is always given. The comparison of the algorithms is mainly the comparison of the graph they build, through concepts such as their weights distributions or the embedding given by the eigenvectors of their Laplacian operators.

# 1 Laplacian Eigenmap

The basic assumption of the field of manifold learning is that some high-dimensional data lie on a (much) lower dimensional submanifold of the ambient space. Moreover, in many practical applications one finds a wealth of easily available unlabeled examples, while collecting labeled examples can be costly and time-consuming. Consequently, it is of interest to develop algorithms that are able to utilize both labeled and unlabeled data for classification and other purposes.
The Laplacian Eigenmap approach leverages the properties of the Laplace-Beltrami operator $\Delta$, naturally defined on Riemannian manifolds, onto or close to which the data are assumed to lie. Such intrisic structure of the dataset can be modeled exploiting both labeled and unlabeled data, through their adjacency graph, i.e. a weighted graph whose vertices are data points. Two data points are connected with an edge if and only if the points are adjacent, which typically means that either the distance between them is less than some $\epsilon$ or that one of them is in the set of n nearest neighbors of the other. The Laplace-Beltrami operator is then approximated by the graph Laplacian L.
In the following the most important properties of the Laplace-Beltrami operator are presented, in order to justify the choices for the implementation and the interpretation of the algorithm.

## 1.1 Theory

A Riemannian manifold is a manifold endowed with a notion of local distance, so that an operator $\Delta$ on twice differentiable functions it's naturally defined. In the case of $\mathbb{R}^n$ it is simply $\Delta = -\sum_i \frac{\partial^2}{\partial x_i^2}$ (note the minus sign).
**Property 1**: when $\mathcal{M}$ is a compact manifold, $\Delta$ has a discrete spectrum and its eigenfunctions provide an orthogonal basis for the Hilbert space $\mathcal{L}^2(\mathcal{M})$. Therefore any function $f \in \mathcal{L}^2(\mathcal{M})$ can be written as

$$f(\mathbf{x}) = \sum_{i=0}^{\infty} a_i e_i(\mathbf{x}) \tag{1}$$

where $e_i$ are eigenfunctions, i.e. $\Delta e_i = \lambda_i e_i$. Note that in the context of classification, the class membership function $m : \mathcal{M} \to \{-1, 1\}$ that assigns each point to a class, usually belongs to $\mathcal{L}^2(\mathcal{M})$.
Let's briefly consider the case in which $\mathcal{M}$ is the circle $S^1$. We have

$$\Delta f(\phi) = -\frac{d^2 f(\phi)}{d\phi}$$

$$\Delta e(\phi) = -\frac{d^2 e(\phi)}{d\phi} = \lambda e(\phi)$$

Clearly the real-valued functions identical in the form to their second derivatives are $e(\phi) = cos(n\phi)$ or $e(\phi) = sin(n\phi)$ with $\lambda = n^2$. (1) in this case becomes

$$f(\mathbf{x}) = \sum_{n=0}^{\infty} a_n sin(n\phi) + b_n cos(n\phi)$$

that's the Fourier series for f. Generalizing this idea for an arbitrary manifold we can expect the eigenfunctions of $\Delta$ to behave as the "oscillatory modes" of the manifold, their eigenvalues being analogue to the "frequencies". Thus to approximate a function of $\mathcal{L}^2(\mathcal{M})$ with the first terms of (1) is close to approximating it with the low frequencies of a Fourier series, the more slowly varying (in a sense, the more smooth).
This (true) intuition can be formalized rigorously introducing the notion of Smoothness. We can now state and give mathematical justification to the
**Property 2**: The basis given by the eigenfunctions of $\Delta$, when used to approximate a function of $\mathcal{L}^2(\mathcal{M})$, provides a maximally smooth approximation of f.
Considering, to fix the ideas, a function f on a unit circle $S^1$, we can measure its degree of smoothness very intuitively through the functional

$$\mathcal{S}(f) = \int_{S^1} \left| \frac{df(\phi)}{d\phi} \right|^2 d\phi$$

With this definition, the most smooth function is the constant function.
Rewriting $\mathcal{S}(f)$ as

$$\int_{S^1} \frac{df(\phi)}{d\phi}\frac{df(\phi)}{d\phi} d\phi$$

and integrating by parts we get

$$\mathcal{S}(f) = f\frac{df}{d\phi}\Big|_0^{2\pi} - \int_{S^1} f\frac{d^2 f}{d\phi^2} d\phi = \langle f, \Delta f\rangle_{\mathcal{L}(S^1)}$$

since 0 and $2\pi$ are the same point on $S^1$. In general for f: $\mathcal{M} \to \mathbb{R}$ it holds that

$$\mathcal{S}(f) = \langle \nabla f, \nabla f\rangle_{\mathcal{L}(\mathcal{M})} = \langle f, \Delta f\rangle_{\mathcal{L}(\mathcal{M})} \tag{2}$$

since the gradient $\nabla$ is adjoint to the negative divergence.
Therefore, the smoothness of the eigenfunctions of $\Delta$ is nothing else than their eigenvalue since

$$\mathcal{S}(e_i) = \langle e_i, \Delta e_i\rangle_{\mathcal{L}(\mathcal{M})} = \lambda_i$$

This fact together with equation (1) allows to write the smoothness of f in the eigenfunctions basis as

$$\mathcal{S}(f) = \langle f, \Delta f\rangle_{\mathcal{L}(\mathcal{M})} = \langle \sum_i a_i e_i, \sum_i a_i \Delta f\rangle_{\mathcal{L}(\mathcal{M})} = \sum_i \lambda_i a_i^2$$

Thus, we can control the smoothness of the approximation employing the first p eigenfunctions of $\Delta$.

**The discrete framework**

In real problems we have to approximate f only knowing its values $f_i$ at a finite set of points $\mathbf{x}_i$. We employ a given number $p$ of eigenfunctions of $\Delta$, that we may call eigenvector in this framework since they assume the form

$$\mathbf{e}_i = [e_i(\mathbf{x_1}), e_i(\mathbf{x_2}), ..., e_i(\mathbf{x_n})]$$

They define the matrix E as $E_{ij} = e_i(\mathbf{x}_j)$. The approximation is defined by the value of the coefficients $a_j$, j = 0, ..., p that are determined such as the values that minimize the following cost function:

$$\sum_{i=1}^{n} \left( f_i - \sum_{j=1}^{p} a_j e_j(\mathbf{x}_i) \right)^2 \tag{3}$$

The solution is analytical: $\mathbf{a}^T = (E^T E)^{-1} E \mathbf{f}^T$, where $\mathbf{f} = (f_1, f_2, ..., f_n)$.
In this discrete framework, the mathematical notions defined previously are represented by parallel notions from graph theory. Consider a graph G with n vertices, one for each data point. Then $\mathbf{f} = (f_1, f_2, ..., f_n)$ is a function defined on the vertices of G of which we can define the smoothness as

$$\mathcal{S}_G(\mathbf{f}) = \sum_{i\ j} w_{ij}(f_i - f_j)^2 \tag{4}$$

measuring it's variation between nearby points.
The graph Laplacian L is defined as L = D - W where D is the degree matrix and W the weight matrix. It's a positive semidefinite operator, thus for the spectral theorem any function of G can be decomposed as a sum of eigenvectors of L. It turns out that

$$\mathcal{S}_G(\mathbf{f}) = \langle \mathbf{f}, L\mathbf{f} \rangle_{\mathbb{E}} = \sum_{i=1}^{n} \lambda_i \langle \mathbf{f}, \mathbf{e}_i \rangle_{\mathbb{E}}$$

which formalizes the concept analogous to the most smooth approximation on $\mathcal{M}$.
The implementation of the algorithm, given in the following section, strictly reflects the line of reasoning outlined above: the dataset is assumed to lie on a manifold that's approximated by the adjacency graph (determined for example with a nearest neighbors process), weighted with some rule. The Laplacian of such a graph is computed and its eigenvectors are used for all sort of applications.

## 1.2 Implementation

The following steps are common to all the possible applications, such as classification, visualization, dimensionality reduction, clustering, etc.

- **1: Adjacency graph computation, through n nearest neighbors**
  The distance used to define the n nearest neighbors and clearly the number of neighbors to employ can be chosen. The standard choice of the distance is euclidean, inherited from the ambient space. These choices are somehow dataset-dependent, but the idea behind is always that data points are connected if they are "close".

- **2: Weighted graph computation**
  To the edges of the adjacency graph are assigned weights depending on the distance between the corresponding points. The possibilities for Laplacian Eigenmap are to employ the heat kernel (see section **1.3**) or a simple connectivity matrix (all existing edges have unitary weights).

- **3: Laplacian graph and eigenfunctions computation**
  From the weighted graph the Laplacian matrix and its eigenvectors are computed. For large datasets that's possible since the nearest neighbors approach leads to sparse graphs, for which specific methods exist.

From this point on many paths are possible. Here the classification and the visualization problems are taken into consideration, being the main concern of the introductory papers of, respectively, Laplacian Eigenmap and UMAP.

- **3a: Classification**

The problem of classification is the problem of approximating the class membership function $m$: $\mathcal{M} \to \{-1, 1\}$, in the case the classes do not intersect. Otherwise, given the classes $S_1$ and $S_2$, it can be $m(\mathbf{x})$ = 1 - 2Prob($\mathbf{x} \in S_1$). In any case assuming that $m$ is a square integrable function on $\mathcal{M}$, applying equation (1), we aim to minimize the error function

$$Err(\mathbf{a}) = \sum_{i=1}^{s} \left( c_i - \sum_{j=1}^{p} a_j e_{ji} \right) \tag{5}$$

where p is the number of eigenvectors we wish to employ, s is the number of labeled points and $c_i$ the corresponding set of labels. Then, the coefficients $\mathbf{a}_i$ defining the approximation of $m$ on $\mathcal{M}$ are $\mathbf{a} = (E^T E)^{-1} E^T \mathbf{c}$, similarly to the case of equation (3). We recall that the matrix E has for rows the eigenvectors of $\mathbf{L}$ evaluated at a set of points (here the labeled points). It will be shown later that the columns of E are strictly related to the embedding coordinates of each point in a lower dimensional space. Once the approximation of $m$ is determined, unlabeled points can be classified accordingly to the prediction the model gives, eventually employing a one-vs-all classifier for each individual class.

- **3b: Visualization**

The problem of visualization can be also stated as the problem of embedding the high dimensional data in 2 or 3 dimensions. The embedding is essentially a mapping $\mathbf{x}_i \rightarrow \mathbf{y}_i$, where $\mathbf{y}_i$ have less components than the original point. Clearly we wish to preserve the information given by the structure of the dataset, thus a "good" mapping can be reasonably determined as the one that minimizes a given cost function, specifically designed to this aim. The following cost function

$$\sum_{ij} (\mathbf{y}_i - \mathbf{y}_j)^2 W_{ij} \tag{6}$$

ask that if $\mathbf{x}_i$ is "close" to $\mathbf{x}_j$ in the high dimensional space, their images $\mathbf{y}_i$ and $\mathbf{y}_j$ must be "close" in the low dimensional space, the "closeness" relation given by the strength of the weight connecting them. Let's notice that a mapping *map*: $\mathbf{x}_i \rightarrow \mathbf{y}_i$ is fully determined by the set of vectors $[\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n]$ i.e., in the case of visualization, by the spatial configuration assumed by the point in the plane or in the space. The visualization problem is thus the choice of such a layout.
For the following discussion, we restrict to the case of the mapping in $\mathbb{R}$. The generalization is conceptually straightforward but much more involved in the notation. That said, it turns out that

$$2\mathbf{y}^T L \mathbf{y} = \sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2\sum_{i,j} y_i y_j W_{ij} = \sum_{ij} (y_i - y_j)^2 W_{ij} \tag{7}$$

Note that the previous computation shows also that L is semidefinite positive. The minimization of (6) is translated into finding $\mathbf{y}$ such that $\mathbf{y}^T L \mathbf{y}$ is minimum, taking care that $\mathbf{y}^T D \mathbf{y} = 1$. The latter constraint removes an arbitrary scaling factor in the embedding. Clearly the minimization of $\mathbf{y}^T L \mathbf{y}$ is equivalent to that of the Rayleigh quotient $\frac{\mathbf{y}^T L \mathbf{y}}{||\mathbf{y}, \mathbf{y}||}$, fact that allows to restate the problem one more time, as the finding of the minimum eigenvalue solution of the following generalized eigenvalue problem

$$L\mathbf{y} = \lambda D \mathbf{y} \tag{8}$$

A trivial solution is to set $\mathbf{y} = \mathbf{1} = (1, 1, ..., 1)$, corresponding to a mapping that collapses all the points into 1. Being a constant function it should have $\lambda = 0$ and in fact, by construction, $L\mathbf{1} = \mathbf{0}$. Adding the constraint that $\mathbf{y}^T D \mathbf{1} = 0$, the solution is given by the eigenvector with the smallest nonzero eigenvalue. This can be interpreted as asking the orthogonality to $\mathbf{1}$, i.e. as removing a translation invariance in $\mathbf{y}$.
The general case has the form

$$\sum_{ij} ||\mathbf{y}_i - \mathbf{y}_j||^2 W_{ij} = tr(\mathcal{Y}^T L \mathcal{Y}) \tag{9}$$

where $\mathcal{Y}$, the map from the high- to the low- dimensional space, is defined by all the positions $\mathbf{y}_i = (\mathbf{y}_1(\mathbf{x}_i), \mathbf{y}_2(\mathbf{x}_i), ..., \mathbf{y}_m(\mathbf{x}_i))^T$ of the input points. Explicitly $\mathcal{Y}_{ij} = \mathbf{y}_j(\mathbf{x}_i)$. The constraint in this setting is that $\mathcal{Y}^T D \mathcal{Y} = I$, and it prevents the mapping onto a subspace of dimension less than m-1 (or less than m if also the orthogonality to $\mathbf{1}$ is added).
Note that equation (8) can be rewritten as

$$D^{-1} L \mathbf{y} = \lambda \mathbf{y}$$

where $D^{-1}L$ is similar to the diffusive Laplacian $L_T$:

$$D^{-1} L = D^{-1} L D^{-1} D = D^{-1} L_T D$$

Another relation of L to diffusive processes, that has a central role in the algorithm implementation, is given in the following section.

## 1.3 Heat Kernel

The Laplace-Beltrami operator is intimately related to the heat flow on $\mathcal{M}$, through the heat equation

$$\left(\frac{\partial}{\partial t} + \Delta\right) u = 0 \tag{10}$$

If $f$:$\mathcal{M} \to \mathbb{R}$ is the initial heat distribution, the solution of (10) is given by

$$u(x,t) = \int_{\mathcal{M}} H_t(x,y) f(y)$$

where the heat kernel $H_t$ is the Green's function of the (partial differential) heat equation. We strictly follow the argument of Belkin and Niyogi in [1]. In an exponential basis, the heat kernel is approximated by a gaussian as

$$H_t(x,y) = (4\pi t)^{-\frac{m}{2}} e^{-\frac{||x-y||^2}{4t}} (\phi(x,y) + o(t)) \tag{11}$$

where $\phi$ is a smooth function equal to 1 if x=y. Since we assume that x is close to y, for small t it is

$$H_t(x,y) \cong (4\pi t)^{-\frac{m}{2}} e^{-\frac{||x-y||^2}{4t}} \tag{12}$$

Since it is also

$$\Delta f(x) = -\frac{\partial}{\partial t}\bigg|_{t=0} \int_{\mathcal{M}} H_t(x,y) f(y) \tag{13}$$

employing again the small t approximation and the definition of derivative we get

$$\Delta f(x) \cong \frac{1}{t}\left[ f(x) - (4\pi t)^{-\frac{m}{2}} \int_{\mathcal{M}} e^{-\frac{||x-y||^2}{4t}} f(y) dy \right] \tag{14}$$

that in the finite case must be discretized as

$$\Delta f(\mathbf{x}_i) \cong \frac{1}{t}\left[ f(\mathbf{x}_i) - \frac{1}{k}(4\pi t)^{-\frac{m}{2}} \sum_{i\ j} e^{-\frac{||\mathbf{x}_i-\mathbf{x}_j||^2}{4t}} f(\mathbf{x}_j) dy \right] \tag{15}$$

Now let's define

$$\alpha = \frac{1}{k}(4\pi t)^{-\frac{m}{2}}$$

If $\mathbf{x}_i = \mathbf{x}_j$ for each i and j, the left term is identically 0. Consequently it must be

$$\alpha = \left( \sum_{i\ j} e^{-\frac{||\mathbf{x}_i-\mathbf{x}_j||^2}{4t}} \right)^{-1}$$

This argument is the justification to compute the graph Laplacian with weights given by

$$W_{ij} = \begin{cases} exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{4t}) & \text{if } \mathbf{x}_i \text{ close to } \mathbf{x}_j \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

At the level of implementation and practical usage, t is a free parameter that can be tuned in order to focus on the local behaviour or on the global one.
If t tends to 0 also the weights tend to 0, except the ones for which the distance is very small. So tuning t towards small values is similar to setting a smaller (sort of) threshold distance, beyond which the weight is set to 0. That's, conceptually, a similar effect to employing a lower number of nearest neighbors and both these choices force the algorithm to focus on more local features and smaller scale topology. On the other hand, in the limit for large t, all the weights are set to 1. That's equivalent to employ the distances between the data points only to compute the nearest neighbors graph and then forget that information. The resulting graph is also called "connectivity graph".
It's clear from the analytical form of the weighting function that the choice of t is highly dependent on the distances distribution of the dataset: so it's highly dependent on the dataset itself. Also the scale of the dataset plays a role in the correct guess of t and the exponential form of this interplay leads to a high sensitivity of the results with respect to the choice of the parameter.
It must be stated, finally, that even if the theoretical argument in favor of this definition lies onto a series of approximations for small t, absolute values for the parameter are not defined. In practical implementations it's treated as a free parameter spanning to infinite.

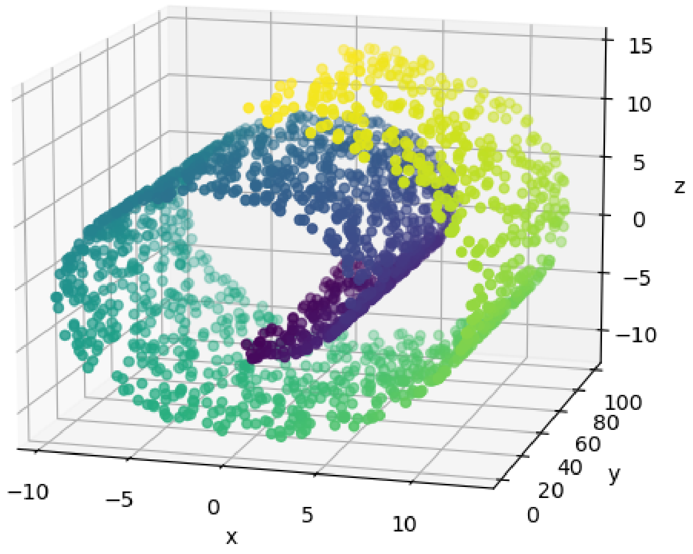


Figure 1: The Swiss roll dataset, with 2000 points. Note that the y axis is greater by an order of magnitude to the other axes.

## 1.4 Results

The Laplacian Eigenmap algorithm has been tested in the task of visualization, following [1]. The same dataset of the paper has been chosen, namely the Swiss roll dataset represented in Figure 1. The aim of the algorithm is, roughly speaking, to "unroll" the dataset i.e. to identify the 2-dimensional manifold on which the 3-dimensional data lie. The first step is the definition of the N nearest neighbors graph. If the N parameter is too high, long links between different "layers" of the roll will be identified and the embedding will perform badly. In any case, even for small N, it's very hard to stop the algorithm to make some "wrong" connection in the extremal portion of the dataset, since there it is a lower density of points. For example, the yellow points at the edge of the roll will include in their nearest neighbors with high probability some blue point, directly below them. In the embedding this result in a sort of attraction felt by these classes of points. Notice that the scale of the y axis is greater by an order of magnitude with respect to the others. In this situation the algorithm works as expected (Figure 2), but if the dataset is normalized, the embedding changes dramatically.

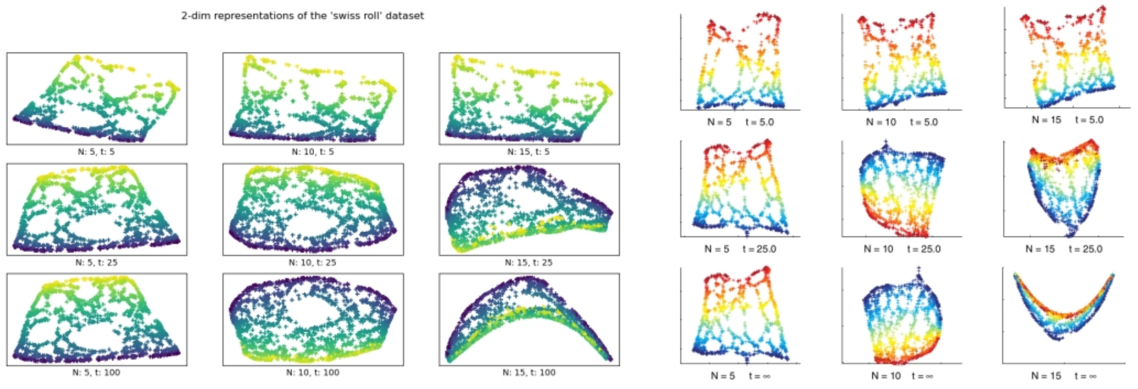


Figure 2: Left: my results, Right: the results by Belkin and Niyogi ([1]). The number of neighbors used are 5, 10 and 15, while t has been set to 5, 25 and 100 (to be interpreted as "very large").

In Figure 2 the results for the Swiss roll datasetare shown, for both my implementation and the one by Belkin and Niyogi. Along the columns the number of nearest neighbors used is increased, while along the rows the time parameter is increased. For small t (top row) the embed is nice for any number of neighbors, but also the analogous is true: for a small number of nearest neighbors employed (left column) the result is nice for every value of t. In both the cases the algorithm had a very local focus and the links have correctly been made only along the true dimension of the manifold. In these cases, the shortest path between two points approximates the geodesic distance on the manifold. It can be shown that for the limit of an infinite number, the shortest path converges to the geodesic. The performance of the algorithm gets worse if either the number of neighbors or the time parameter are increased.
Let's consider the case in which the dataset is homogeneous in scale along all the three axes. In the left pane of figure 3 the scale is lower, in the right pane is higher. The topology of the dataset is the same as before and between the two panes of Figure 3 only a global scaling has been applied. The distribution of the distances has changed with respect to the dataset of figure 1 and the time parameter should change accordingly. Anyway, we have no clues to find its most suitable value, apart its proportionality to the squared of the distance. Moreover, to simply guess the parameter leads often to numerical instability.

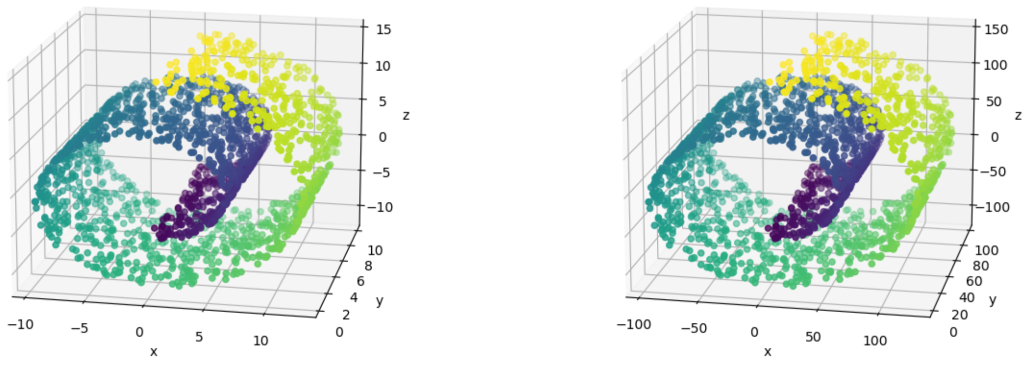


Figure 3: The homogeneous Swissroll dataset, with diminished (left) and increased (right) scale.

In the figure 4 the Laplacian Eigenmap has been applied to the small (left) and to the big (right) homogeneous datasets. The time parameter has been chosen empirically (0.5, 2.5 and 10 for the small dataset, 50, 250 and 1000 for the big one). The results in the two cases are really similar. One of the two intrinsic dimensions of the manifold is accurately captured, while the other is totally collapsed. For none of the tested combination of nearest neighbors number and time parameter (spanning three orders of magnitude) the results could change. This fact could mean that the limitation is intrinsic to the dataset structure. The following example could enforce this idea.

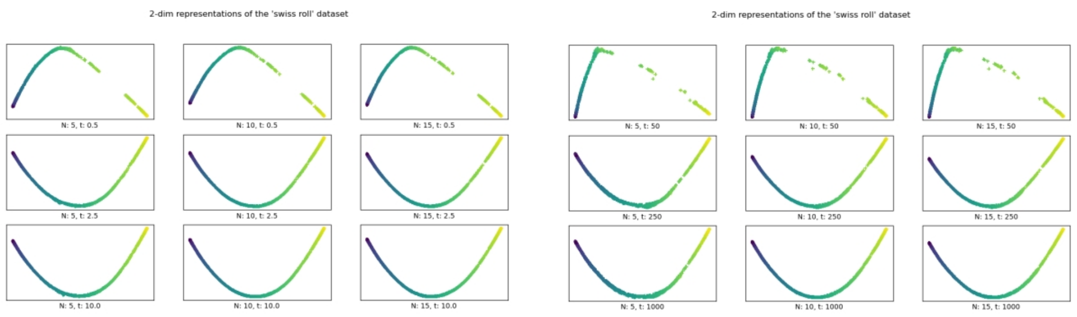


Figure 4: The resulting embedding of the diminished (left) and increased (right) homogeneous versions of the Swissroll dataset, for the same neighbors parameters of Figure 2.

To conclude the set of experiments regarding the scale of the dataset, a last situation has been tested. Instead of the y axis, either the x or the z axis is greater by an order of magnitude. The embedding given by Laplacian Eigenmap is shown in Figure 5: on the top pane the x scale is increased and in the bottom pane the dataset is stretched along the z axis. 10 nearest neighbors have been used and infinite time. Yet as in the homogeneous case, the algorithm could find only one intrinsic dimension of the manifold. In both the versions the y axis were small compared to one of the others and it has been neglected.
In conclusion, Laplacian Eigenmap is highly dependent on the relative scales of the independent variables of the dataset. In some cases, such as the one shown in Figure 5, this could be expected and even appreciated, since the embedding correctly represented the main direction of variability of the dataset, treating the fluctuactions in the other axis as a sort of noise. In the homogeneous case though, I think that it would be valuable to retain nice results. It's easier to normalize the dataset and then apply a given method, than to adapt the parameters each time to a different dataset with few clues on this choice. That said, it's even more surprising the high quality of the embedding obtained for the particular ratio of scales of Figure 1. In that case the main direction of variabilty was the y axis and the nonlinear folding of the points was contained in a really small area (in the xz plane), with respect to the global span of the dataset.

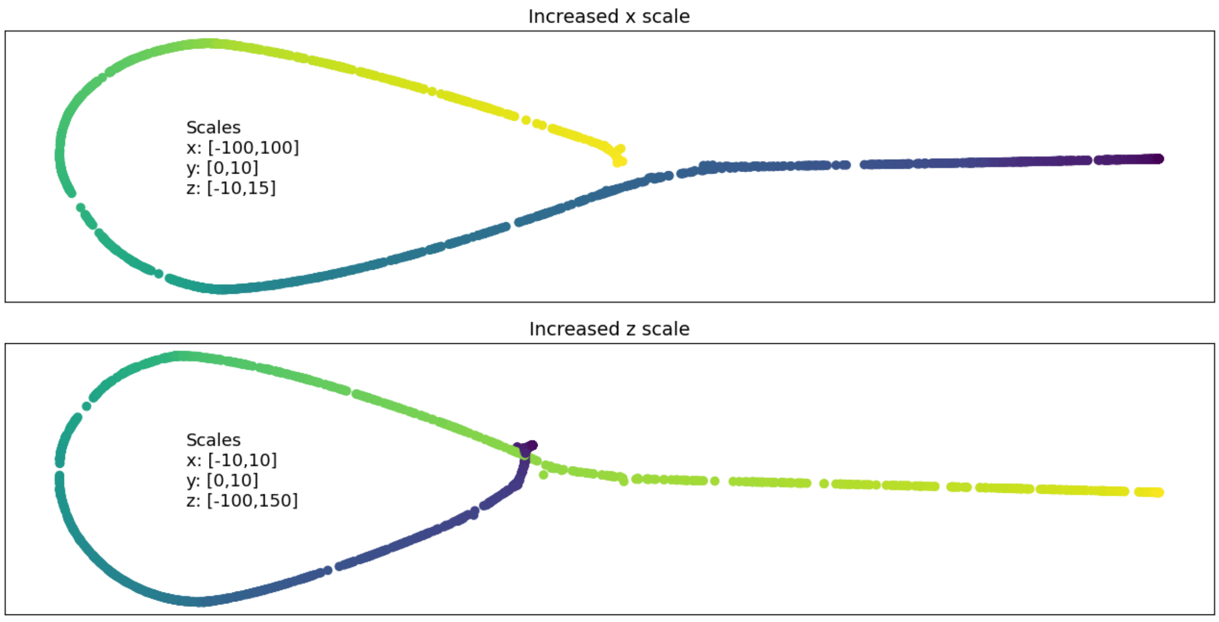


Figure 5: The results of the Eigenmap Algorithm applied to different versions of the Swissroll dataset, stretched along the x or the z axis. 10 nearest neighbors have been used and infinite time.

Recall that the first step of the algorithm is to compute the square matrix of the mutual distances of the data points. From this matrix the N nearest neighbors of each point are computed and a graph is built, whose vertices are the data points, connected by an edge to all their nearest neighbors with weight equal to their distance. Applying to each of these weight the heat kernel relation 16, a new graph is computed that we may call the heat kernel graph. In Figure 6 these two graphs are taken into account through their weights distributions. In the top pane, each row refers to a different number of nearest neighbors employed. On the left column the distances distribution (the weights distribution of the nearest neighbors graph) are shown. The dataset is always the same, but to employ a greater number of neighbors cause the average distance to slightly increase. However, even small changes in the distances distribution induce big differences in the resulting heat kernel graph. In the top pane, at the right of each distance distribution of the nearest neighbors graph, the corresponding weights distributions of the heat kernel graph are represented, for three values of the time parameter. $t$ increases from blue to green to red. Fixing the attention on a particular color of distributions, we see that increasing the number of nearest neighbors (i.e. increasing slightly the distances) the weights are pulled towards 0. On the other hand, for a given neighbors graph, to increase the time is equivalent to pushing all the weights of the heat kernel graph towards 1. This behaviour depends entirely on the formula 16.

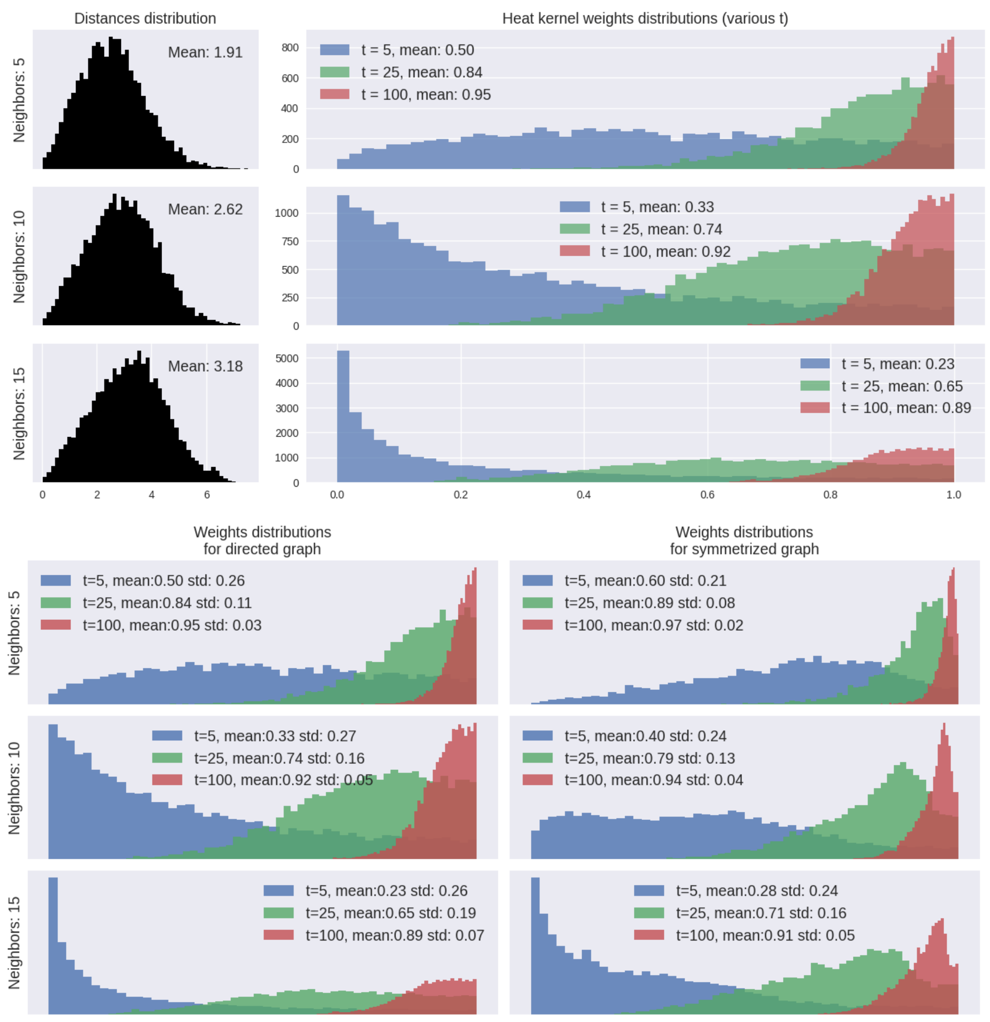


Figure 6: Top pane: distances distributions of the nearest neighbors graph (left) and weights distributions for the resulting heat kernel graph (right). Bottom pane: comparison between the weights distributions of the heat kernel computed from a directed (left) and symmetrized (right) nearest neighbors graph.

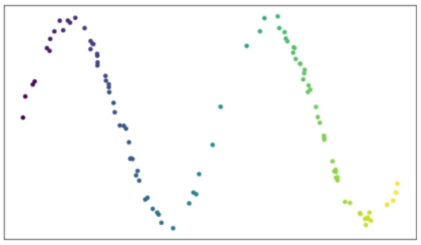
In the bottom pane of Figure 6 the effects of the symmetrization are analyzed. The nearest neighbors graph is an asymmetrical relation, in the sense that x can be in the neighbors set of y without the opposite to hold. Since the Laplacian graph needs to be symmetric (just think for example to the spectral theorem), the nearest neighbors graph is symmetrized. Here we employed a standard symmetrization given by the multiplication by its transpose and a divison by 2. Then the heat kernel is applied to the "original" (not symmetrized) and to the symmetrized graph and the resulting weights distributions (for different numbers of neighbors and time parameters) are compared in the bootom pane of the figure. On the left the heat kernel's weights resulting from a directed neighbors graph are shown, on the right the same distributions as obtained from a symmetrized neighbors graph. We could say concisely that the symmetrization causes the final weights distributions to get more narrow (always the mean increases and the standard deviation diminishes, slightly). Overall the difference is not dramatic, nevertheless it's not very justified by the theory.

# 2 UMAP
# Uniform Manifold Approximation and Projection

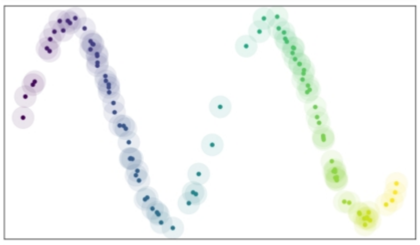
UMAP is a machine learning technique specifically designed for dimensionality reduction, fast and easy scalable to massive datasets. Its theoretical framework is based on Riemannian geometry and Algebraic topology and it's built directly on the top of the work of Belkin and Niyogi, the authors of Laplacian Eigenmap. In fact it's more an expansion and an improvement of Laplacian Eigenmap, rather than a different approach. The key passages of both the algorithms are the same, but the singular implementation choices are different and have different interpretations. The embedding of a graph in the sense of Laplacian Eigenmap is still present, but only as an intermediate step of the whole UMAP pipeline.
In this section both the theoretical and the practical implementation of UMAP are exposed, as always with a particular focus for graph related concepts and interpretations. Despite the high level of formalization of the UMAP's theory, which makes use of rather exotic notions for the working physicist as category theory and fuzzy sets, all boils down to a quite simple implementation. The main concern of the following presentation will be to understand the reasoning leading to the practical implementation choices, the assumptions underpinning the algorithm and the conceptual thread above the mathematical rigour. We anticipate that UMAP consists of two main sections: the computation of a graph representing the dataset and the optimization of its low dimensional embedding through the numerical minimization of a cost function. We focus mainly on the former step, briefly mentioning all the rest. Similarly, the comparison between Laplacian Eigenmap and UMAP will be realized mainly in terms of the graph they build and their differences.

## 2.1 Theory

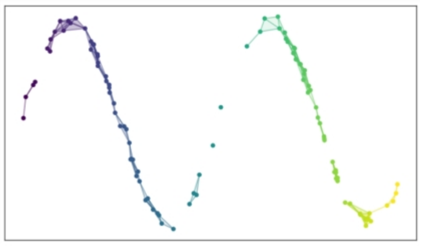
UMAP's conceptual backbone is really similar to that of Laplacian Eigenmap: the data, given in a high dimensional ambient space, are assumed to lie on a lower dimensional manifold. Such a manifold is approximated by a graph having as vertices the datapoints. The dimensionality reduction is made with a mapping, from the graph to a target space, determined by a cost function that defines which quantity we wish to preserve. Recall that Laplacian Eigenmap works with the underlying assumption that the points are homogeneous on the manifold $\mathcal{M}$. Often that's not the case in real situations and UMAP starts from this problem, addressing it directly.



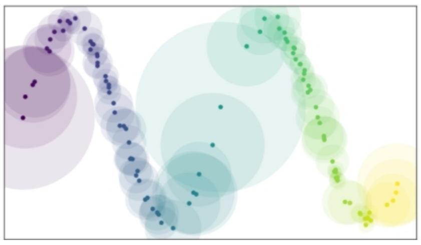
Imagine this to be the dataset: the ambient space is 2-dimensional but the data lie on a 1-dimensional manifold. The data are not homogeneous on $\mathcal{M}$.



Usually the determination of the nearest neighbors graph is the starting step of manifold learning algorithms, but here its mathematical justification and interpretation are given too. In order to do this, let's look for a cover of the manifold $\mathcal{M}$, through a set of balls centered on the datapoints.



The choice of a radius is not simple in not homogeneous cases: in the example, a radius good for some areas is too small for others, leading to a not fully connected graph (we will shortly give the rule to associate a graph to a covering).



We can *assume* that the points are homogeneous with respect to a suitable varying metric on the manifold. This means that we employ a cover of balls with different dimensions, as measured by the ambient space, but all equal as measured by the Riemannian metric of the manifold, which is different from point to point.

This construction just is the N nearest neighbors approach usually employed.

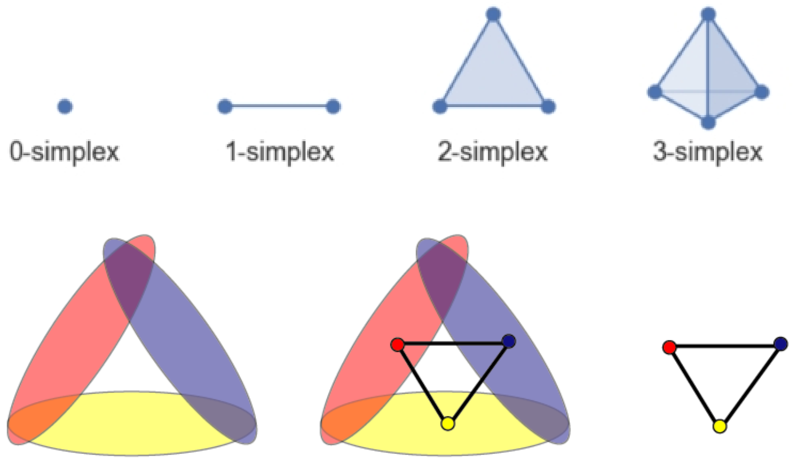


Figure 7: Top: first four simplices. Bottom: Representation of the Nerve theorem associating a graph to an open cover.

The Nerve theorem gives a rule to associate a simplicial set to the covering of a manifold, in a way that preserves the topological properties of the manifold. For our purposes, we can think of a simplicial set as the generalization of a graph that includes, besides points (0-simplices) and edges (1-simplices), also 2-simplices (triangular faces), 3-simplices (volumes of tetraedrons), and so on (see Figure 7, top). Therefore if, having a set of vertices, a graph is defined claiming what edges exist (each edge identified by two vertices), a simplicial set is defined claiming also what faces exists (defined by three vertices), what volumes (defined by four vertices) and so on. It's clear that from a computational point of view it's unfeasible to employ all this information. In the pratical implementation of the algorithm, in fact, only 0- and 1- simplices are employed, that are nothing else than a usual graph. That said, we state the Nerve theorem as the rule to associate a graph to a manifold covering, represented in Figure 7. Roughly speaking, a vertex is associated to the center of each ball and an edge is traced between two vertices, if their balls do intersect.

Up to now, we justified the n nearest neighbors approach using algebraic topology and Riemannian geometry. A neighbors graph can be interpreted as a set (one for each data point) of metric spaces, each one with its local notion of distance. We need to make the union of these spaces in order to recover a global description of the dataset, but this cannot be done in the mathematical framework of metric spaces. For this reason, we shift our interpretation from metric spaces to fuzzy sets, the union of which is intuitive and well defined.
McInnes employs category theory to formalize this change of perspective, but we can

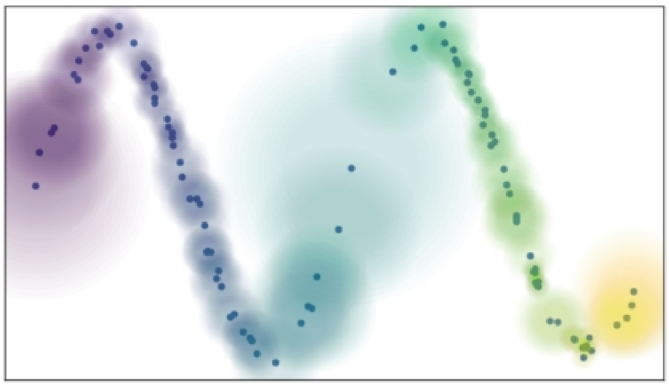
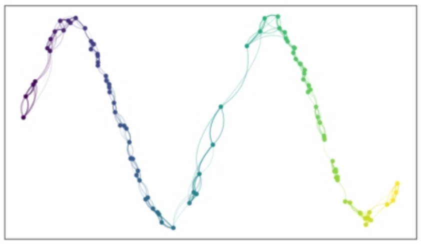


Figure 8: Representation of an open cover of the manifold given by fuzzy sets, rather than classical sets. The shading indicates that the membership function of each set of neighbors of $x_i$ decay as the distance from $x_i$ increases.

avoid this level of abstraction without loosing any insight in the algorithm. At a practical level, all boils down to determine the weights to give to the N nearest neighbors graph, starting from the mutual distances of the points. We will proceed the discussion at this level of understanding.
In concrete, with "metric spaces" we refer to the set of the distances from each point to its N neighbors, i.e. to the neighbors graph with edges weighted by the distances. Note that the notion of distance on the manifold is different from point to point, therefore even if x and y are mutually in the set of neighbors of the other, in general $d_{\mathcal{M}}(x, y) \neq d_{\mathcal{M}}(y, x)$. In graph terms, we have a directed graph that we wish to symmetrize. Now, consider that the membership function of each point to the neighbors set of another one is a boolean function, i.e. either x is in the neighbor set of y or it is not. That holds for classical sets, while the membership function of a fuzzy set is a continuous value between 0 and 1: we have a sort of "confidence" that the point belongs to that set. Applying this notion to the neighbors graph is the next step of the discussion: to each edge x$_\sim$y we must assign the value representing the degree of membership of x to the set of nearest neighbors of y. Once that is done, we will have successfully shifted the interpretation from metric spaces to fuzzy simplicial sets. In this new framework it's straightforward to realize their union (to symmetrize the directed graph) while for their metric counterparts we had no options. Note that a very similar passage was made in Laplacian Eigenmap through the heat kernel, after having symmetrized the neighbors graph weighted by the distances.

Note that these steps are applied sequentially in the implementation: the classical neighbors graph is the first step to be made. Therefore both Laplacian Eigenmap and UMAP start from the very same graph, weighted by the distances with respect to the ambient space. The second step of both is to transform the weights of such a graph with a given rule: the heat kernel for Laplacian Eigenmap, the fuzzy membership value for UMAP. Conceptually also the embedding is really the same, except for the cost function used to determine the optimal low dimensional layout. In order not to loose the thread of the discussion, the weighting function used by UMAP will be examinated later in more detail.

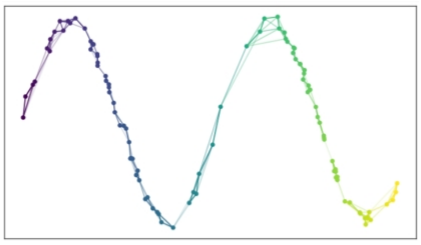
Having mapped distances to fuzzy memberships, we now have a directed graph with weights between 0 and 1. We can think of these weights as akin to the probability that the given edge exists.



The last step is to make the union of the local fuzzy sets i.e. to symmetrize the graph. Fuzzy set theory suggest the correct way, corresponding to compute the probability that at least one of the edges exists:

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

where $\circ$ is the Hadamard product.
Finally the symmetric graph representing our data, designed to preserve the topological information, has been computed.



From this point on, many possibilities arise such clustering, classification, etc. UMAP focuses on the tasks of dimensionality reduction and visualization, that can be stated as the problem of finding the optimal mapping of the data to a lower dimensional space. The "optimality" refers to the conservation of the kind of information we are interested in. Laplacian Eigenmap concerns with respecting the local metric of the dataset: if $\mathbf{x}_i$ is close in the high dimensional space to $\mathbf{x}_j$, then $\mathbf{y}_i$ must be close to $\mathbf{y}_j$ in the embedding (note that nothing prevents very distant points to be mapped closely). Therefore the function that Laplacian Eigenmap seeks to minimize is

$$\sum_{ij} ||\mathbf{y}_i - \mathbf{y}_j||^2 W_{ij} \tag{17}$$

Since UMAP uses fuzzy neighbors graphs instead of metric notions, it needs a mean to compare fuzzy representations. Differently from the high dimensional case, in which the manifold had to be approximated, the low dimensional manifold in which UMAP embeds the data set can be chosen arbitrarily. Then the fuzzy topological representation of the embedding can be computed and compared to the high dimensional one. Two fuzzy graphs $(A, \mu)$ and $(B, \nu)$ can be compared via their Cross Entropy, defined as

$$C((A, \mu), (B, \nu)) = \sum_{a \in A} \left( \mu(a) log\left(\frac{\mu(a)}{\nu(a)}\right) + (1 - \mu(a)) log\left(\frac{(1 - \mu(a))}{(1 - \nu(a))}\right)\right) \tag{18}$$

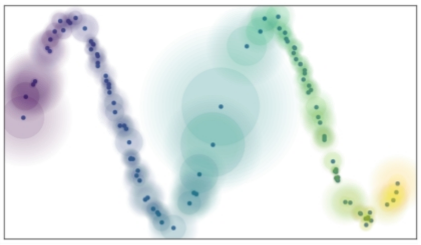
The solution is not analytical as in the case of (17) and UMAP employs a numerical minimization, leading to a force directed graph layout algorithm in the low dimensional space, with a set of attractive forces applied along edges and a set of repulsive forces applied among vertices. The non-differentiable weighting function, shown later, is fitted by a family of smooth curves, in order to allow the use of the Stochastic Gradient Descent method. The non-convexity of the optimization is addressed with a strategy similar to the simulated annealing.

## 2.2 Implementation

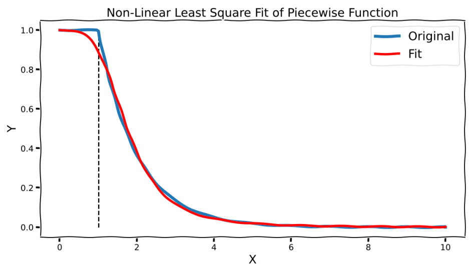
Given a nearest neighbors graph weighted by the distances, we compute the weights of its associated fuzzy graph through the following function:

$$p_{i|j} = w_{(x_i,x_{i_j})} = \begin{cases} exp(-\frac{d(x_i,x_{i_j})-\rho_i}{\sigma_i}) & \text{if } d(x_i,x_{i_j}) \geq \rho_i \\ 0 & \text{if } d(x_i,x_{i_j}) < \rho_i \end{cases} \tag{19}$$

where $x_{i_j}$, j=1,2,...,k denotes the set of k neighbors of the point $x_i$. Intuitively, one can think of the weight of an edge as akin to the probability that the given edge exists. $d(x,y)$ is the distance in the metric of the ambient space and the weight of the edge i $\sim$j of the distance neighbors graph. $\rho_i = d(x_i,x_{i_1})$ is the distance from $x_i$ of its first nearest neighbors, while $\sigma_i$ is the fuzzy analogous of the distance of its $k^{th}$ neighbor. The quantity $\frac{d(x_i,x_{i_j})}{\sigma_i}$ approximates the geodesic distance in $\mathcal{M}$ and note that it's different from point to point, since we're employing local metrics defined by the local parameters $\rho_i$ and $\sigma_i$. Overall the function is identically zero up to its first nearest neighbor and an exponential decay starting from that point on. In this way we have confidence = 1 that each point is connected at least to another one point, i.e. that there aren't isolated points.



Here it's depicted the behaviour of the function (19): the connectivity of the graph is constrained setting unitary membership up to the first neighbor and letting it decay from that point on.



The particular shape of the decay is different from point to point, being different the $\rho$ and $\sigma$ parameters. In order to apply later the SGD minimization, a fit is performed with a specific family of smooth curves.

Also another important problem is addressed by the choice of measuring the decay from the first neighbor of the point rather than from the point itself: the curse of dimensionality. One aspect of this famous concept is that in high dimensional spaces "all the distances looks the same", in the sense that the normalized distribution of the distances of a bunch of random points is very narrow. The distances tends to be larger, but also more similar one to the other. Therefore, to employ an exponential function and to compute the difference in distances among nearest neighbors rather than the absolute distance helps in discriminating between close points in high dimensional spaces: the

distance to the the first nearest neighbor could be quite large, but the distance from the $10^{th}$ nearest neighbor often is only slighlty larger.
The cardinality of a finite classic set is the number of its elements. If we would like to set a radius for which a ball centered in a point contains exactly k other points, that would simply be the distance of the center point from its $k^{th}$ nearest neighbor. What about the cardinality of a finite fuzzy set? That's defined as

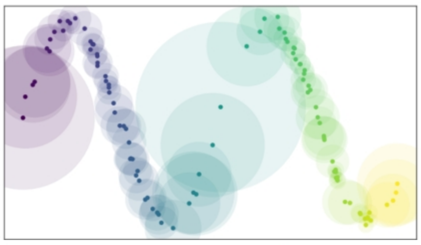
$$|A| = \sum_{x \in U} \mu_A(x) \tag{20}$$

where U is the finite support of A and $\mu_A$ its membership function. That said, $\sigma_i$ is a parameter automatically determined by the algorithm requiring that

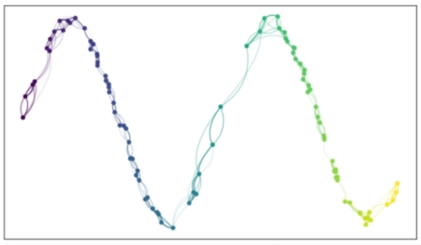
$$\sum_{j=1}^{k} exp\bigg( - \frac{d(x_i, x_{i_j}) - \rho_i}{\sigma_i} \bigg) = log_2(k) \tag{21}$$

where k is the chosen number of nearest neighbors of each point. So $\sigma_i$ is defined computing the distance such that the cardinality of the generated fuzzy set is $log_2(k)$. The selection of $\sigma_i$ corresponds to a smooth normalization factor. In this way we trade choosing the radius of the balls for choosing a value for k. However it is often easier to pick a resolution scale in terms of number of neighbors than it is to correctly choose a distance, since choosing a distance is very dataset dependent. A very similar argument could be formulated for the time parameter of the heat kernel. Moreover we get a further insight in the parameter k: its choice determines how locally we wish to estimate the Riemannian metric. Small values of k translate in a very local focus while larger values tend to estimate the manifold with a larger resolution, trying to understand more global features.
To conclude, we wish to mention that the distance $d(x, y)$ of the ambient space is usually euclidean, but there's nothing preventing other choices. In fact, that's one of the parameters of UMAP, like the metric of the space in which the embedding is made. Also note that despite both the notation and the interpretation suggest the weight of the fuzzy graph to be a probability, it's not normalized. The function goes "naturally" from 0 to 1, but there is no normalization applied to it. This saves a lot of computational time. Due to the local parameters $\rho$ and $\sigma$, $p_{i|j}$ is in general different to $p_{j|i}$ and also the interpretation of the symmetrization process is given in probabilistic terms.

## UMAP algorithm - scheme

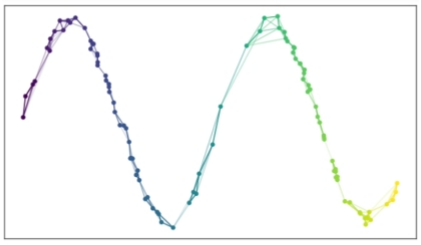


**1. K nearest neighbors graph**
(weighted by distances)



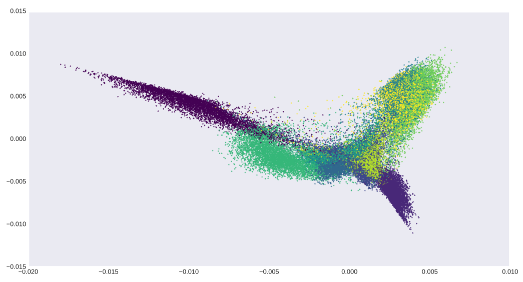
**2. Directed fuzzy graph**

$$exp(-\frac{d(x_i, x_{i_j}) - \rho_i}{\sigma_i})$$

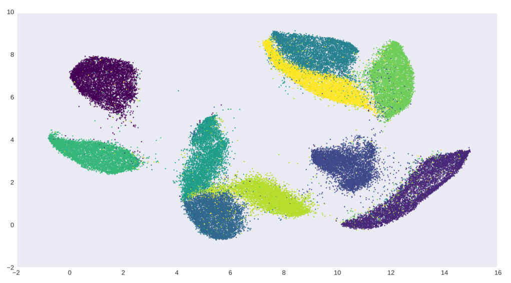


**3. Symmetrization (Fuzzy set union)**

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



**4. Spectral embedding of the graph**
(Inizialization of the layout)



**5. Optimization of the layout**
(minimizaton of Cross Entropy)

## 2.3 Results

In the following section some results are shown using UMAP on two classic datasets, the MNIST and the COIL20. The final layout presented by UMAP is given by the optimization, with respect to the cross entropy, of an initialization layout, that occurs to be the laplacian embedding of the fuzzy topological graph. This choice is made both for reasons of stability and performance. Since as usual a particular care for the graph related aspects is taken, this laplacian embedding will be always computed individually, beside its optimization that is the "true" output of UMAP. With laplacian embedding we mean the embedding given by the projection of the data points onto the first components of the eigenvectors of the Laplacian graph. That's essentially the same theory underpinning Laplacian Eigenmap and that's the reason for which we stated that UMAP is an expansion of Laplacian Eigenmap, rather than an alternative.

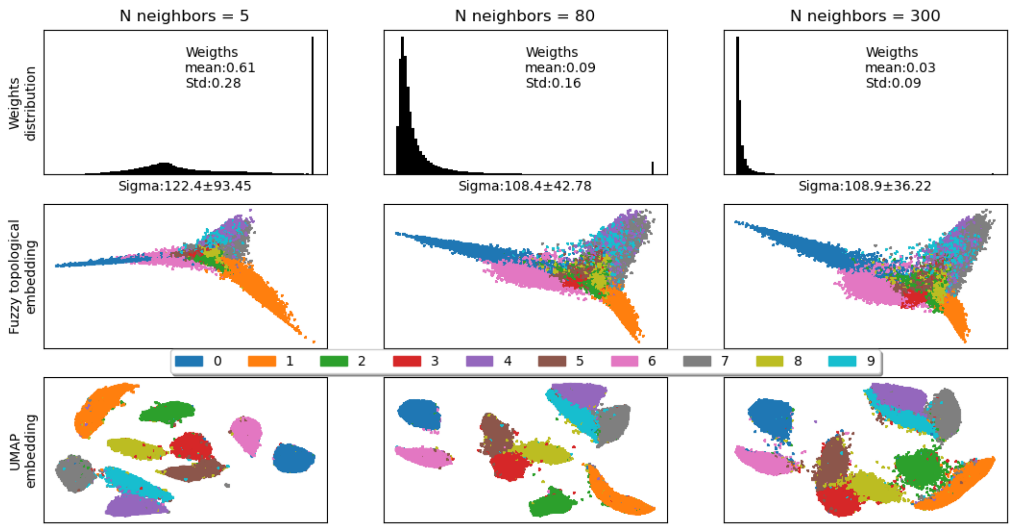


Figure 9: The MNIST dataset has been employed. For different values of nearest neighbors the resulting weights distribution of the graph built by UMAP, its spectral embedding (called "fuzzy topological") and the final optimization of the layout are shown.

### 2.3.1 MNIST dataset

Despite a high variability in the weights distribution, the final embedding is very stable. Notice that both the local and the global features of the dataset are taken into account by the algorithm. For example, the cluster of the zeros (blue) is always opposite to the cluster of the ones (orange) and always close to the cluster of the sixes (pink), due to their mutual similarities. This kind of inter-cluster relationships, holding even for the other clusters, are examples of global features of the dataset. Local features are the presence of the clusters and their shapes, that somehow reflect the intra-cluster variability of the points. For example, the ones have a main direction of variability (maybe the angle of the vertical bar) while the zeros are in the shape, approximately, of a bidimensional gaussian, because there are relatively few degrees of freedom over how a person draws a number one.
In this case the number of neighbors employed didn't change particularly the results. Both the optimized layout and the laplacian embedding of the graph are similar for a very large span of values. Nevertheless, the distribution of the weights of such a graph are quite different between N $= 5$ and greater values. That seems to affect only slightly the results.

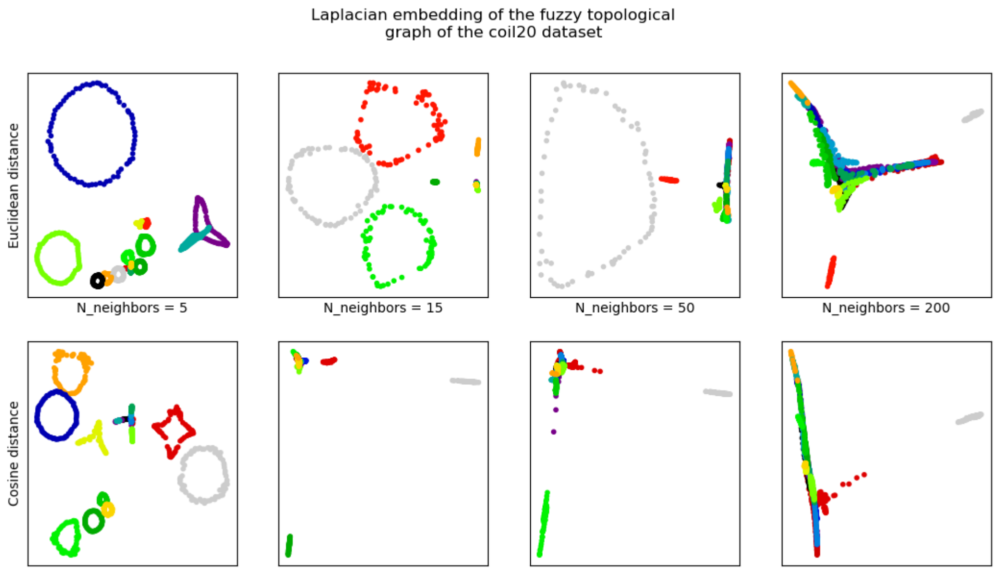


Figure 10: Spectral embedding of the fuzzy topological graph representing the COIL20 dataset, using the Euclidean distance (top row) and the Cosine distance (bottom row), for different number of neighbors.

### 2.3.2 COIL 20 dataset

The COIL20 dataset has been employed in this example, with valuable results. In Figure 10 the laplacian embedding of the fuzzy topological graph is shown, for two different metrics assigned to the ambient space. We recall that the COIL20 dataset has an intrinsic rotational topology, due to the sets of different and connected views associated to each object. Therefore, there are natural clusters given by the different objects and a natural circular topology of each of them. The cosine distance has been chosen due to this angular structure, but the euclidean distance seems to perform better for any number of neighbors. For both the choices, a smaller number of neighbors is more suitable. The most surprising features are the representation given by the algorithm to what we called the "natural" structure of the data set. A number of circles arise, each point of them being a particular view of the same object. Different circles are associated to different objects. Only few of them, as the purple and the turquoise in the left upper pane, could not be disentangled. Note that in these images, no optimization has been applied: that's just the projection of the data points onto the first two components of the eigenvectors of the graph Laplacian associated to the fuzzy topological graph. In Figure 11 the minimization of the cross entropy has been performed starting from the layouts of Figure 10.

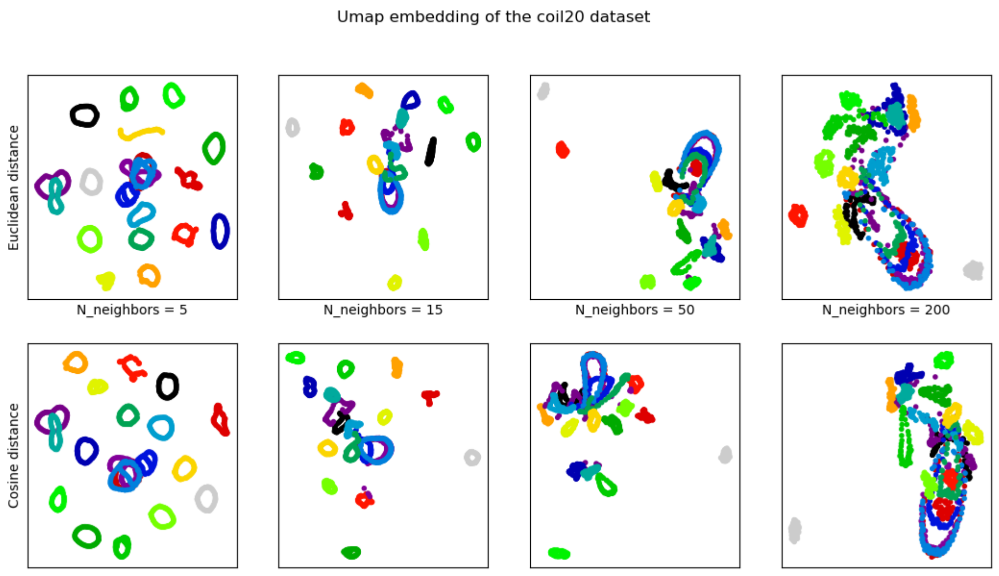
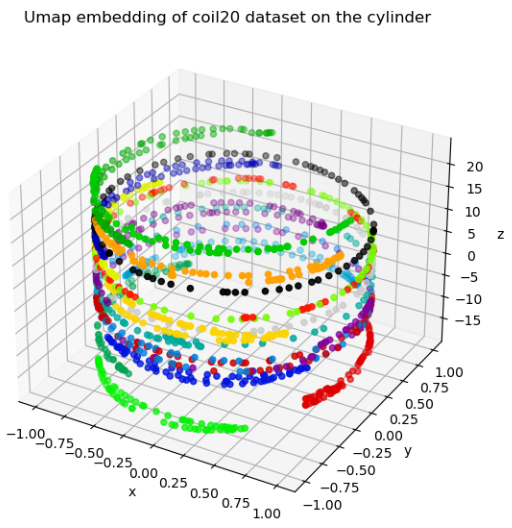


Figure 11: Final layout computed by UMAP for the COIL20 dataset, using two different metric for the ambient space and different numbers of nearest neighbors

Clearly even in this case a high number of nearest neighbors leads to poor performances. To employ large values of this parameter is equivalent to employ large radii for the covering of the manifold. It's likely that connections have been made between parts of the manifold that should be very far with respect to the geodesical distance, or even not connected. In the swiss roll case that would mean that "shortcircuits" occurred between different "layers" of the roll, in this data set could be that links between different objects have got comparable in stregth to the links between the different views of the same object. The results for 5 nearest neighbors are the best, even better than the spectral embeddings of Figure 10. The embeddings obtained with the Euclidean and the Cosine metrics are comparable. In fact, the most critical objects to differentiate are the same in the two cases, as the already mentioned purple-turquoise pair and the purple-turquoise-red-blue mess at the center of both the layouts. Nevertheless, the results are surprisingly coherent with the aim of the algorithm, designed to preserve the topological structure of the dataset.



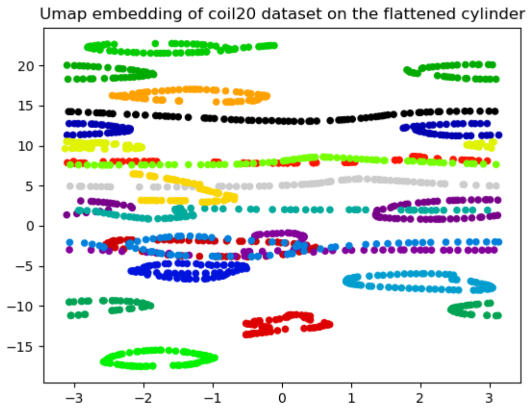


Figure 12: Top: embedding of the COIL20 dataset with UMAP onto a cylindrical manifold. Bottom: the same embedding, flattened onto a plane.

To conclude this section, the results are shown for the embedding of the data onto a manifold different than $\mathbb{R}^2$. In this case the metric of the high dimensional ambient space is euclidean, but the low dimensional manifold onto which the data have been embedded has been defined as a cylinder. This choice affects the final layout, since the cost function depends on the spatial configuration of the low dimensional points, i.e. on their mutual distances and therefore on the metric used to define them. In any case the embedding worked as expected: the circles of $\mathbb{R}^2$ are represented by straight lines on the cylinder and the objects are differentiated along the linear direction.

## 3 UMAP vs Laplacian Eigenmap

In this section we compare Laplacian Eigenmap and UMAP by means of the graphs by which they approximate the manifold. Mainly the strategy is to analyze the performance of their spectral embedding onto a plane. UMAP has been specifically designed for dimensionality reduction and visualization while Laplacian Eigenmap has been presented in the context of classification, but the visualization task has been chosen for both in this case because it's the method closer to the graphs we are concerned with. The classification would require a number of further steps.

**LAPLACIAN EIGENMAP**

1. **K nearest neighbors graph** (weighted by distances)
2. **Symmetrization (Euristic)**
$$B = \frac{A + A^T}{2}$$
3. **Directed heat kernel graph**
$$exp(-\frac{(d(x_i, x_{i_j})^2)}{t})$$
4. **Spectral embedding of the graph**
5. **Optimization of the layout** (not needed: automatic with the embedding)

**UMAP**

1. **K nearest neighbors graph** (weighted by distances)
2. **Directed fuzzy graph**
$$exp(-\frac{d(x_i, x_{i_j}) - \rho_i}{\sigma_i})$$
3. **Symmetrization (Fuzzy union)**
$$B = A + A^T - A \circ A^T$$
4. **Spectral embedding of the graph** (Inizialization of the layout)
5. **Optimization of the layout** (minimizaton of Cross Entropy)

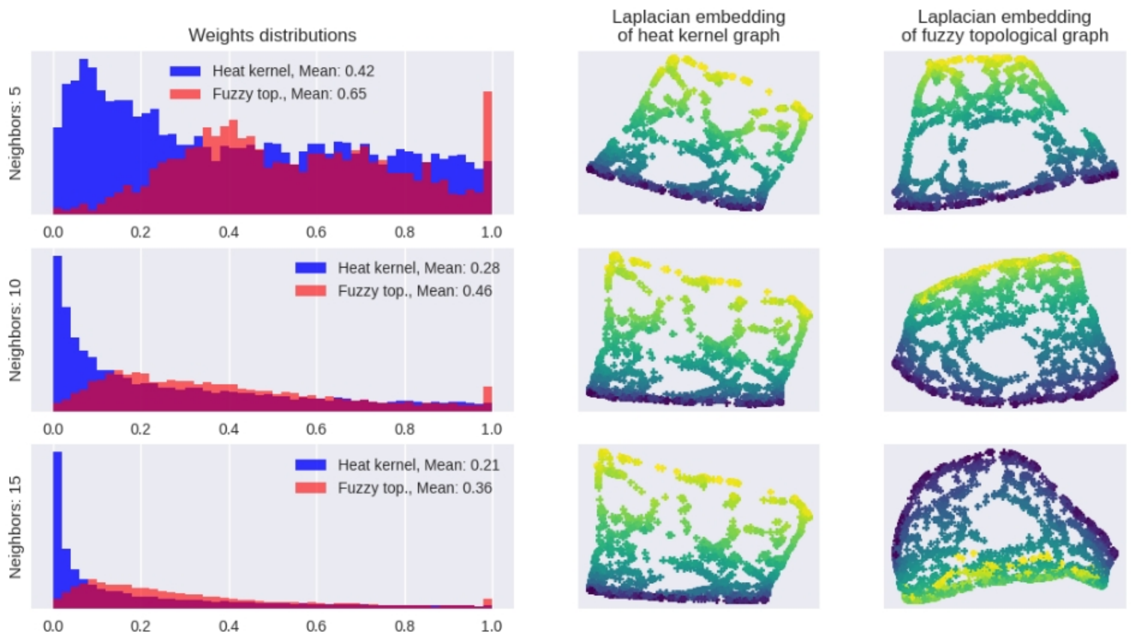


Figure 13: The Swissroll dataset embedded with Laplacian Eigenmap (middle column) and using the fuzzy graph representation of UMAP (right column). On the left column the weights distributions of the two graphs are shown and each row employs a different number of nearesr neighbors.

The Swissroll dataset has been represented by the heat kernel graph given by Laplacian Eigenmap and by the fuzzy topological graph built by UMAP. In Figure 13 their weights distributions are shown on the left column, their spectral embedding on the other columns. Three different number of nearest neighbors have been employed, increasing along the rows. For the heat kernel embedding has been chosen t=5. The weights distributions of the two graphs differs mainly in their extremal values: the heat kernel produces a lot of weights close to 0, resembling in fact an exponential distribution for more than 5 neighbors, while the fuzzy topological graph, due to the connectivity constraint, has always a unitary class of weights. The laplacian embedding of the heat kernel is shown in the middle column: the manifold could be accurately estimated. The laplacian embedding of the fuzzy topological graph works with no need to choose a time parameter and gives good results too. The two embeddings for N=5 are particularly similar.

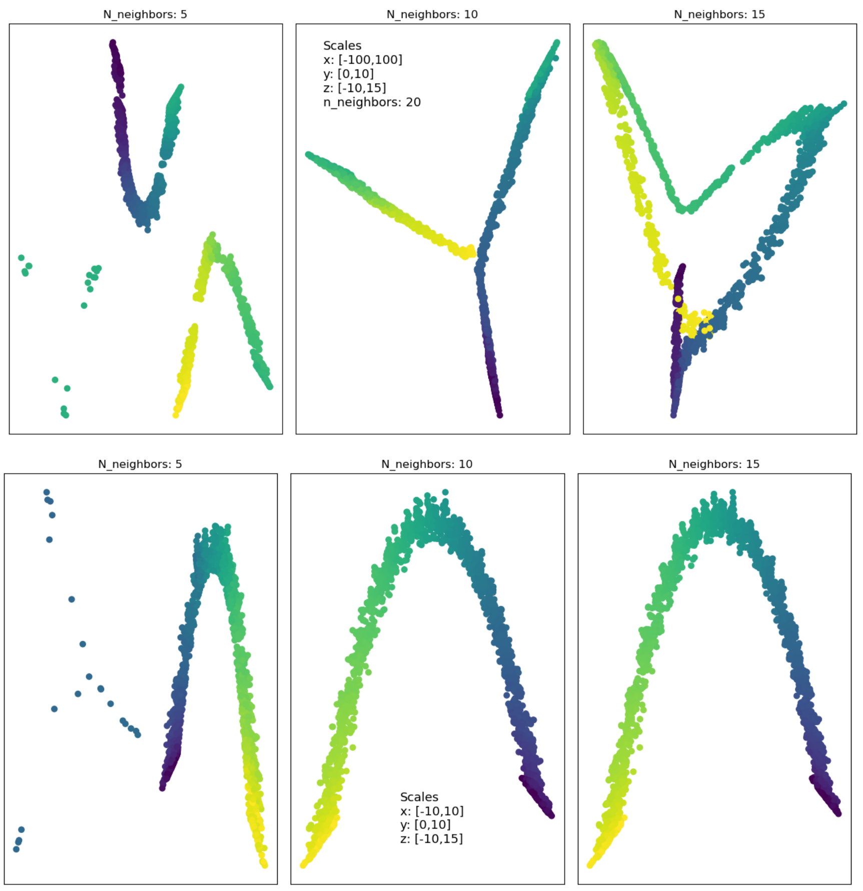


Figure 14: In the figure the fuzzy graph embedding of two different versions of the Swissroll dataset has been performed. In the top row the scale of the dataset is increased in the x direction, while in the bottom row the dataset is normalized.

The fuzzy graph embedding has been tested also for the differently scaled version of the Swissroll dataset, as Laplacian Eigenmap in Figures 4 and 5. The results are shown in Figure 14. In the top row the x axis were scaled as greater by an order of magnitude to the other axes. The spectral embedding of the fuzzy graph has been computed for three different numbers of nearest neighbors. For k=5 the resulting graph is not connected. In any of the cases a good representation could be obtained. Nevertheless, for the homogeneous case the results are valuable. The Laplacian Embedding of the same dataset is a line (see Figure 4). i.e. the algorithm totally neglects the second dimension of the manifold. In constrast, the fuzzy graph correctly represents the dataset as bidimensional, if a sufficient number of neighbors is employed (yet for k=5 the resulting embedding is not fully connected).

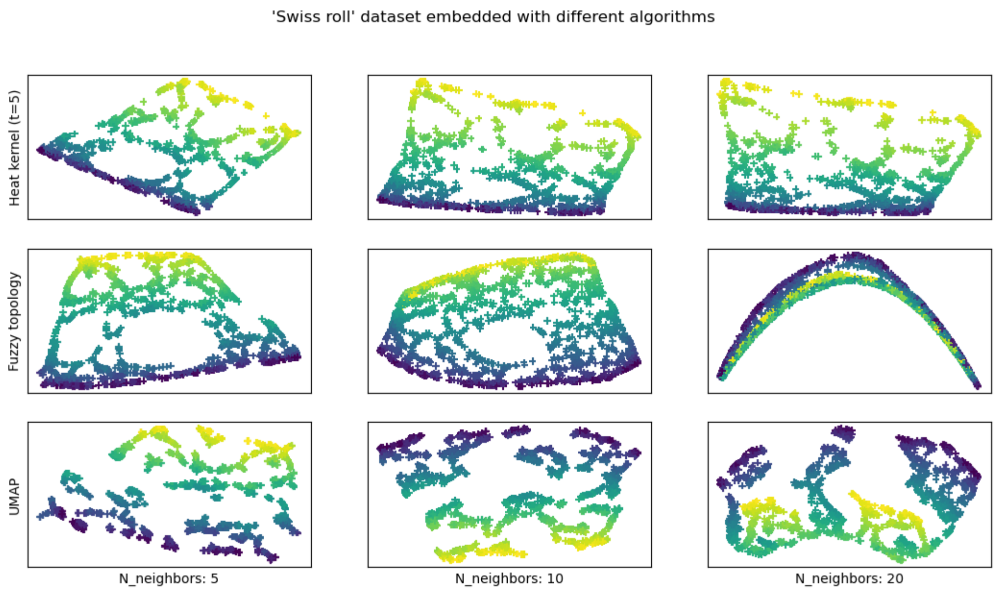


Figure 15: The Swissroll dataset is embedded in the figure with all the algorithm analyzed up to now: Laplacian Eigenmap (top row), UMAP without optimization (middle row) and the full UMAP algorithm (bottom row).

Finally in Figure 15 the results of Laplacian Eigenmap on the top row, of the embedding of the fuzzy graph in the middle row and of its optimization with respect to the Cross Entropy in the bottom row, are shown. In this case the optimization of the final layout performed by UMAP leads to worst results than the initial embedding, apart for the case in which 20 nearest neighbors were employed. It's likely that UMAP looks for structures that doesn't exist, in fact optimizing the embedding to the noise.
We recall that the actual noise for the dataset is 0, in the sense that the data lied exactly on the manifold. Nevertheless a small sampling on the manifold could be regarded as a sort of "topological" noise since, for example, holes or other kind of structures could be created.

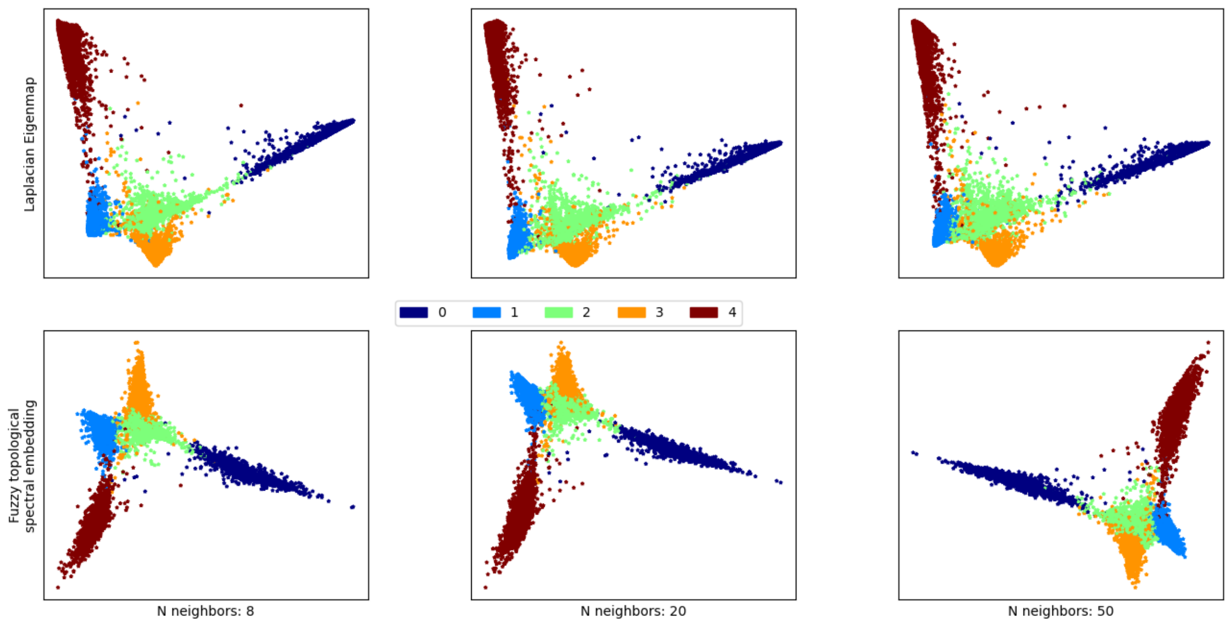


Figure 16: Spectral embedding of the neighbors graph (top) and of the fuzzy topological graph (bottom) realized from the MNIST dataset.

In Figure 16 the MNIST dataset has been employed in a reduced version. Only the data referring to the first 5 digits (half of the classes of the dataset) have been used, moreover only the 60% of the data points. That has been done in order to reduce the huge computational time required by Laplacian Eigenmap[1]. UMAP's fuzzy graph computation is almost immediate in comparison. The time parameter of Laplacian Eigenmap has been set to $\infty$, i.e. the graph used were a connectivity graph with edges weighted either 1 (the edges exists) or 0 (no edge at all). This is, in fact, the direct classical analogy to the fuzzy topological graph employed by UMAP. The results, apart for the computational time, are very similar. In fact the two approaches give the same embedding, that is really stable even with respect to the differences in the number of neighbors. In this case, UMAP gives the best results if it's allowed to perform the layout optimization as it's shown in Figure 9.

[1]The implementation of Laplacian Eigenmap of the python library scikit learn has been used.

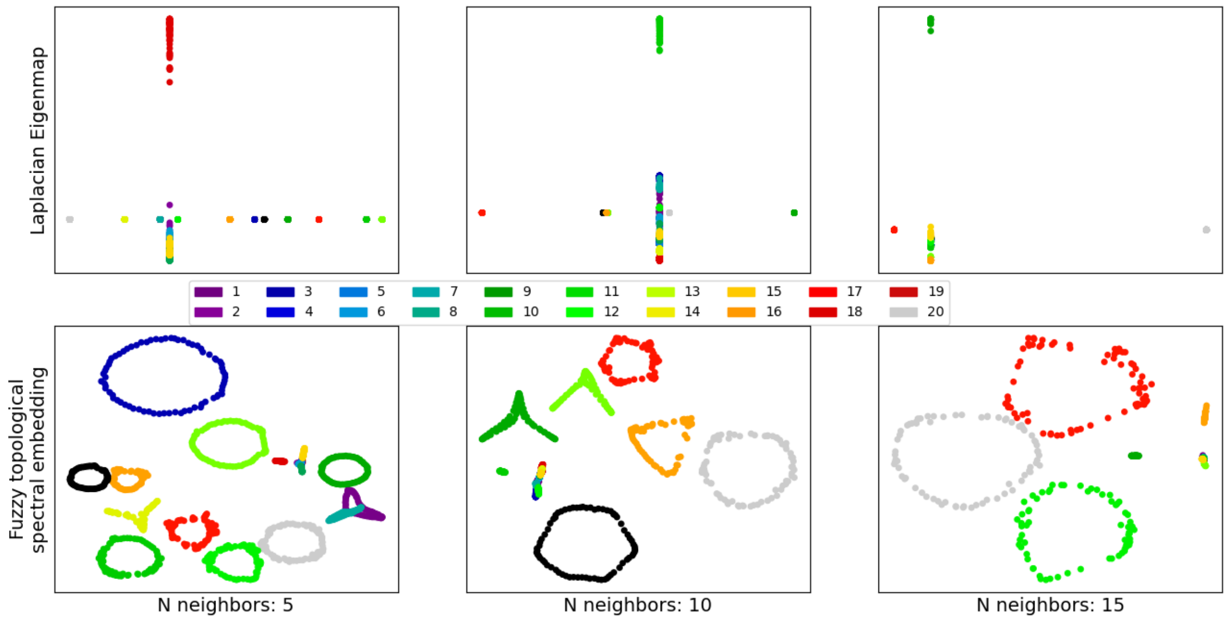


Figure 17

In Figure 17 the full COIL20 dataset has been employed, Laplacian Eigenmap could manage the analysis in a minute. Note that the MNIST dataset consists of 60000 points in a 784-dimensional space (each point being an image with 28x28 pixels), while the COIL20 of 1440 points in a 16384-dimensional space (128x128). At the end of the day, the features of the samples are used in order to compute the distances between the points. Much more computational weight is retained by the number of the points, i.e. by the size of the dataset, since it determines the size of the graph that will be used.
For the COIL20 datset, the differences are striking. Laplacian Eigenmap performes extremely poorly while the embedding of the fuzzy graph, especially for 5 nearest neighbors, is able to capture the structure of the dataset. With no optimization performed, already the clustering into separate objects has been made and the rotational topology of the clusters has been correctly represented.

# Conclusions

15 years divide Laplacian Embedding ([1],[2]) from UMAP ([3]) but their approach is essentially the same. In fact, UMAP contains and expands the framework developed with Laplacian Eigenmap and improves greatly the mathematical justification of some key passages common to many manifold learning techniques. Moreover, it's superior to all of them in the scaling with the size of the dataset, being de facto the current state-of-the-art in the field of manifold learning. Nevertheless, the final choice of the algorithm to be used is highly dependent on the dataset in hand and on the aim of the study.
The most impressive results of UMAP are maybe related to the COIL20 dataset, that has a clear intrinsic topology. Being UMAP designed with the claimed aim to preserve the topological, rather than the metrical, structure of the dataset, that could be expected. This isn't necessarily an advantage, since the main concern could be sometimes the metric relation between the data points. In such a case, UMAP could be not the best choice. The MNIST dataset has natural clusters, slighlty overlapping even for human classifications (it has surely happened to everyone to have misread a number), that UMAP could reconstruct only through its Cross Entropy based optimization. As for the fuzzy graph, it seems equivalent in such a case to the classical neighbors graph used by Laplacian Eigenmap. The Swissroll dataset could be easily unrolled by both the algorithms, even if the optimization step in UMAP led to overfitting. The heat kernel approach integrated with Laplacian Eigenmap results a bit unhandy, due to the sensitivity of the results on the choice of the time parameter. Moreover, it depends highly on the characteristic distances distribution of the dataset and an automatic pipeline to determine it is not present, as far as I know. In contrast, UMAP is straightforward to apply. In any case, let's mention that in principle a number of different kernels can be employed in combination with Laplacian Eigenmap and that the heat kernel as a notion still retains a huge potential. It's strength resides in its strict connection with the Heat diffusion, that gives an immediate and powerful interpretation to the process. An example of a different implementation of the Heat kernel in tasks similar to these is given by some papers of Hancock [4].
That's a big difference with respect to UMAP, whose final layout lacks completely of interpretability. The transformation operated by UMAP at the end of its optimization is not anlytical and its axes don't have a specific meaning, unlike for example PCA. Another potential limitation of UMAP is its assumption that exists manifold structure in the data. This could lead it to find manifold structures within the noise of a dataset, increasing the risk of overfitting. The second main assumption of both UMAP and Laplacian Eigenmap is that the data are uniformly distributed on the manifold. The theory behind UMAP starts from this assumption and continues through the definition of local metric spaces and local weighting functions. If there are reasons to believe that the non homogeneity of the dataset is informative, UMAP could be not the best choice. Anyway, note that the current version of UMAP also provides support for the so called densMAP. The

densMAP algorithm augments UMAP to preserve local density information in addition to the topological structure of the data, as detailed in [5]. Finally, UMAP concerns itself primarily with representing the local structure (even if the global structure arises nicely in many cases). If the global structure is of main interest, approaches as MDS could be a better choice. Furthermore, MDS tries to preserve the metric relations between the points, with secondary care with respect to the topology of the dataset. Recall though that for larger dataset sizes MDS is going to quickly become completely unmanageable.

**Code employed**

All the code used to make the figures of the presentation is available at:
`https://github.com/FMagnani/ManifoldLearning_scripts`

## References

[1] Belkin, Mikhail, and Partha Niyogi. "Laplacian eigenmaps for dimensionality reduction and data representation." Neural computation 15.6 (2003): 1373-1396.

[2] Belkin, Mikhail, and Partha Niyogi. "Semi-supervised learning on Riemannian manifolds." Machine learning 56.1 (2004): 209-239.

[3] McInnes, Leland, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction." arXiv preprint arXiv:1802.03426 (2018).

[4] ElGhawalby, Hewayda, and Edwin R. Hancock. "Heat kernel embeddings, differential geometry and graph structure." Axioms 4.3 (2015): 275-293.

[5] Narayan, A, Berger, B, Cho, H, Density-Preserving Data Visualization Unveils Dynamic Patterns of Single-Cell Transcriptomic Variability, bioRxiv, 2020

Oskolkov, Nikolay. "How Exactly UMAP Works, and why exactly it is better than tSNE". Towards Data Science (2019).
`https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668`

UMAP documentation
`https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`

UMAP code
`https://github.com/lmcinnes/umap`