Autoencoder with Locally Linear Embedding

Dimensionality Reduction Project

Mathieu Pont and Lucas Rodrigues Pereira

January 2020

Abstract

1 Introduction

In the context of the Master's Degree on Machine Learning for Data Science, at the University of Paris (Descartes), we have been given the task to discuss some dimensionality reduction methods, as part of the "Dimensionality Reduction" course.

At first, we used the synthetic FCPS data sets to run different dimensionality reduction (DR) algorithms such as PCA, MDS, LLE, Laplacian Eigenmap, Isomap and Autoencoder. Methods were used here to reduce the data sets to only two dimensions.

Finally we have tested a model combining Autoencoder and LLE in a unique algorithm.

1.1 Brief methods description

Before experimenting, we will briefly discuss each method particularity in order to guide the following analysis.

PCA. The Principal Component Analysis searches new axis that maximizes variance represented in each axis (principal component). The experiments have been conducted without using a Kernel for PCA.

Autoencoder. Autoencoders are deep neural networks with a middle layer with reduced number of neurons. The main goal is the reconstruction of the original data "forcing" the compression of the information propagated by reducing the number of neurons (encoding) and then re-augmenting it (decoding) to the original dimension. The structure of the network may considerably influence the result. The plotted result can be the encoded data in two dimensions.

Manifold learning. The following methods are part of the manifold learning technique. These methods are not based on the projection of the dataset, but they aim to representing the data with the desired number of dimensions while trying to keep as much as possible the relation between data points and its neighbors (distance).

MDS. MDS is based on dissimilarity among data points. It can be metric (respecting the triangular inequality) or non metric. It tries to keep the similarity between each data point and all others, which means it takes into consideration both local and global distances.

Isomap. This method searches for an embedding that maintains geodesic distances between all points as intact as possible.

LLE. This method is based on the linear combination of the k-nearest neighbors that generates each produces data point.

Laplacian Eigenmap. This embedding method uses spectral decomposition of the graph Laplacian to preserve local distances while lowering dimensionality.

2 Dimensionality Reduction on synthetic FCPS data sets

The synthetic FCPS dat asets are very interesting for machine learning algorithms experiments because each of them has a peculiarity[1].

Dataset	#samples	#features	#classes	Peculiarity
Atom	800	3	2	Different variances and linearly not separable
Chainlink	1000	3	2	Linearly not separable
EngyTime	4096	2	2	Gaussian mixture
Hepta	212	3	7	Clearly defined clusters, different variances
Lsun	400	2	3	Different variances and inter cluster distances
Target	770	2	6	Outliers
Tetra	400	3	4	Almost touching clusters
TwoDiamonds	800	2	2	Cluster borders defined by density
WingNut	1016	2	2	Density vs. distance

Table 1: Description of FCPS data sets.

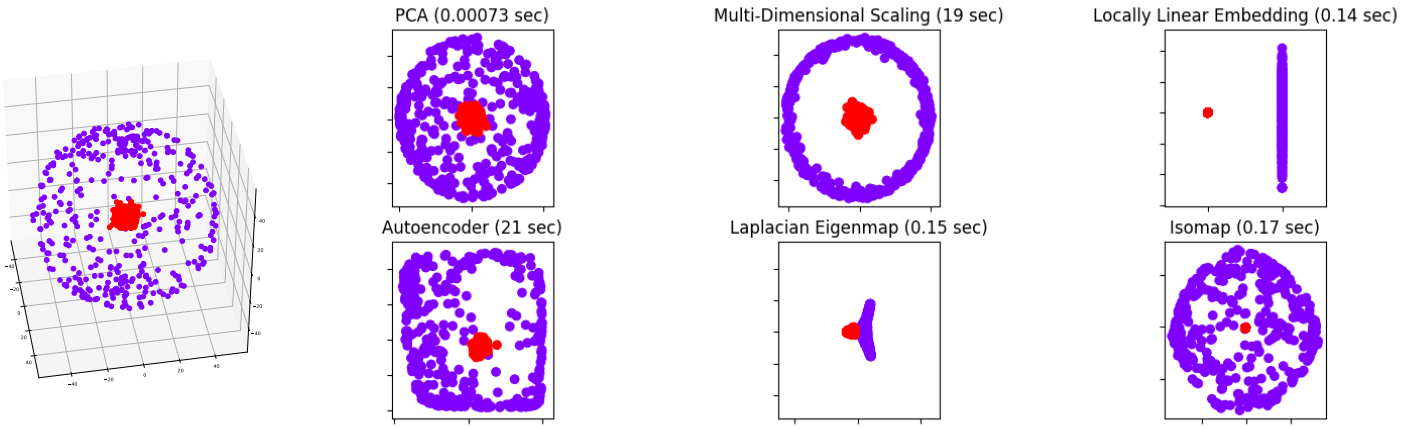


Figure 1: Atom data set.

As seen from Table 1 Atom has the peculiarity of being not linearly separable, therefore, PCA (a linear algorithm) can't find a correct plane where the classes are well separated. Moreover, due to the structure of the data, a plane maximizing the variance will not be able to separate both classes. Since we can see the autoencoder as a generalization of PCA it is not surprising to see kind of the same result even if the autoencoder performs better because it captures non-linearity.

MDS and Isomap try to preserve both local and global distances, which is why we can observe a separation between the two classes (core and surface of the atom). However, in this case, local distances are more important for separating classes, thus, LLE and Laplacian Eigenmaps seemed to have worked well.

[1]https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/data

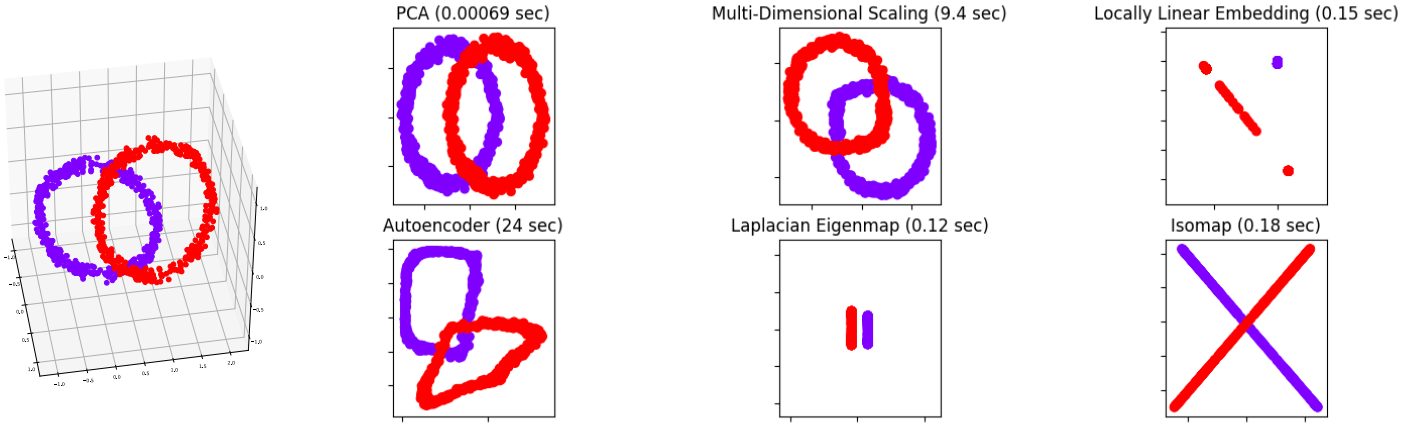


Figure 2: Chainlink data set.

As regards the Chainlink dataset, we have obtained similar results as compared to the Atom dataset, except the difference between MDS and Isomap. As the Isomap seeks to preserve the geodesic distance among data points, it does not attempt to keep the shape of the data, whereas the metric MLDS does.

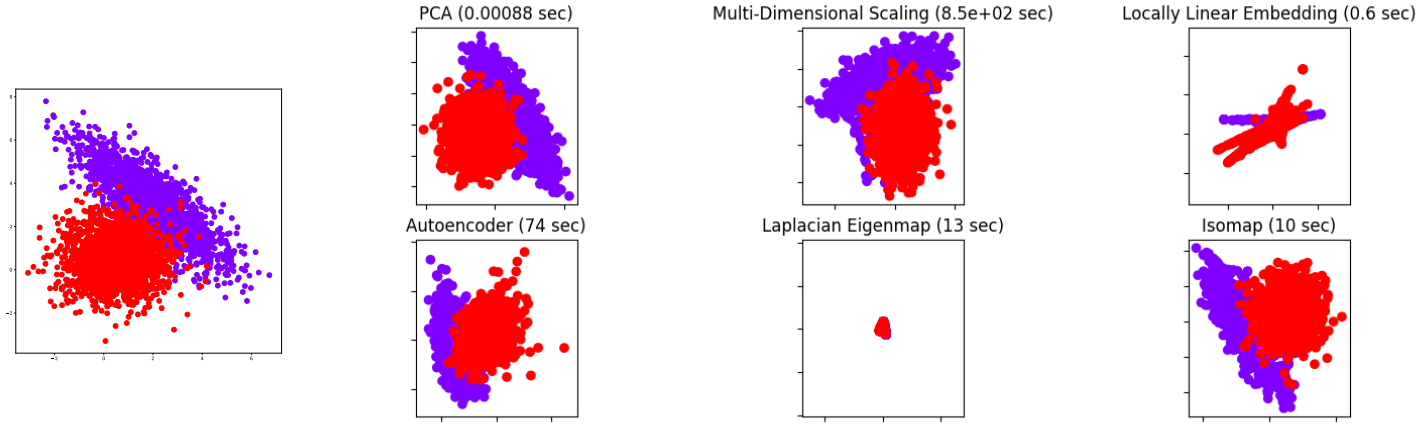


Figure 3: EngyTime data set.

Here, the EngyTime dataset presents two partially superposed clouds of points. This is where LLE and Spectral Learning struggle. All points are closely connected and local distances should lead to bad results. The other methods perform relatively well in this scenario by keeping the original shape of the data. Except for the Laplacian Eigenmap which concentrates all points in a, kind of, single point.

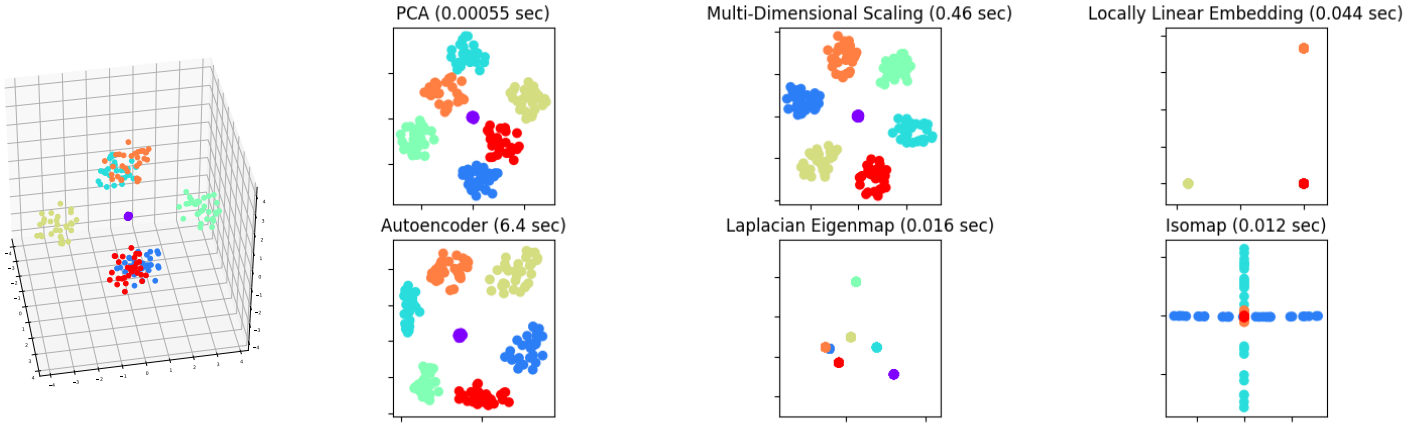


Figure 4: Hepta data set.

The Hepta dataset has 7 well-separated classes. Once more, the PCA, MDS and autoencoder methods have performed well. However, LLE, the Laplacian Eigenmap and Isomap have not yielded useful results. Having multiple very separated classes with aggregated data points in each class caused the LLE to "merge" them into almost a single point, even "merging" classes while trying to represent them in two dimensions. The same analysis is valid for the Laplacian Eigenmap even if it manages to better separate classes. Regarding the Isomap approach, as it tries to preserve geodesic distances (the shortest path) the well-separated classes around the strongly dense purple class distort the 2D projection, making it impossible to analyze and distinguish clusters.

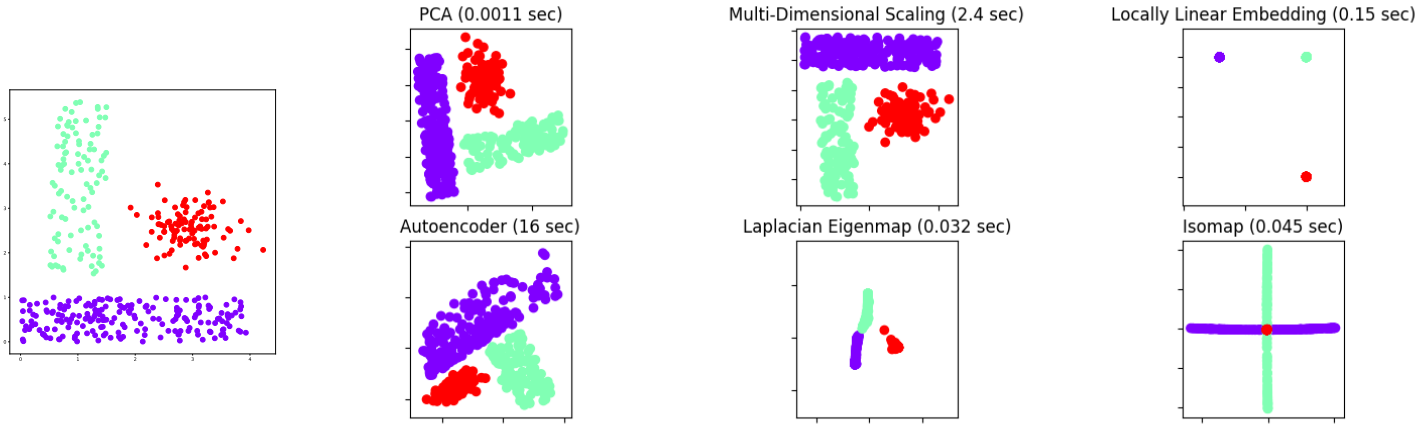


Figure 5: Lsun data set.

The same analysis fits the Lsun dataset. The difference here is that we are not reducing the number of dimensions.

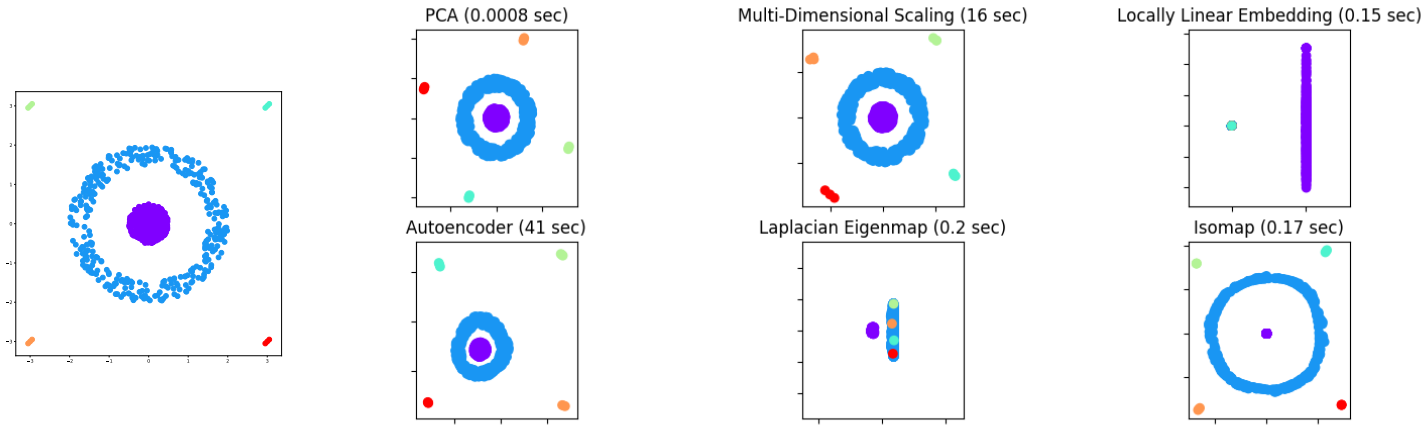


Figure 6: Target data set.

The Target dataset adds complexity for clustering beacuse of its outliers. As can be seen from the plots above, all six methods have presented interesting projections. Isomap has concentrated local neighborhoods. Except LLE and Laplacian Eigenmaps, all have taken outliers into consideration. On the other hand, these two exceptions have grouped the outliers along with the external ring of the target.

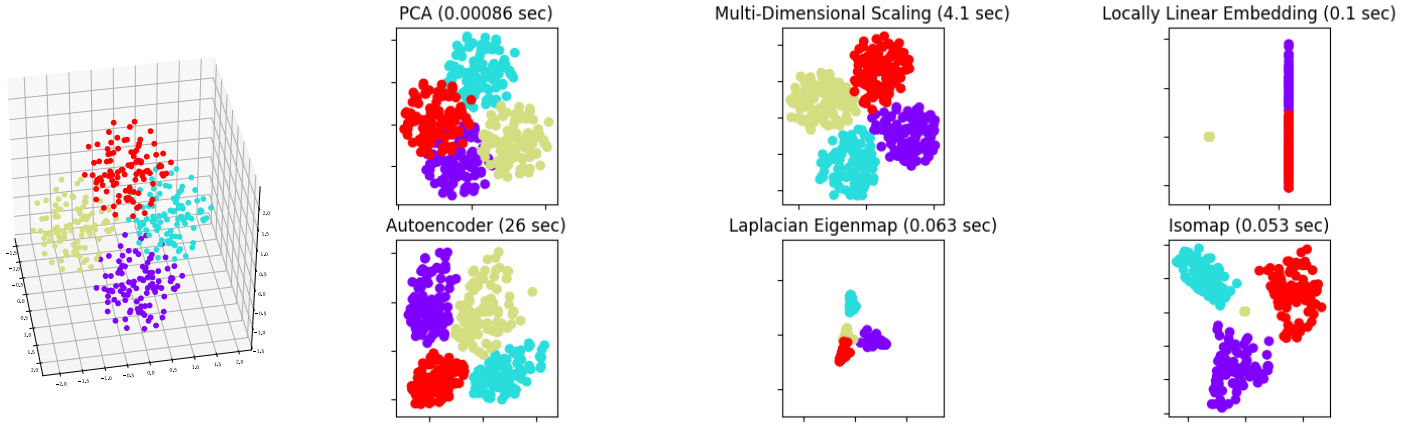


Figure 7: Tetra data set.

The LLE separated the less superposed datapoints (the green cluster), however flattened all other, making it hard to analyse the plot. All other methods have delivered good two-dimensional data visualization. So far, across all datasets, it is possible to affirm that PCA, MDS and the Autoencoder have been the most stable and with best performance.

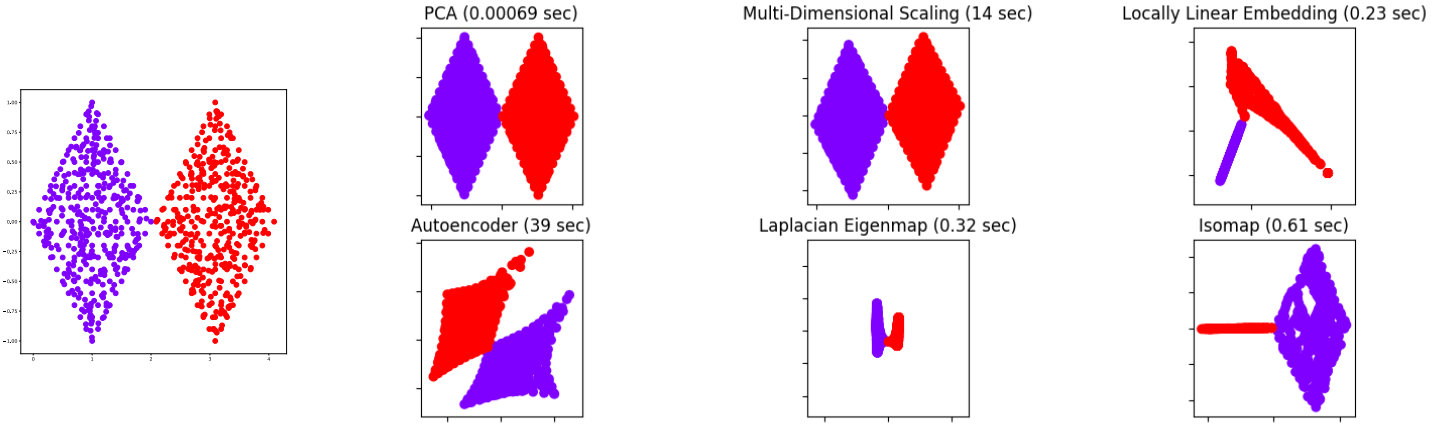


Figure 8: TwoDiamonds data set.

The TwoDiamonds dataset has two clusters touching each other and homogeneous density. Even though the methods that take into consideration global distances have clearly performed better than the other ones (LLE and Laplacian Eigenmaps), all have kept the clusters separated.

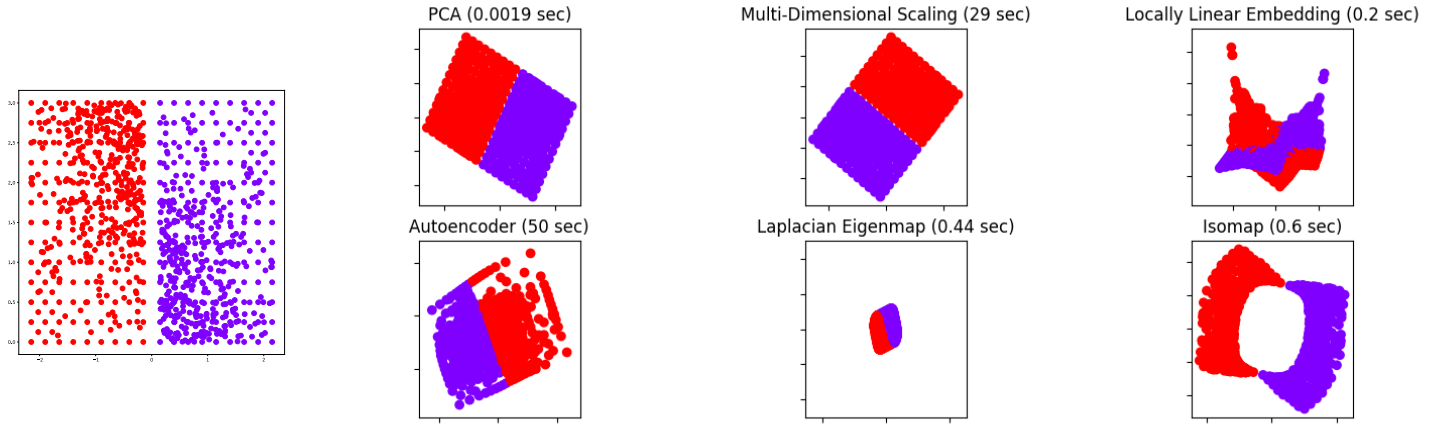


Figure 9: WingNut data set.

The same thing did not occur when the clusters have different densities. In this case, local versus global distances plays greater role in the clustering. LLE, as it depends on local neighborhoods, the separation between the two clusters become indistinguishable where there is low density in one cluster and high in the other. Therefore, the LLE did not project the data well. The Laplacian Eigenmap on the other hand has managed to capture the overall appearance of the dataset.

3 Combining Autoencoder and LLE

3.1 Methodology

Here, we describe how we can combine Autoencoder [Hinton and Salakhutdinov, 2006] and Locally Linear Embedding (LLE) [Roweis and Saul, 2000] in a single algorithm.

The objective function of the Autoencoder can be defined as below:

$$\min_{\theta_1,\theta_2} \|\mathbf{X} - g_{\theta_2}(f_{\theta_1}(\mathbf{X}))\|^2 \tag{1}$$

It has the aim to minimize the reconstruction error, that is, the difference between the original input $\mathbf{X}$ and the reconstructed input $g_{\theta_2}(f_{\theta_1}(\mathbf{X}))$ where g is the function achieved by the decoder according to its parameters θ_2 and f the function performed by the encoder with its parameters θ_1.

LLE can be decomposed in three steps. At first we compute the k nearest neighbors of each data point. Then we try to reconstruct each data point according to its neighbors (as barycentric coordinates). Finally we minimize the projection of each data point in the new vector space with its barycentric coordinates of its neighbors in the new vector space.

The objective function of LLE for the weight matrix can be defined as follows:

$$\min_{\mathbf{W}} \|\mathbf{X} - \mathbf{W}\mathbf{X}\|^2 \tag{2}$$

Then, to project data points in the new space we minimize the following function:

$$\min_{\mathbf{Y}} \|\mathbf{Y} - \mathbf{W}\mathbf{Y}\|^2 \tag{3}$$

Where $\mathbf{Y}$ is therefore the projected dataset with fewer dimensions than $\mathbf{X}$.

The method combine the Autoencoder objective function (Eq. 1) and the one of the Locally Linear Embedding algorithm (Eq. 2):

$$\min_{\theta_1,\theta_2,\mathbf{W}} \|\mathbf{X} - g_{\theta_2}(f_{\theta_1}(\mathbf{X}))\|^2 + \lambda\|f_{\theta_1}(\mathbf{X}) - \mathbf{W}f_{\theta_1}(\mathbf{X})\|^2 \tag{4}$$

The first term of Eq. 4 corresponds to the Autoencoder loss whereas the second term corresponds to the LLE loss. Here, the LLE loss is based on the encoded data (by the encoder of the autoencoder).

The first term can therefore be minimized by a classical training on the autoencoder, as well for the second term for the LLE algorithm. Moreover, since the second term depends on the parameters of the encoder we can also minimize it by training the encoder. The algorithm to minimize Eq. 4 is described here after:

Algorithm 1 Calculate θ_1, θ_2 and $\mathbf{W}$

Require: data matrix $\mathbf{X}$
  repeat
    (1) - Update $\mathbf{W}$ with LLE
    (2) - Update θ_1 and θ_2 with Autoencoder training
  until convergence

The step 2 of Algorithm 1 minimizes both terms of Eq. 4 by training the autoencoder for the first term and the encoder for the second one. The addition of the second term to the classical autoencoder loss can be done easily with *Tensorflow* and its automatic differentiation module (for example, with *GradientTape*). We have decided to do the LLE update at first because the loss of the autoencoder needs to have the $\mathbf{W}$ matrix.

3.2 Experiments and Results

3.2.1 FCPS Data Sets

After having implemented the AE-LLE algorithm we have used the FCPS data sets to evaluate the performance of the method in term of clustering in the reduced latent space. Gaussian Mixture Model (GMM) was the model selected for the clustering part. Indeed, we saw that generally, for our case, it has better performances than k-Means (which makes sense since GMM can be seen as a generalization of it). Moreover, since we work with small data sets the execution time was reasonnable.

Table 2 shows the different results. We used five representations for each dataset, the original dataset, the one encoded by the AE, the embedding of LLE, then the two representations provided by AE-LLE (with the encoder and with LLE). Each experiment was run 10 times and we have computed the mean and standard deviation for each. The autoencoder was run for 500 epochs to ensure convergence, when it was not the case the results were discarded. Its hyper-parameters are available in Appendix A and visualizations of the latent space in Appendix B.

For each row the higher value has been put in bold, here we have decided to compare only DR methods between them. We nevertheless show the result on original data as a comparison.

Dataset	Metrics	AE	LLE	AE-LLE ($f_{\theta_1}(\mathbf{X})$)	AE-LLE (Y of LLE)	Original
Tetra	NMI	**0.94 ± 0.03**	0.72 ± 0.02	0.87 ± 0.07	0.84 ± 0.08	1.0
	ARI	**0.95 ± 0.03**	0.69 ± 4e-3	0.85 ± 0.10	0.79 ± 0.13	1.0
Lsun	NMI	0.77 ± 0.12	**1.0**	0.94 ± 0.06	0.93 ± 0.13	1.0
	ARI	0.73 ± 0.17	**1.0**	0.95 ± 0.05	0.92 ± 0.17	1.0
Atom	NMI	0.38 ± 0.27	**1.0**	0.94 ± 0.04	0.46 ± 0.32	0.14
	ARI	0.30 ± 0.33	**1.0**	0.97 ± 0.03	0.41 ± 0.39	0.03
TwoDiamonds	NMI	**0.98 ± 0.02**	0.97	**0.98 ± 0.02**	0.93 ± 0.06	1.0
	ARI	**0.99 ± 0.01**	**0.99**	**0.99 ± 0.01**	0.97 ± 0.03	1.0
Hepta	NMI	**1.0**	0.64	**1.0**	0.67 ± 0.10	1.0
	ARI	**1.0**	0.23	**1.0**	0.31 ± 0.14	1.0
Target	NMI	0.72 ± 0.02	0.65	**0.73 ± 0.03**	0.69 ± 0.03	0.70
	ARI	0.67 ± 0.02	0.57	**0.68 ± 0.02**	0.66 ± 0.05	0.66
EngyTime	NMI	**0.77 ± 0.01**	0.11 ± 0.09	**0.77 ± 0.01**	0.44 ± 0.22	0.79
	ARI	**0.85 ± 0.01**	0.08 ± 0.10	**0.85 ± 0.01**	0.46 ± 0.29	0.87
Chainlink	NMI	0.41 ± 0.10	0.42 ± 0.22	**0.50 ± 0.22**	0.20 ± 0.13	0.84
	ARI	0.34 ± 0.14	0.35 ± 0.27	**0.49 ± 0.25**	0.10 ± 0.15	0.91
WingNut	NMI	0.54 ± 0.05	0.13	0.50 ± 0.04	**0.65 ± 0.16**	0.78
	ARI	0.65 ± 0.05	0.17	0.61 ± 0.04	**0.69 ± 0.21**	0.86

Table 2: Metrics of Gaussian Mixture Model on different representations for each FCPS data set.

The choice of λ can be very important for the convergence and the quality of results, this choice will be more discussed in 3.3. Here, we have chosen the value for each data set for which we have the best results overall.

For Tetra AE-LLE seems to have done kind of an average between AE and LLE making it less efficient overall even if its $\mathbf{Y}$ representation seems to be better for the clustering that the one of the original LLE.

For the Atom and Lsun data sets, while LLE manages to have perfect NMI and ARI, both representations of AE-LLE did not perform as well. However, we must still notice that the metrics values for the encoded data of AE-LLE are very much higher that those for the AE. Whereas, the metrics of $\mathbf{Y}$ of AE-LLE are lower than those of the original LLE, even much more lower for Atom. AE-LLE

seems to have given in some way more expression power to the encoded data at the expense of the $\mathbf{Y}$ representation. It did, like for Tetra, kind of an average between AE and LLE results.

On original data, GMM did not performs very well with Atom since its decision boundary is highly non-linear and classes can't be represented as two Gaussian mixtures correctly. Whereas, due to the structure of Lsun, GMM has perfect metrics on this data set.

For TwoDiamonds, Hepta, Target and EngyTime the results are more or less the same when we compare clustering on encoded data of AE and of AE-LLE. For the clustering on $\mathbf{Y}$ of LLE and of AE-LLE the latter seems slighlty better except for TwoDiamonds, moreover the results of $\mathbf{Y}$ of AE-LLE are much more higher for EngyTime than those of original LLE. For these data sets (except TwoDiamonds) AE-LLE seems to have

For Chainlink and WingNut AE-LLE seems to be better, on the encoded data for Chainlink and on $\mathbf{Y}$ for WingNut.

Overall, AE-LLE performed very well and did the same or better than the best of original methods for 6 of 9 data sets. Even when it did not beats them it manages to increase the metrics values for one of the two representations compared to the original. Here we can observe 3 behaviours, the first where AE-LLE did kind of an average between AE and LLE. The second where it manages to did like the best of AE and LLE and improve the results of the worst. Finally, the third where it outperforms both AE and LLE.

3.2.2 Images Data Sets

Here we evaluate the AE-LLE algorithm on the following images data sets.

Dataset	**#samples**	**#features**	**Image dim.**	**#classes**
USPS	9298	256	16 x 16	10
MNIST	70,000	784	28 x 28	10
ORL	400	10,304	112 x 92	40
Coil20	1440	16,384	128 x 128	20
Coil100	7200	16,384	128 x 128	100
Yale	165	45,045	231 x 195	15
PIE	41,368	311,040	640 x 486	68

Table 3: Description of images data sets.

As done in Table 2 we made a clustering in the reduced latent space of different DR methods. Since we work with relatively large data sets we have decided to use k-Means instead of GMM for the clustering part in order to not have a too high execution time.

The experimental protocol was quite the same than with FCPS data sets, except that we have run autoencoder for 250 epochs instead of 500. Since we work with images data sets we have decided to use a convolutional neural network whose hyper-parameters are available in Appendix C.

Moreover, due to the dimensions of the train set of MNIST (60000×784) we have decided to use the test set that is much smaller. Same for Coil100 we took a subset of the full data set (it will be more discussed later).

Dataset	Metrics	AE	LLE	AE-LLE ($f_{\theta_1}(\mathbf{X})$)	AE-LLE (Y of LLE)	Isomap
USPS	NMI	0.61 ± 0.01	0.62 ± 0.05	**0.66 ± 0.02**	0.59 ± 0.04	**0.81**
	ARI	0.52 ± 0.01	0.29 ± 0.08	**0.57 ± 0.02**	0.28 ± 0.05	**0.74**
MNIST	NMI	0.46 ± 0.02	0.56 ± 0.02	0.56 ± 0.02	**0.61 ± 0.03**	**0.72**
	ARI	0.35 ± 0.04	0.24 ± 0.05	**0.41 ± 0.02**	0.35 ± 0.06	**0.64**
ORL	NMI	0.89 ± 0.01	0.52 ± 0.02	**0.91 ± 0.01**	0.57 ± 0.04	0.87 ± 0.01
	ARI	0.68 ± 0.02	0.02 ± 0.01	**0.73 ± 0.02**	0.04 ± 0.01	0.62 ± 0.03
Coil20	NMI	**0.79 ± 0.02**	0.33 ± 0.02	0.78 ± 0.01	0.33 ± 0.05	0.69 ± 0.01
	ARI	0.60 ± 0.04	0.02 ± 0.01	**0.61 ± 0.02**	0.02 ± 0.01	0.26 ± 0.01
Coil100	NMI	**0.73 ± 0.01**	0.56 ± 0.01	**0.73 ± 0.01**	0.57 ± 0.04	0.70 ± 3e-3
	ARI	**0.33 ± 0.01**	0.03 ± 0.01	**0.33 ± 0.01**	0.03 ± 0.02	0.25 ± 0.01

Table 4: Metrics of k-Means on different representations for each data set. (Best results among AE, LLE, and both AE-LLE representations are bolded. Moreover, results of Isomap are bolded too when they are better.)

For the USPS, MNIST and ORL data sets AE-LLE performed better than both of its original equivalents. However, Isomap did better except for ORL. We must notice that, due to the size of this data set, we were able to choose a much bigger architecture than for MNIST or USPS, therefore, perhaps a bigger architecture for these latter will allow AE-LLE to do better than Isomap.

The results did not change much for Coil20 and Coil100, perhaps that a bigger neural network architecture will gives more expression power to the autoencoder giving it more ability to correctly learn from the two terms of the loss. We were forced to take a small architecture due to the size of the dataset (see 3.3).

For all these images data sets it is interesting to note that Isomap performed better than LLE, meaning that, here, geodesic distances are a better property to take into account that reconstruction from neighbors as barycentric coordinates.

3.3 Discussion

Execution time. The execution time until convergence is much longer with AE-LLE than a classical autoencoder with the same architecture (almost 10 times longer). The computation of LLE at step 1 of Algorithm 1 is principally what makes this duration. It's why we have decided to do this step not at every iteration but once in ae_step iterations, the latter being a hyper-parameter to tune, we have found that a value of $ae\_step = 5$ is a good trade-off. We noticed that it speeds up the convergence and make it more stable since the $\mathbf{W}$ matrix does not change at every iteration.

Choice of λ. The choice of λ in Eq. 4 is crucial for a good and quick convergence. We found that the LLE weight reconstruction loss (of Eq. 2) can be a good indicator to choose λ in order to speed up training.

Most of the time the loss begins very low because at the beginning of the autoencoder training encoded data are kind of noise and are overall the same. Therefore, the reconstruction with neighbors (as barycentric coordinates) of LLE is easy, making the loss low. However, the value of convergence is generally higher since at the end of training encoded data are more "structured" and harder to represent as barycentric coordinates than noise from the beginning of training. Still, this loss value at the convergence with AE-LLE is nevertheless lower than the loss value of the original LLE.

The typical behaviour for a quick convergence is for the LLE loss to slowly increase to the value of convergence. However, the choice of λ will change a lot this behaviour if not set correctly.

If the loss rapidly increases (above the value of convergence) to finally decreases to the value we can get a higher λ, it can avoid the increasing of the LLE loss above the value of convergence by giving it more impact. If it decreases for a time to finally increases to the value of convergence we can get a lower λ. We found that this method tends to give the best results in term of clustering in the reduced latent space, however it is not a universal rule.

However, we can still choose the λ to give to the model more of the LLE or of the AE. Indeed, we saw, particularly with FCPS data sets, that for some of them increasing or decreasing λ tends to make the results more likely to those of the LLE or the AE respectively.

Here after, we plot the metrics values for some data sets with different λ values. The both straight horizontal lines are the mean of results of original LLE and AE, in dotted line and dashed (long dot) line respectively.

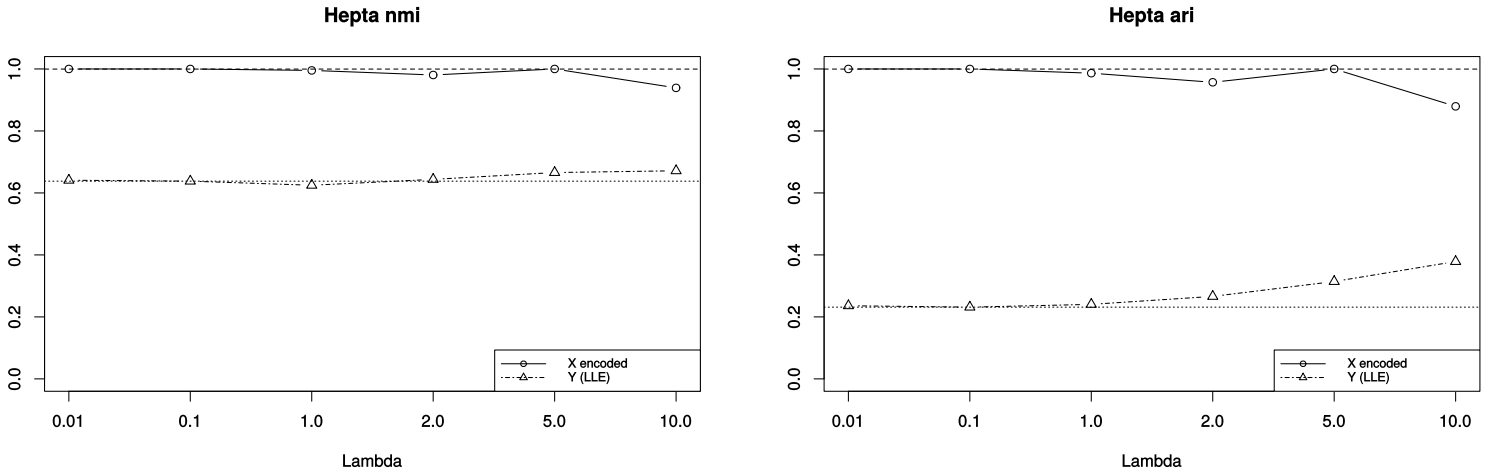


Figure 10: Metrics for **Hepta** data set and different λ.

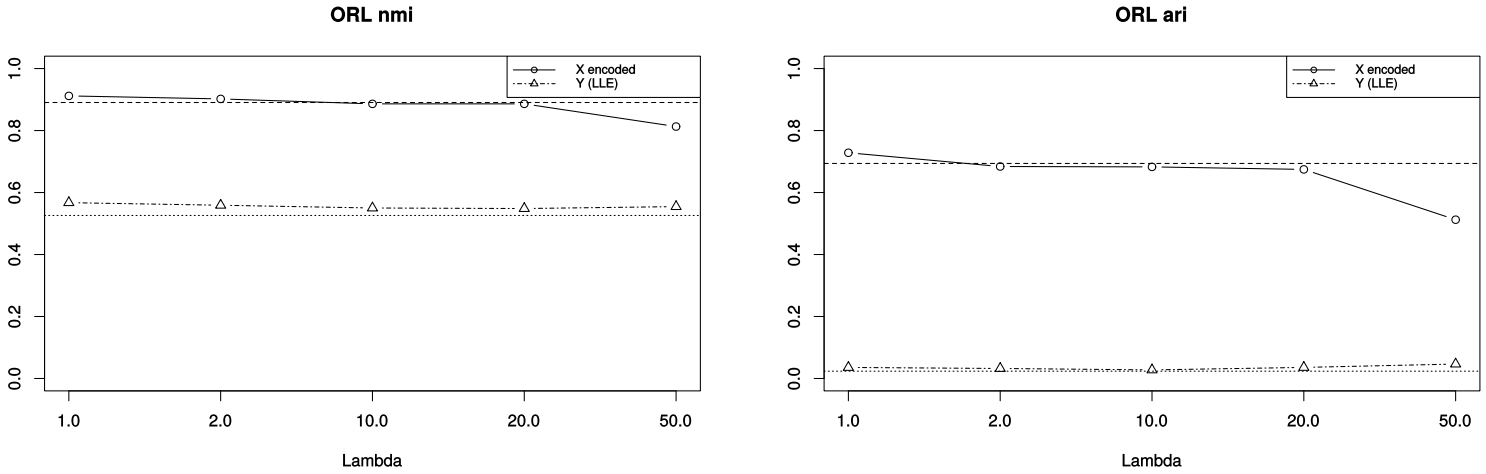
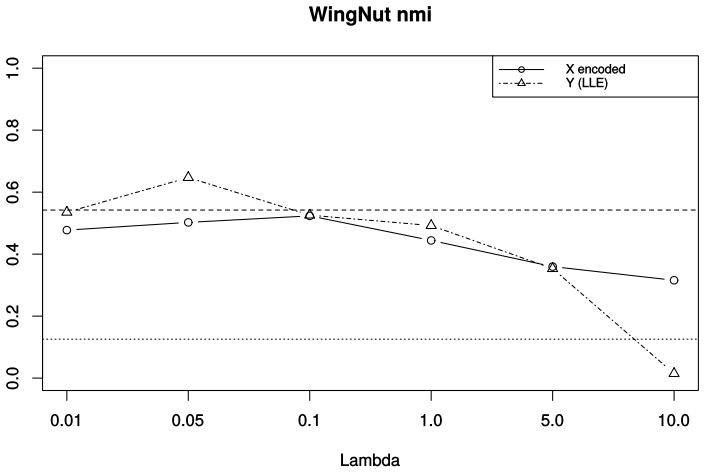


Figure 11: Metrics for **ORL** data set and different λ.



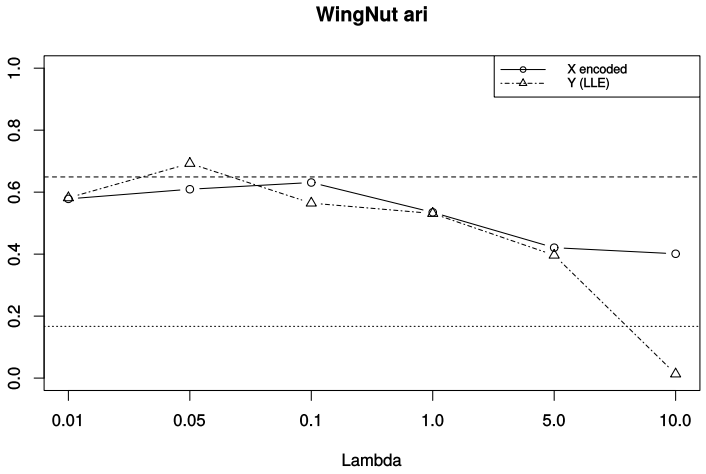
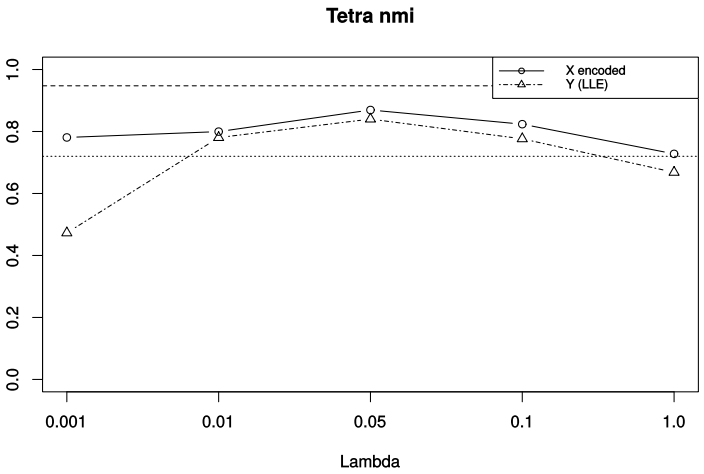


Figure 12: Metrics for **WingNut** data set and different λ.



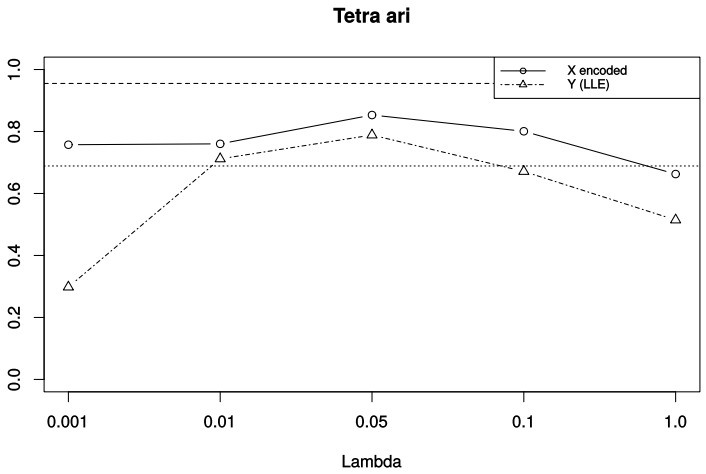
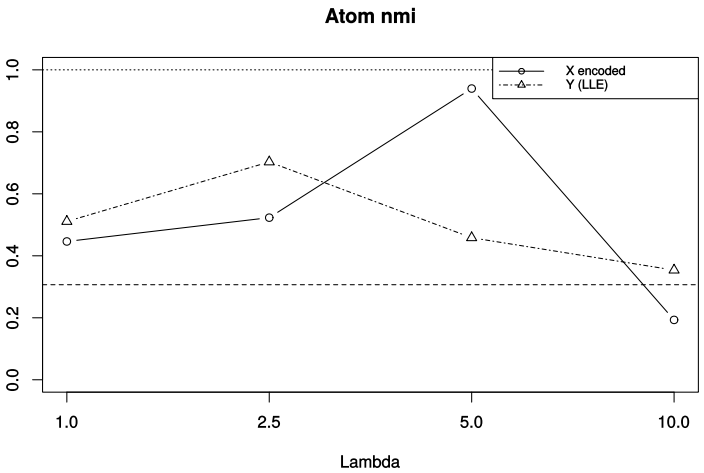


Figure 13: Metrics for **Tetra** data set and different λ.



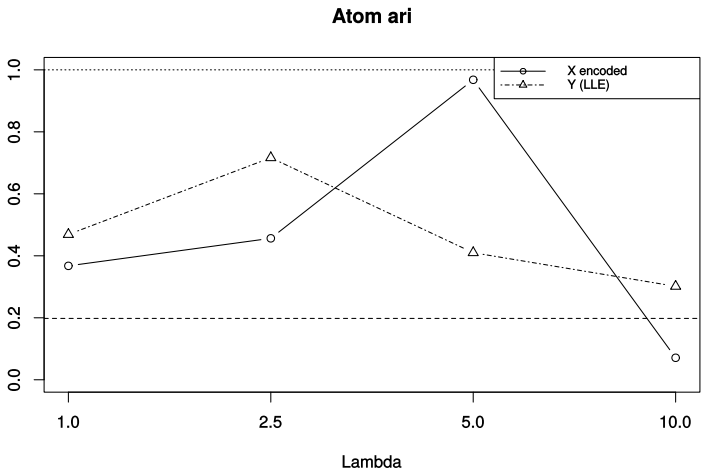


Figure 14: Metrics for **Atom** data set and different λ.

Batch training. Most of the time neural network are trained in mini batch, meaning that the backpropagation is computed for a subset of samples at each iteration and that the whole training data set has been used at each epoch. Unlike full batch, where the backpropagation is computed for the full training data set at once. The major drawback of the latter is that, at a moment, we need to store in memory huge tensor due to the computation of the neural network and size of the data set. Avoiding this drawback is one of the reason why mini batch is used.

However, LLE needs to work on the full data set to be able to take into account its whole structure. Otherwise, if the computation is made on a small subset of the data set, the neighbors found by kNN may not be accurate according to the whole structure. Hence, the second term of the AE-LLE loss of Eq. 4 (corresponding to the LLE loss) also needs to work on the full data set, making it not trainable in mini batch.

At first we have decided to minimize the two terms of the loss in two separate steps, for the first term we train the autoencoder in mini batch then, for the second term the encoder in full batch. However, we found that the results were much better when we minimize the loss at once with the autoencoder in full batch.

The inconvenient is that we suffer from the drawback of the full batch training described before. It's why we had to work on subset of data for MNIST and Coil100 and that we were forced to reduce the neural network architecture for Coil20 and Coil100.

4 Conclusion

Through experiments, we can see that manifold learning algorithms can provide good representations to visualize a data set, even if the new components do not have meaning like for PCA.

Moreover, combining Autoencoder and LLE in a single algorithm where each method has an impact on the other can provide even better representations than both separate methods. As we saw it can also helps the clustering in the reduced latent space. The λ parameter can be interesting to tune, either to speed up convergence or to give more impact of the AE or the LLE to the training.

However, some works could be done in order to have a shorter execution time or to be able to train in mini batch to permit bigger architectures and data sets. Moreover, some experiments with different neural network architectures could be interesting to do in order to see the impact on the clustering in the reduced latent space.

Finally, our github repository is available[2].

References

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[Roweis and Saul, 2000] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326.

[2] https://github.com/MatPont/Deep_Dimensionality_Reduction

Appendices

A Autoencoder Hyper-Parameters for FCPS Data Sets

Layer Type	#neurons	Activation
Input	d	-
Dense	128	Leaky ReLU
Dense	64	Leaky ReLU
Dense	2	Sigmoid
Dense	64	Leaky ReLU
Dense	128	Leaky ReLU
Output	d	Sigmoid

Parameter	Value
Loss	Binary Cross-Entropy
Optimizer	Adam

B AE-LLE Visualization for FCPS Data Sets

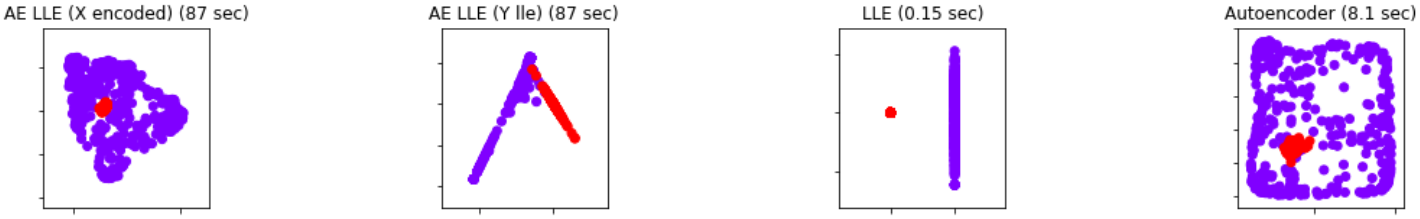


Figure 15: Atom data set experiments with AE-LLE.

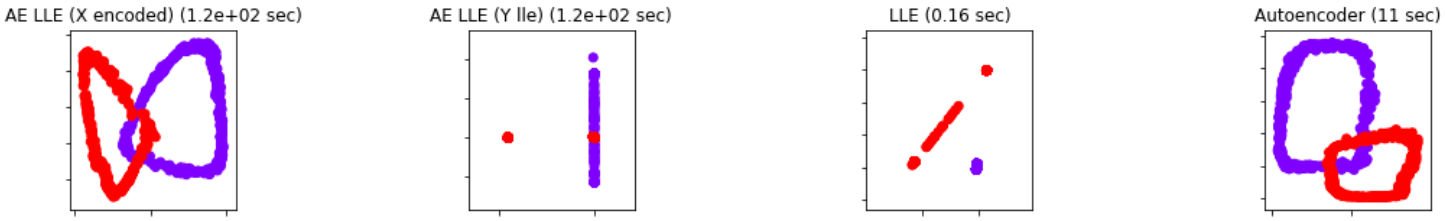


Figure 16: Chainlink data set experiments with AE-LLE.

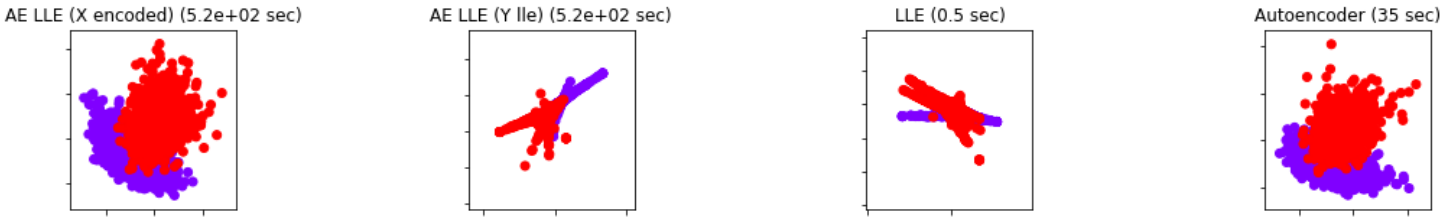


Figure 17: EngyTime data set experiments with AE-LLE.

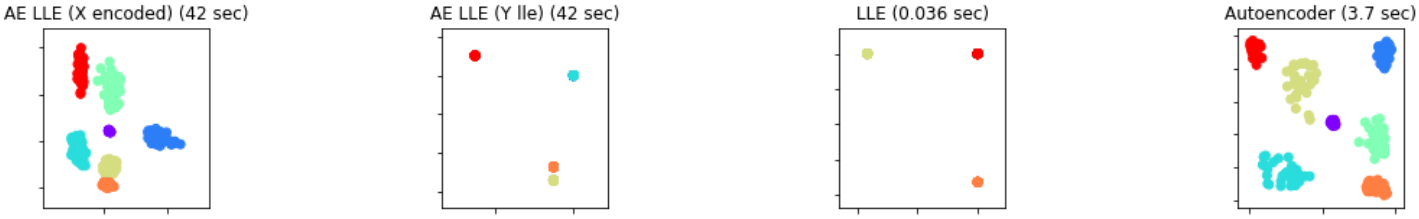


Figure 18: Hepta data set experiments with AE-LLE.

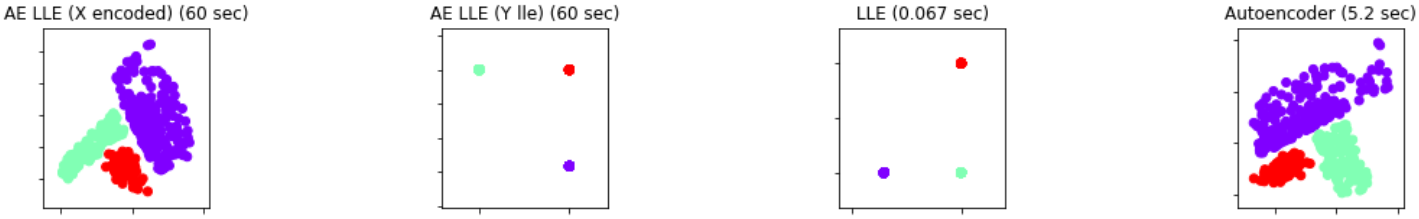


Figure 19: Lsun data set experiments with AE-LLE.

Figure 20: Target data set experiments with AE-LLE.

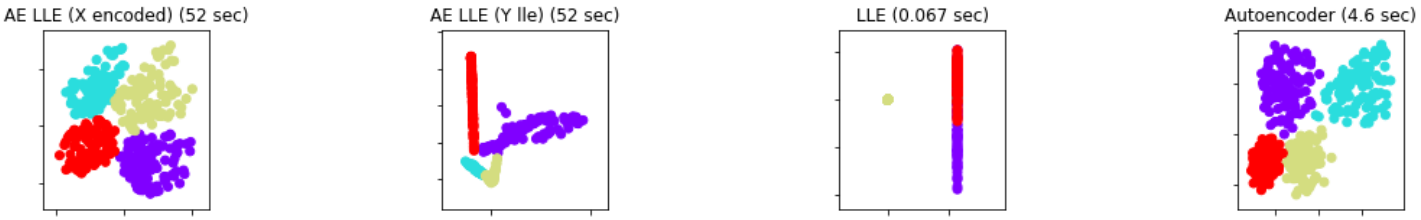


Figure 21: Tetra data set experiments with AE-LLE.

The **Y** representation of AE-LLE seems to better separate classes than the original LLE. We can suppose that the autoencoder training, that works very well on this data set, has a positive impact on the **Y** representation, making it able to better separate the classes.

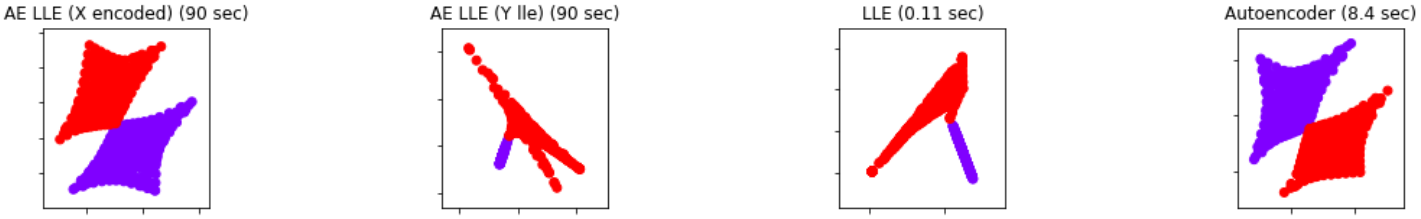


Figure 22: TwoDiamonds data set experiments with AE-LLE.

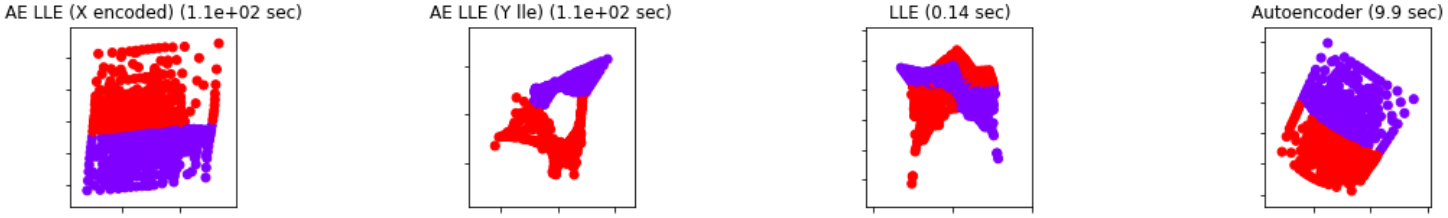


Figure 23: WingNut data set experiments with AE-LLE.

Once again, **Y** of AE-LLE seems to, thanks to the autoencoder training, be able to well separate classes unlike for original LLE.

C Autoencoder Hyper-Parameters for the Images Data Sets

Dataset	USPS	MNIST	ORL	Coil20/100	**Loss**	Binary Cross-Entropy
#trainable parameters	215,169	228,018	685,482	236,601	**Optimizer**	Adam

Architecture for USPS (16 x 16) and MNIST (28 x 28).

Layer Type	**#neurons**	**#filters**	**Kernel**	**Strides**	**Padding**	**Activation**
Input	$d_1 \times d_2$	-	-	-	-	-
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Valid*	Leaky ReLU
Dense	$d_1/4 \times d_2/4$*	-	-	-	-	Sigmoid
Conv2DTranspose	-	128	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(2, 2)*	Same	Leaky ReLU
Output (Conv2D)	-	1	(3, 3)	(1, 1)	Same	Sigmoid

* for USPS: #neurons = $d_1/2 \times d_2/2$, Strides = (1, 1) and Padding = 'Same'

Architecture for ORL (112 x 92).

Layer Type	**#neurons**	**#filters**	**Kernel**	**Strides**	**Padding**	**Activation**
Input	$d_1 \times d_2$	-	-	-	-	-
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2D	-	64	(3, 3)	(1, 1)	Valid	Leaky ReLU
Dense (encoding)	$d_1/8 \times d_2/8$	-	-	-	-	Sigmoid
Dense	$d_1/4 \times d_2/4$	-	-	-	-	Linear
Conv2DTranspose	-	128	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	128	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Output (Conv2D)	-	1	(3, 3)	(1, 1)	Same	Sigmoid

Architecture for Coil20 and Coil100 (both 128 x 128).

Layer Type	**#neurons**	**#filters**	**Kernel**	**Strides**	**Padding**	**Activation**
Input	$d_1 \times d_2$	-	-	-	-	-
Conv2D	-	8	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	16	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2D	-	64	(3, 3)	(1, 1)	Valid	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Dense	$d_1/8 \times d_2/8$	-	-	-	-	Sigmoid
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	16	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	16	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	8	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	8	(3, 3)	(1, 1)	Same	Leaky ReLU
Output (Conv2D)	-	1	(3, 3)	(1, 1)	Same	Sigmoid