Deep Learning for dimensionality reduction

Mahdiou Diallo, Helmy El Rais, and Dah Diarra

June 2020

Abstract

In the context of our dimensionality reduction project for the master's degree at the university of Paris, our task was to implement a new algorithm that combines simulatanously an autoencoder and LLE(Locally Linear Embedding) by optimizing a single objectif function. As there are several method for dimensionality reduction in the literature such as PCA,ISOMAP, EIGENMAP etc, we compare our results to them in order to evaluate it. This report describes our experiments and results on many image datasets.

1 Introduction

Dimensionality reduction consists of learning a new lower-dimensional representation from given data data. Several traditional techniques exist in the literature which are minimize a criterion based on the euclidean distance. That makes them suffer from the curse of dimensionality. In these last years, research was done on combining deep learning and traditional dimensionality reduction techniques to solve that problem.
In order to take advantage of the two concepts we propose a new algorithm which optimizes simultaneously the objective function of a deep autoencoder and the Locally Linear Embedding (LLE) algorithm. In this report, we start by reviewing dimensionality reduction techniques. In the next section, we describe the method we implemented. The following sections describe the datasets used for evaluation, the metrics we used, and finally the results obtained followed by a conclusion.

2 Dimensionality reduction techniques

In the domain of dimensionality reduction there several method, some of theme are describe below:

- **Autoencoder** [3] is an unsupervised artificial neural network that learns how to efficiently encode data then learns how to reconstruct a data from the encoded representation to a representation that is as close to the original data as possible by optimizing the loss between the reconstruct and the original data. Different metrics can be used to calculate the loss and different neural network architectures can also be used for encoding and reconstruction. An autoencoder is mainly used for dimensionality reduction but can also be used for generative models.

- **Locally Linear Embedding (LLE)** [1] is unsupervised learn technique for dimensionality reduction which tries to reduce these n dimensions while trying to preserve the geometric features of the original non-linear feature structure. It first finds the k-nearest neighbors of the points. Then, it approximates each data vector as a weighted linear combination of its k-nearest neighbors. Finally, from these local weights, it computes the data embedding with a global operation that couples all the points.

- **PCA** Principal Component Analysis is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the maximize the variance in the reduced data. PCA is one of the most popular linear dimension reduction technique.

- **ISOMAP** stands for isometric mapping. Isomap is a non-linear dimensionality reduction method based on the spectral theory which tries to preserve the geodesic distances in the lower dimension. Isomap starts by creating a neighborhood network. After that, it uses graph distance to the approximate geodesic distance between all pairs of points. And then, through eigenvalue decomposition of the geodesic distance matrix, it finds the low dimensional embedding of the dataset.

- **Multidimensional scaling (MDS)** [6] is a linear dimensionality reduction method that can be utilized to find a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space. MDS attempt to preserve pairwise distances between each points in the dataset.

3 Method

The basic idea of the algorithm implemented is to simultaneously optimize the cost function of the autoencoder and that of the LLE algorithm. The cost function is defined as follows:

$$\underset{\theta_1,\theta_2,S}{min} ||\mathbf{X} - g_{\theta_2}(f_{\theta_1}(\mathbf{X}))||^2 + \lambda ||f_{\theta_1}(\mathbf{X}) - \mathbf{S} f_{\theta_1}(\mathbf{X})||^2$$

where:

- $\mathbf{X}$ is the input dataset
- f_{θ_1} is the encoder with parameters θ_1
- g_{θ_2} is the decoder with parameters θ_2
- S is the weight matrix obtained by using LLE on the output of the encoder λ the relative importance of the two cost functions

Basically, minimizing this cost function requires finding parameters for which most of the information in the data is retained in the latent space (DAE) while also retaining the relationships of a point with it's neighbors. The algorithm is outlined in 1.

Algorithm 1: Algorithm to minimize the joint cost function

```
input : $X$: the input data matrix
output: $\mathbf{S}$: weight matrix
        $f_{\theta_1}$: the encoding matrix
        $\mathbf{B}$: the LLE embedding matrix

1 repeat
2     (a) - update $\Theta_1$ and $\Theta_2$ using deep AE;
3     (b) - update $S$ using the same strategy as LLE;
4 until convergence;
5 $\mathbf{M} \leftarrow (\mathbf{I}\text{-}\mathbf{S})^T(\mathbf{I} - \mathbf{S})$;
6 $\mathbf{B} \leftarrow eigs(\mathrm{M})$
7 return $\boldsymbol{S}$, $f_{\theta_1}(\boldsymbol{X})$, $\boldsymbol{B}$;
```

In our implementation, implementation of the algorithm we added the option to update lambda after each iteration or batch. Our rationale is that in the initial stages, it is more important to summarize well the dataset (DAE is more important) and after the first few iterations, the LLE reconstruction error becomes gradually more important.
We provided several scheduling functions to update λ:

- constant: the value does not change
- unit step: the value changes from a low value to a high value after a number of batches of iterations
- saturation: similar to the unit step, but the value of λ grows linearly between the low and high values

- sigmoid: the growth function is a sigmoid.

Deep learning models are known to be harder to train than traditional learning algorithms. Several techniques have been proposed to improve the quality of the training. We used the following techniques, some of which are known to work well for DAEs:

- Xavier initialisation for the weights. The bias values are initialized to 0
- Activation functions: here we chose to use the ReLU function
- Optimizer: we used the Adam optimizer

4 Experiments

We compare our algorithm to PCA, UMAP [9], and the original LLE algorithm. For that we reduce the image datasets to 2, 3, 10 dimensions. In each case we use an autoencoder output equal or different from the output dimension of LLE. The setup is summarized in table 1.

Table 1: Description of tests carried out.

Experiment #	Autoencoder	LLE
Exp 1	2	2
Exp 2	8 / 16	2
Exp 3	3	3
Exp 4	8 / 16	3
Exp 5	10	10
Exp 6	32	10

Next we describe the quality measure used to compare the methods.

4.1 Metrics

There are several ways to assess the quality of a dimensionality reduction algorithm. The one we were asked to use was the improvement of a downstream task by using those methods. We also used distance-based and rank-based metrics to evaluate the dimensionality reduction algorithms. [7, 8] explain the most used techniques in the literature. Let $X \in \mathbb{R}^n$ be the input dataset and $Y \in \mathbb{R}^k$ be the reduced dataset. d_{ij} represents the distance between points x_i and x_j in X while δ_{ij} is the distance between the corresponding points y_i and y_j in Y.

4.1.1 Downstream task

We run K-means clustering on the X and Y and compared their resulting normalized mutual indices (NMI) scores with respect to the real classes of the points. We expect that clustering on the result of a good dimensionality reduction technique will have a better NMI score.

4.1.2 Distance based measures

Those mesures assess the conservation of the distances from the original space into the reduced space.

- Root Mean Squared Error (RMSE): It measures for each point, the average change in distance from X to Y.

$$\text{rmse}(x_i, y_i) = \sqrt{\frac{\sum_{j=1}^{n}(d_{ij} - \delta_{ij})^2}{n}}$$

$$\text{RMSE}(X, Y) = \frac{\sum_{i=1}^{n} \text{rmse}(x_i, y_i)}{n}$$

- Kruskal's stress: This metric is similar to RMSE but penalizes small distances more

$$\text{kruskal}(x_i, y_i) = \sqrt{\frac{\sum_{j=1}^{n}(d_{ij} - \delta_{ij})^2}{d_{ij}}}$$

$$\text{KRUSKAL}(X, Y) = \frac{\sum_{i=1}^{n} \text{kruskal}(x_i, y_i)}{n}$$

4.1.3 Rank based methods

Those methods assess the conservation of the neighborhood order.

- Spearman's Rho: It works by converting the distances d_{ij} and δ_{ij} into ranks r_{ij} and ρ_{ij} respectively. Those ranks represent how close X_j is to X_i compared to the other points. From these ranks, the correlation between the ranks in the original space and the latent space is computed as:

$$\text{spearman}(x_i, y_i) = 1 - 6 \sum_{j=1}^{n} \frac{r_{ij} - \rho_{ij}^2}{n(n^2 - 1)}$$

$$\text{SPEARMAN}(X, Y) = \frac{\sum_{i=1}^{n} \text{spearman}(x_i, y_i)}{n}$$

- Neighborhood loss: It measures how many of the k-nearest neighbors in the original space are kept in the latent space. If $n_k(x_i)$ and $\nu_k y_i \nu_k(y_i)$ represent the k-nearest neighbors of the point i in the original and latent space, respectively, the neighborhood loss is computed as:

$$\text{neighborhood\_loss}(x_i, y_i) = 1 - \frac{|n_k(x_i) \cup \nu_k(y_i)|}{k}$$

$$\text{NEIGHBORHOOD\_LOSS}(X, Y) = \frac{\sum_{i=1}^{n} \text{neighborhood\_loss}(x_i, y_i)}{n}$$

- Co-ranking framework: It is a framework based on the construction of a co-ranking matrix Q_{kl} given as:

$$Q_{kl} = |\{(i, j)|r_{ij} = k \text{ and } \rho_{ij} = l\}|$$

  This matrix counts the number of times when a point X_j in the original space is the k-nearest neighbor of X_i while y_j is the l-nearest neighbor of y_i in the latent space. If the order is perfectly conserved, Q_{kl} is a diagonal matrix. The quality of the reduction can be measured as a function of the off-diagonal entries. A simple quality measure explained and improved in [4] is Q_{NX}.

$$Q_{NX} = \frac{1}{KN} \sum_{k=1}^{K} \sum_{l=1}^{K} Q_{kl}$$

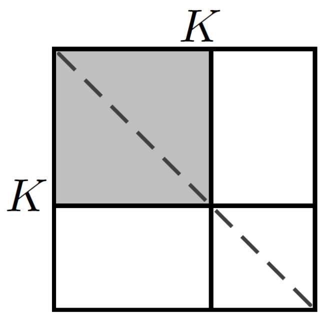
  In [5], another improvement of $Q_N X$ is Q_{ND} where, in stead of taking a square region of Q_{kl} of size K, a region of interest K_s and a maximum tolerated rank error κ_t that represents how many off-diagonal entries will be considered.

$$Q_{ND} = \frac{1}{K_s N} \sum_{i \leq K} \sum_{j:|i-j| \leq \kappa_t} Q_{kl}$$

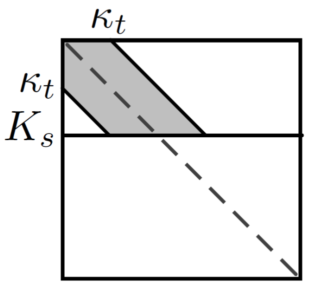
  [5] shows that Q_{ND} gives more stable results as the number of neighbors increases. Figure 1 illustrates the difference between the regions considered by Q_{NX} and Q_{ND}

4.2 Datasets

In this project we worked on two type of data as asked: synthetic datasets (FCPS) and images datasets.



(a) Region used by Q_{NX}



(b) Region used by Q_{ND}

Figure 1: Comparison of regions used by Q_{NX} and Q_{ND}

4.2.1 FCPS

As describe in FCPS paper [2] the Fundamental Clustering Problems Suite (FCPS) offers a variety of clustering problems any algorithm shall be able to handle when facing real world data. FCPS serves as an elementary benchmark for clustering algorithms. We use the 3-dimensional datasets to compare the visualisation quality of the used algorithms.

4.2.2 Image datasets

Image datasets is a set of real data and was mainly used to evaluate our method. As image separation is a hard task these data with different structure can show how good our method is compare to some classic one. Table 2 describes the image datasets we used.

Table 2: Description of Image datasets

datasets	# samples	# features	# classes	sparsity
Coil20	1440	1024	20	34.48
Coil100	7200	1024	20	0
ORL	400	1024	40	0
Yale	165	1024	15	30.54
USPS	9298	256	10	0
MNIST	70000	784	10	80.85

5 Results

In figure 2 we use the Tetra dataset from FCPS to show that a lower value of loss leads to a better representation.

Figures 3, 4, and 5 show the reduced version of Tetra, Chainlink, and USPS datasets by the considered algorithms. It is clear that UMAP offers the best class separation. That was also seen for the other FCPS datasets. The other embeddings have comparable visual quality. To differentiate them, quantitative methods are required.

Table 3 shows the NMI values obtained by running the k-means algorithm on the dataset in the original number of dimensions.

Table 3: NMI values for k-means on original space.

ORL	YALE	COIL20	COIL100	USPS	MNIST
0.783	0.561	0.784	0.773	0.613	0.493

Table 4 shows the comparison of the vanilla LLE embeddings, the DAE embeddings, and the LLE embeddings from the DAE. We can see that on the ORL dataset, vanilla LLE has the best results in

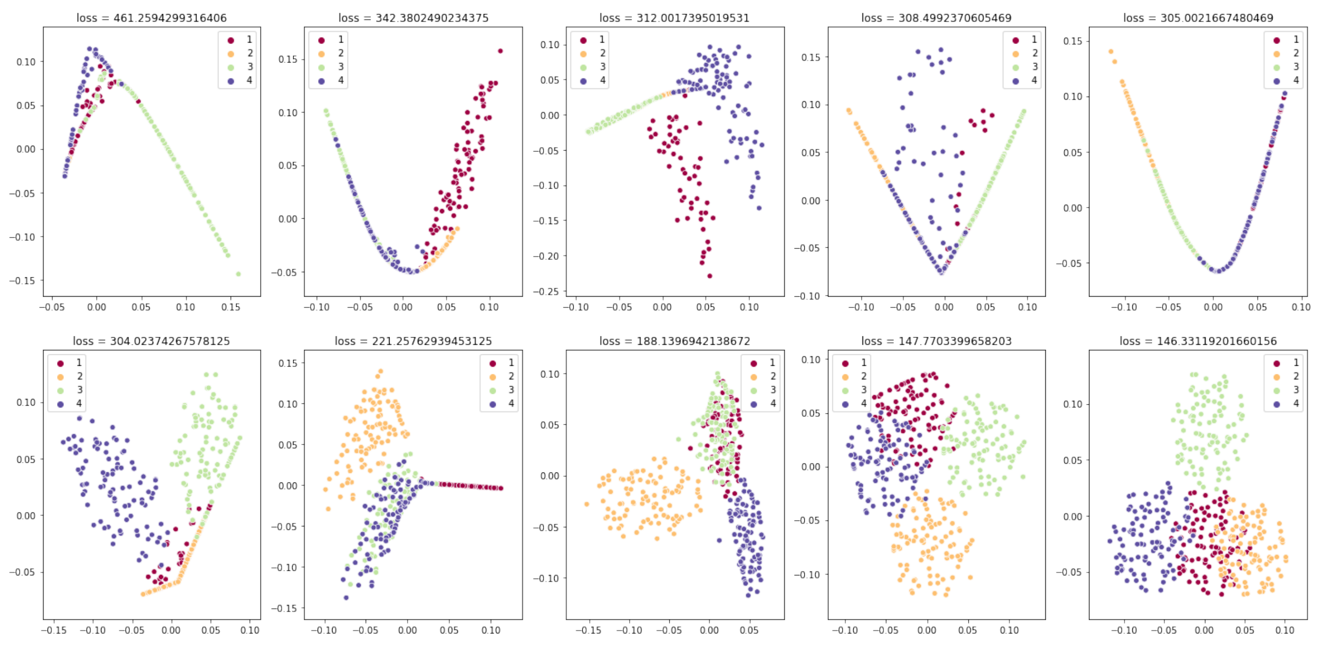


Figure 2: Improvement of reduction quality with loss.

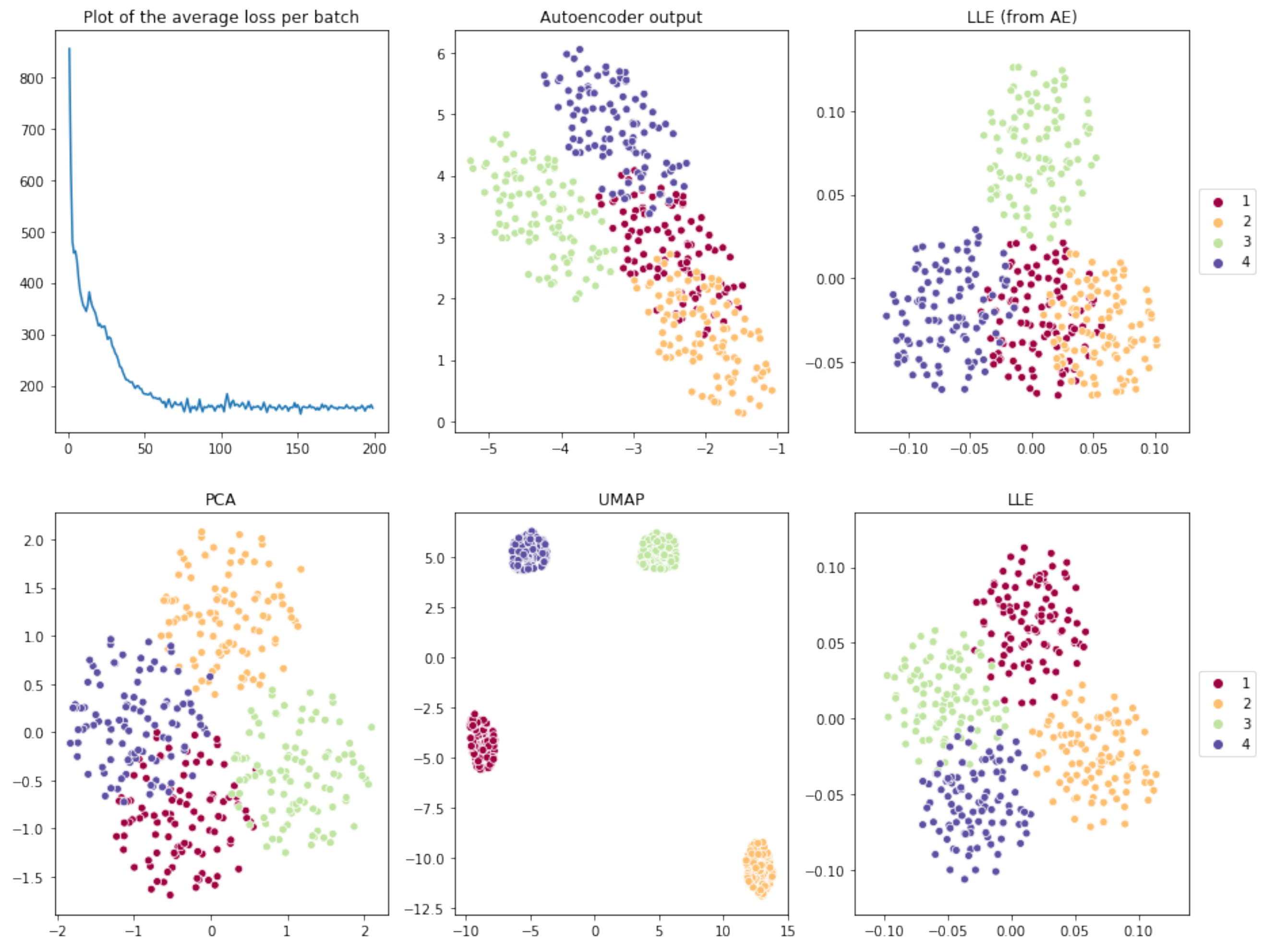


Figure 3: Comparison of DR methods on the Tetra dataset

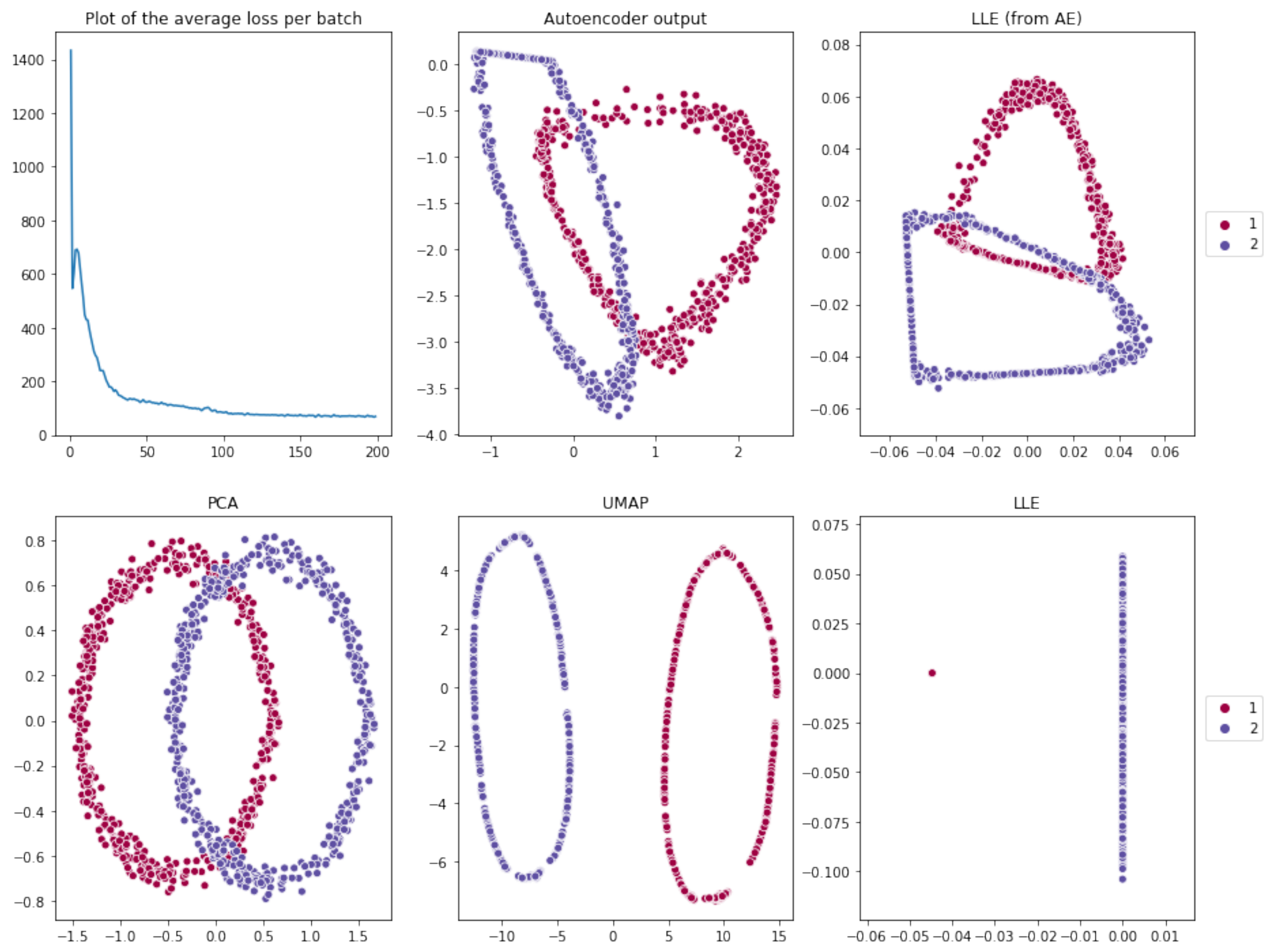


Figure 4: Comparison of DR methods on the Chainlink dataset

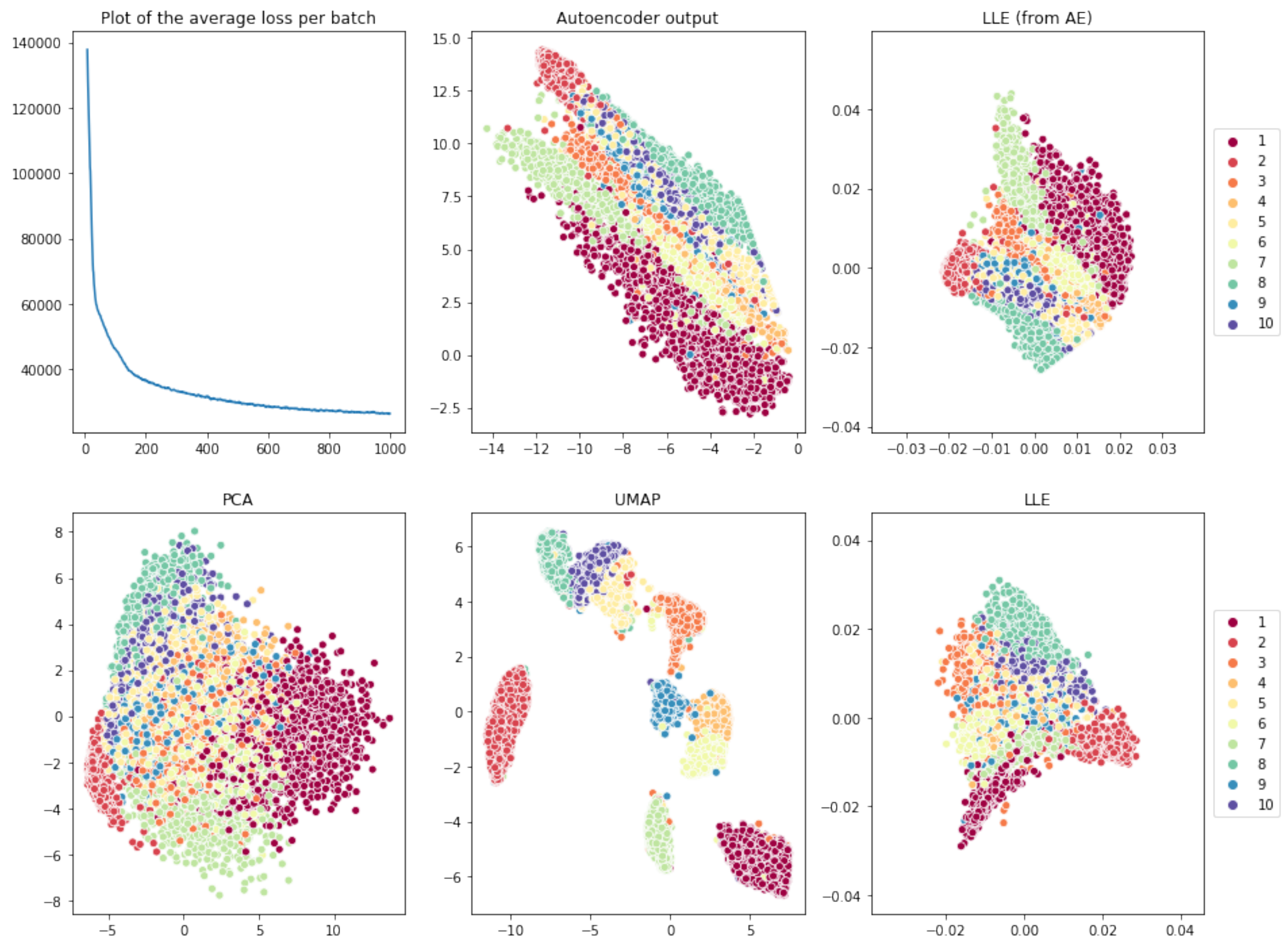


Figure 5: Comparison of DR methods on the USPS dataset

most cases although in the 10-dimensional embedding case the DAE embeddings have the best results. On the other datasets, we can see a clear improvement over LLE on almost all metrics. Sometimes the improvement is two-fold. Also, although the resulting NMI is generally lower than the one on the input space (shown in table 3), the decrease is not too large and in the case of MNIST, the NMI is even better.

Dataset	# dims	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Spearman	Nbhood loss	NMI
ORL	2	LLE	**0.433**	**0.342**	1511.980	1.000	**0.092**	**0.629**	**0.653**
		AE	0.422	0.311	**1286.683**	**0.728**	0.087	0.661	0.549
		AE_LLE	0.419	0.314	1511.977	1.000	0.091	0.661	0.536
	3	LLE	**0.463**	**0.371**	1511.959	1.000	**0.076**	**0.599**	**0.710**
		AE	0.400	0.305	**1272.924**	**0.712**	0.063	0.667	0.531
		AE_LLE	0.432	0.324	1511.955	1.000	0.073	0.652	0.543
	10	LLE	0.558	0.459	1511.863	1.000	0.050	**0.514**	**0.780**
		AE	**0.592**	**0.465**	**1133.645**	**0.565**	**0.099**	0.516	0.620
		AE_LLE	0.535	0.431	1511.854	1.000	0.041	0.547	0.644
YALE	2	LLE	0.494	0.392	2274.405	1.000	0.020	0.582	0.338
		AE	0.526	0.407	**1995.059**	**0.775**	0.020	0.570	0.398
		AE_LLE	**0.579**	**0.443**	2274.387	1.000	**0.051**	**0.540**	**0.418**
	3	LLE	0.502	0.395	2274.372	1.000	0.027	0.581	0.353
		AE	0.629	0.495	**1907.828**	**0.709**	0.061	0.489	**0.449**
		AE_LLE	**0.670**	**0.531**	2274.352	1.000	**0.066**	**0.458**	0.441
	10	LLE	0.625	0.520	2274.191	1.000	0.037	0.454	**0.553**
		AE	**0.719**	**0.564**	**1804.003**	**0.633**	**0.051**	**0.421**	0.441
		AE_LLE	0.632	0.504	2274.188	1.000	0.013	0.476	0.427
COIL20	2	LLE	0.579	0.447	11.219	0.993	-0.037	0.547	0.649
		AE	**0.719**	**0.575**	**8.283**	**0.545**	**0.046**	**0.420**	**0.670**
		AE_LLE	0.499	0.370	11.225	0.994	0.011	0.621	0.578
	3	LLE	0.636	0.497	11.214	0.992	-0.049	0.498	0.700
		AE	**0.824**	**0.668**	**8.786**	**0.610**	**0.071**	**0.329**	**0.745**
		AE_LLE	0.705	0.572	11.206	0.991	-0.014	0.423	0.711
	10	LLE	0.734	0.596	11.154	0.981	-0.060	0.397	**0.770**
		AE	**0.870**	**0.727**	**7.325**	**0.424**	0.043	**0.268**	0.763
		AE_LLE	0.767	0.640	11.151	0.981	**0.050**	0.356	0.737
COIL100	2	LLE	0.367	0.283	2058.808	1.000	**0.008**	0.705	0.621
		AE	0.401	**0.313**	**1532.396**	**0.555**	0.006	**0.675**	**0.629**
		AE_LLE	**0.403**	0.312	2058.802	1.000	0.004	0.676	0.628
	3	LLE	0.420	0.331	2058.805	1.000	0.006	0.659	0.674
		AE	**0.553**	**0.439**	**1178.646**	**0.330**	**0.011**	**0.552**	**0.700**
		AE_LLE	0.450	0.357	2058.806	1.000	0.007	0.631	0.647
	10	LLE	0.574	0.482	2058.781	1.000	0.008	0.508	**0.744**
		AE	**0.696**	**0.569**	**1130.403**	**0.303**	**0.013**	**0.424**	0.721
		AE_LLE	0.560	0.460	2058.780	1.000	0.007	0.530	0.685
USPS	2	LLE	0.173	0.128	11.579	0.997	0.000	0.856	0.498
		AE	0.255	0.194	**6.195**	**0.300**	0.001	0.792	0.457
		AE_LLE	**0.285**	**0.213**	11.579	0.997	**0.001**	**0.773**	**0.534**
	3	LLE	0.248	0.192	11.574	0.996	0.001	0.793	0.449
		AE	**0.368**	**0.287**	6.782	0.347	**0.001**	**0.701**	**0.517**
		AE_LLE	0.364	0.285	11.574	0.996	0.000	0.703	0.487
	10	LLE	0.481	0.386	11.554	0.992	**0.001**	0.603	0.596
		AE	**0.604**	**0.490**	**6.357**	**0.302**	0.001	**0.501**	0.524
		AE_LLE	0.545	0.435	11.553	0.992	0.001	0.555	**0.631**
MNIST	2	LLE	0.117	0.086	2611.597	1.000	0.000	0.896	0.207

Dataset	# dims	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Spearman	Nbhood loss	NMI
		AE	**0.209**	**0.165**	**1521.310**	**0.347**	0.000	**0.820**	**0.467**
		AE_LLE	0.208	0.164	2611.592	1.000	0.000	0.820	0.460
	3	LLE	0.156	0.121	2611.594	1.000	0.000	0.863	0.214
		AE	**0.388**	**0.309**	**1501.125**	**0.335**	0.000	**0.678**	**0.540**
		AE_LLE	0.380	0.304	2611.587	1.000	0.000	0.684	0.523
	10	LLE	0.352	0.285	2611.575	1.000	0.000	0.701	0.511
		AE	0.416	0.350	**1138.463**	**0.194**	0.000	0.638	0.290
		AE_LLE	**0.483**	**0.394**	2611.567	1.000	0.000	**0.595**	**0.592**

Table 4: Improvement of the method over LLE

Next we compare the embeddings of the different methods on each dataset. The results are shown in tables 5, 6, 7, 8, 9, and 10. They show that PCA usually has better results on distance based metrics which is expected. On rank based metrics, UMAP usually dominates unless the output dimension is high enough. In those cases, PCA gives better results. UMAP has the best NMI results. Its resulting embeddings are well suited to clustering. We can also see that vanilla LLE usually has the worst performance, so the improvement made by our method is welcome. Additional results can be found in our github repository [1].

6 Conclusion

In this project we have seen how we can combine the optimization objectives of two algorithms to improve their results by a simultaneous optimization. We saw the clear improvements made by LLE by using DAE although the resulting model did not beat UMAP.
Better results could be obtained by paying more attention to the activation functions, initializations, optimizers, and training strategies used on the DAE (simulated annealing for example).

References

[1] Sam T. Roweis and Lawrence K. Saul. "Nonlinear dimensionality reduction by locally linear embedding." In: *Science* 290 5500 (2000), pp. 2323–6.

[2] Alfred Ultsch. *Fundamental Clustering Problems Suite (FCPS).* Jan. 2005. DOI: 10.13140/RG.2.1.2394.5446.

[3] Geoffrey E. Hinton and Ruslan Salakhutdinov. "Reducing the dimensionality of data with neural networks." In: *Science* 313 5786 (2006), pp. 504–7.

[4] Wouter Lueks et al. "How to Evaluate Dimensionality Reduction? - Improving the Co-ranking Matrix". In: *ArXiv* abs/1110.3917 (2011).

[5] Bassam Mokbel et al. "Visualizing the quality of dimensionality reduction". In: *Neurocomputing* 112 (2012), pp. 109–123.

[6] Michael C. Hout, Megan H Papesh, and Stephen D. Goldinger. "Multidimensional scaling." In: *Wiley interdisciplinary reviews. Cognitive science* 4 1 (2013), pp. 93–103.

[7] Antonio Gracia Berná et al. "A methodology to compare Dimensionality Reduction algorithms in terms of loss of quality". In: *Inf. Sci.* 270 (2014), pp. 1–27.

[8] Bastian Alexander Rieck and Heike Leitte. "Agreement Analysis of Quality Measures for Dimensionality Reduction". In: 2015.

[9] Leland McInnes and John Healy. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *ArXiv* abs/1802.03426 (2018).

[1] https://github.com/mahdiou/Projet_Red_Dim

Table 5: ORL results

# dims	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Spearman	Nbhood loss	NMI
2	PCA	0.451	0.336	**697.061**	**0.228**	**0.101**	0.639	0.558
	LLE	0.433	0.342	1511.980	1.000	0.092	0.629	0.653
	UMAP	**0.616**	**0.521**	1508.737	0.996	0.096	**0.455**	**0.777**
	AE	0.422	0.311	1286.683	0.728	0.087	0.661	0.549
	AE_LLE	0.419	0.314	1511.977	1.0	0.091	0.661	0.536
3	PCA	0.603	0.457	**573.856**	**0.154**	**0.107**	0.522	0.634
	LLE	0.463	0.371	1511.959	1.000	0.076	0.599	0.71
	UMAP	**0.664**	**0.555**	1508.888	0.996	0.09	**0.421**	**0.782**
	AE	0.400	0.305	1272.924	0.712	0.063	0.667	0.531
	AE_LLE	0.432	0.324	1511.955	1.0	0.073	0.652	0.543
10	PCA	**0.906**	**0.723**	**321.536**	**0.048**	**0.130**	**0.267**	0.746
	LLE	0.558	0.459	1511.863	1.000	0.05	0.514	0.78
	UMAP	0.689	0.574	1509.087	0.996	0.104	0.405	**0.786**
	AE	0.592	0.465	1133.645	0.565	0.099	0.516	0.620
	AE_LLE	0.535	0.431	1511.854	1.0	0.041	0.547	0.644

Table 6: YALE results

# dim	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Nbhood loss	NMI
2	PCA	0.597	0.458	**1151.087**	**0.275**	0.528	0.414
	LLE	0.494	0.392	2274.405	1.000	0.582	0.338
	UMAP	**0.693**	**0.555**	2271.623	0.997	**0.425**	**0.502**
	AE	0.526	0.407	1995.059	0.775	0.57	0.398
	AE_LLE	0.579	0.443	2274.387	1.0	0.54	0.418
3	PCA	0.703	0.559	**924.951**	**0.185**	0.433	0.465
	LLE	0.502	0.395	2274.372	1.000	0.581	0.353
	UMAP	**0.724**	**0.580**	2271.647	0.997	**0.403**	**0.538**
	AE	0.629	0.495	1907.828	0.709	0.489	0.449
	AE_LLE	0.67	0.531	2274.352	1.0	0.458	0.441
10	PCA	**0.990**	**0.809**	**445.289**	**0.042**	**0.193**	0.505
	LLE	0.625	0.52	2274.191	1.000	0.454	**0.553**
	UMAP	0.741	0.605	2271.888	0.998	0.381	0.552
	AE	0.719	0.564	1804.003	0.633	0.421	0.441
	AE_LLE	0.632	0.504	2274.188	1.0	0.476	0.427

Table 7: COIL20 results

# dim	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Nbhood loss	NMI
2	PCA	0.579	0.451	**4.913**	**0.200**	0.541	0.638
	LLE	0.579	0.447	11.219	0.993	0.547	0.649
	UMAP	**0.869**	**0.753**	5.34	0.234	**0.242**	**0.823**
	AE	0.719	0.575	8.283	0.545	0.42	0.67
	AE_LLE	0.499	0.370	11.225	0.994	0.621	0.578
3	PCA	0.741	0.585	**3.960**	**0.129**	0.41	0.726
	LLE	0.636	0.497	11.214	0.992	0.498	0.700
	UMAP	**0.876**	**0.758**	5.792	0.279	**0.237**	**0.818**
	AE	0.824	0.668	8.786	0.61	0.329	0.745
	AE_LLE	0.705	0.572	11.206	0.991	0.423	0.711
10	PCA	**0.952**	**0.805**	**1.928**	**0.031**	**0.192**	0.796
	LLE	0.734	0.596	11.154	0.981	0.397	0.77
	UMAP	0.876	0.765	5.686	0.27	0.231	**0.819**
	AE	0.87	0.727	7.325	0.424	0.268	0.763
	AE_LLE	0.767	0.64	11.151	0.981	0.356	0.737

Table 8: COIL100 results

# dim	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Nbhood loss	NMI
2	PCA	0.411	0.319	**894.268**	**0.195**	0.669	0.662
	LLE	0.367	0.283	2058.808	1.000	0.705	0.621
	UMAP	**0.721**	**0.578**	2049.99	0.991	**0.414**	**0.770**
	AE	0.401	0.313	1532.396	0.555	0.675	0.629
	AE_LLE	0.403	0.312	2058.802	1.0	0.676	0.628
3	PCA	0.536	0.426	**737.924**	**0.134**	0.564	0.721
	LLE	0.420	0.331	2058.805	1.0	0.659	0.674
	UMAP	**0.744**	**0.599**	2051.381	0.993	**0.393**	**0.779**
	AE	0.553	0.439	1178.646	0.33	0.552	0.7
	AE_LLE	0.45	0.357	2058.806	1.000	0.631	0.647
10	PCA	**0.801**	**0.660**	**419.424**	**0.043**	**0.334**	0.76
	LLE	0.574	0.482	2058.781	1.000	0.508	0.744
	UMAP	0.753	0.609	2051.772	0.993	0.383	**0.778**
	AE	0.696	0.569	1130.403	0.303	0.424	0.721
	AE_LLE	0.560	0.460	2058.78	1.0	0.530	0.685

Table 9: USPS results

# dim	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Nbhood loss	NMI
2	pca	$\underline{0.169}$	$\underline{0.124}$	5.496	0.236	$\underline{0.860}$	$\underline{0.433}$
	LLE	0.173	0.128	$\underline{11.579}$	$\underline{0.997}$	0.856	0.498
	UMAP	**0.506**	**0.388**	**4.978**	**0.193**	**0.602**	**0.822**
	AE	0.255	0.194	6.195	0.3	0.792	0.457
	AE_LLE	0.285	0.213	11.579	0.997	0.773	0.534
3	pca	0.31	0.236	**4.518**	**0.162**	0.75	$\underline{0.430}$
	LLE	$\underline{0.248}$	$\underline{0.192}$	11.574	0.996	$\underline{0.793}$	0.449
	UMAP	**0.588**	**0.460**	5.182	0.211	**0.531**	**0.822**
	AE	0.368	0.287	6.782	0.347	0.701	0.517
	AE_LLE	0.364	0.285	$\underline{11.574}$	$\underline{0.996}$	0.703	0.487
10	pca	**0.834**	**0.659**	**1.771**	**0.025**	**0.335**	0.6
	LLE	$\underline{0.481}$	$\underline{0.386}$	$\underline{11.554}$	$\underline{0.992}$	$\underline{0.603}$	0.596
	UMAP	0.619	0.49	5.254	0.217	0.501	**0.823**
	AE	0.604	0.49	6.357	0.302	0.501	$\underline{0.524}$
	AE_LLE	0.545	0.435	11.553	0.992	0.555	0.631

Table 10: MNIST results

# dim	Method	Q_{ND}	Q_{NX}	RMSE	Kruskal	Nbhood loss	NMI
2	PCA	$\underline{0.110}$	$\underline{0.080}$	1673.114	0.416	$\underline{0.903}$	0.368
	LLE	0.117	0.086	$\underline{2611.597}$	$\underline{1.000}$	0.896	$\underline{0.207}$
	UMAP	**0.456**	**0.349**	2605.637	0.995	**0.640**	**0.757**
	AE	0.209	0.165	**1521.310**	**0.347**	0.82	0.467
	AE_LLE	0.208	0.164	2611.592	1.0	0.82	0.46
3	PCA	0.195	0.146	**1462.178**	**0.318**	0.839	0.351
	LLE	$\underline{0.156}$	$\underline{0.121}$	$\underline{2611.594}$	$\underline{1.000}$	$\underline{0.863}$	$\underline{0.214}$
	UMAP	**0.528**	**0.415**	2606.446	0.996	**0.575**	**0.736**
	AE	0.388	0.309	1501.125	0.335	0.678	0.54
	AE_LLE	0.38	0.304	2611.587	1.0	0.684	0.523
10	PCA	**0.647**	**0.512**	**832.258**	**0.104**	**0.479**	0.466
	LLE	$\underline{0.352}$	$\underline{0.285}$	$\underline{2611.575}$	$\underline{1.000}$	$\underline{0.701}$	0.511
	UMAP	0.552	0.439	2606.647	0.996	0.551	**0.734**
	AE	0.416	0.35	1138.463	0.194	0.638	$\underline{0.290}$
	AE_LLE	0.483	0.394	2611.567	1.0	0.595	0.592