SLT coding exercise #1

Locally Linear Embedding

https://gitlab.vis.ethz.ch/vwegmayr/slt-coding-exercises

Due on Monday, March 6th, 2017

Amirreza Bahreini
11-801-495

Contents

The Model

The model section is intended to allow you to recapitulate the essential ingredients used in Locally Linear Embedding. Write down the *necessary* equations to specify Locally Linear Embedding and and shortly explain the variables that are involved. This section should only introduce the equations, their solution should be outlined in the implementation section.
Hard limit: One page

The idea behind Locally Linear Embedding (LLE) is that we can learn non linear structure in the data by patching together lots of linear structures in smaller regions. It essentially finds a neighbourhood for each data point, finds the weights for linearly approximating the data in that neighbourhood and finally finds the low-dimensional coordinates best reconstructed by those weights.
More precisely, given the data matrix $\mathbf{X}_{n\times p}$, a desired dimension $q < p$ and a fix number of neighbours k for each data point, LLE outputs a matrix $\mathbf{Y}_{n\times q}$. Following steps show how LLE works:

1. Find the k nearest neighbour for each data point x_i. Of course here the concept of "near" is directly related to the chosen distance metric.

2. Find the weight matrix $\mathbf{W}$ which minimises the following error:

$$\epsilon(\mathbf{W}) = \Sigma_i |x_i - \Sigma_j W_{ij} x_j|^2$$

   where $W_{ij} = 0$ if x_j does not belong to the set of neighbours found in the first step, and for each i $\Sigma_j W_{ij} = 1$

3. Find the coordinates $\mathbf{Y}$ which minimises the reconstruction using the obtained weights in the second step:

$$\Theta(\mathbf{Y}) = \Sigma_i |y_i - \Sigma_j W_{ij} y_j|^2$$

   where $\Sigma_i Y_{ij} = 0$ and $\mathbf{Y}^\intercal \mathbf{Y} = \mathbf{I}$

Essentially we would like to keep the same neighbouring structure in a lower dimensional space.

The Questions

This is the core section of your report, which contains the tasks for this exercise and your respective solutions. Make sure you present your results in an illustrative way by making use of graphics, plots, tables, etc. so that a reader can understand the results with a single glance. Check that your graphics have enough resolution or are vector graphics. Consider the use of GIFs when appropriate.
Hard limit: Two pages

(a) Get the data

For this exercise we will work with the MNIST data set. In order to learn more about it and download it, go to http://yann.lecun.com/exdb/mnist/.

(b) Locally linear embedding

Implement the LLE algorithm and apply it to the MNIST data set. Provide descriptive visualizations for 2D & 3D embedding spaces. Is it possible to see clusters?

(c) Cluster structure

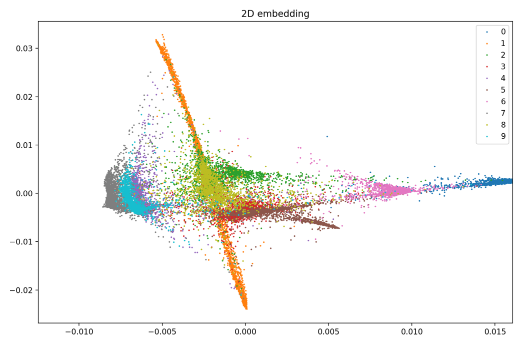
Investigate the cluster structure of the data. Can you observe block structures in the M matrix (use matrix plots)? Also plot the singular values of M. Do you notice something? Can you think of ways to determine the optimal embedding dimension?

(d) Nearest Neighbors

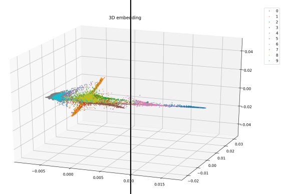
Investigate the influence of the choice of how many nearest neighbors you take into account. Additionally, try different metrics to find the nearest neighbors (we are dealing with images!).

(e) Linear manifold interpolation

Assume you pick some point in the embedding space. How can you map it back to the original (high dimensional) space? Investigate how well this works for points within and outside the manifold (does it depend on the dimensionality of the embedding space?) Try things like linearly interpolating between two embedding vectors and plot the sequence of images along that line. What happens if you do that in the original space?



(a) 2D embedding



(b) 3D embedding

Figure 1: Embeddings of 20000 data points of MINST dataset using 10 neighbours and L1 distance. Although not perfect, we can see a kind of clustering between all 10 numbers, where there are of course some overlapping points. We can also observe that the 3D embedding does not use all the space available in order to separate clusters from each other. This is of course in contrast from PCA which tries to maximises the variance; here we try to keep the neighbouring structure.

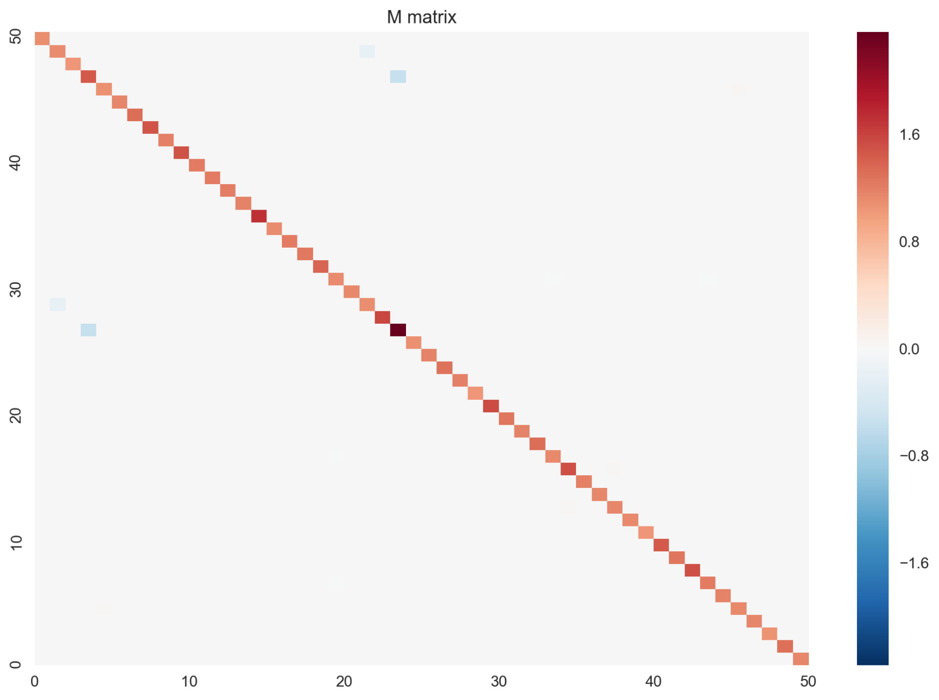
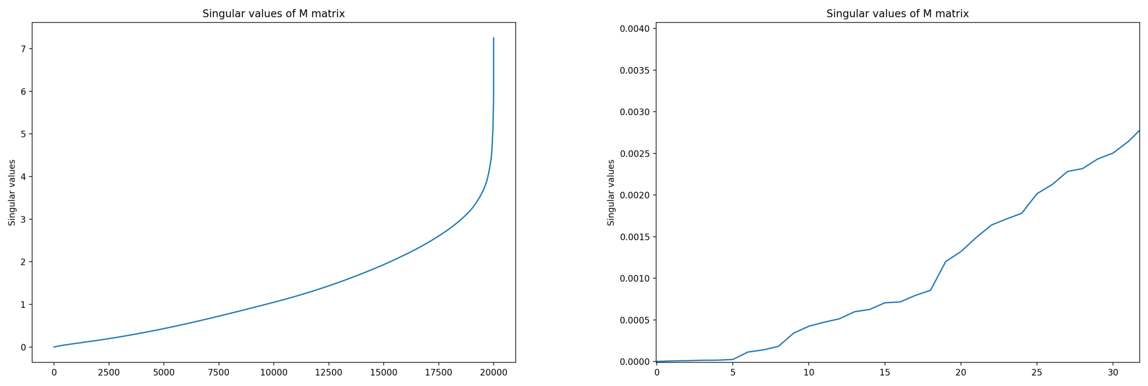


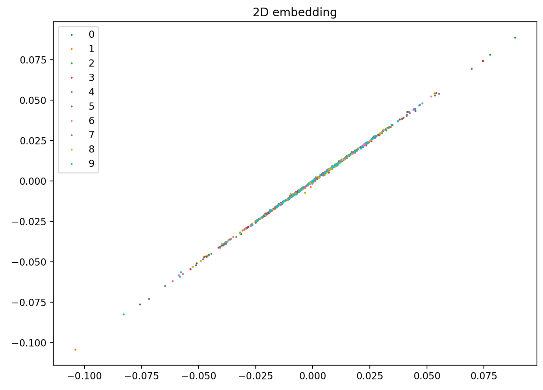
Figure 2: Only a small portion of the matrix $\mathbf{M}$, precisely $\mathbf{M}_{[0:50][0:50]}$. This small portion is representative of the entire matrix as $\mathbf{M}$ seems to be very sparse, with positive values on the secondary diagonal and few negative values elsewhere.



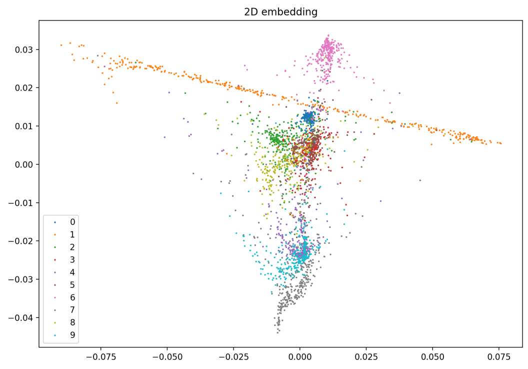
(a) Singular values for all 20000 dimensions

(b) Zoomed on only first few dimensions

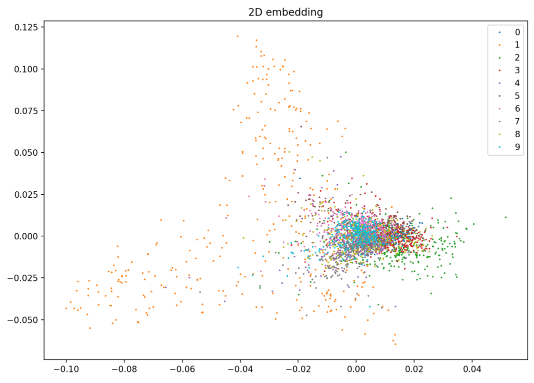
Figure 3: Singular values of the **M** matrix. We may guess that 5 dimensions is the optimal number of dimensions to take as the singular values still stay very small, but we will have also more dimensions for to express a better embedding. The problem with more dimensions is that the singular values increase and thus the objective value of our error function, which means that we lose some desired neighbouring structure.



(a) $k = 1$

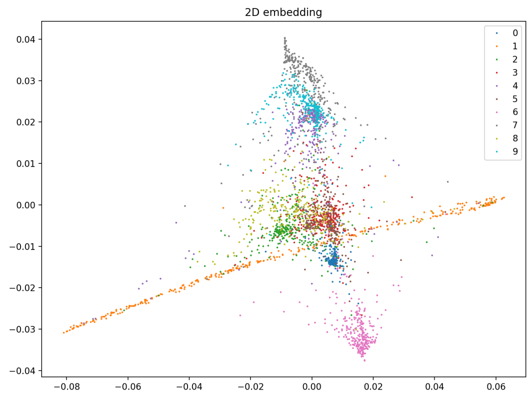


(b) $k = 11$

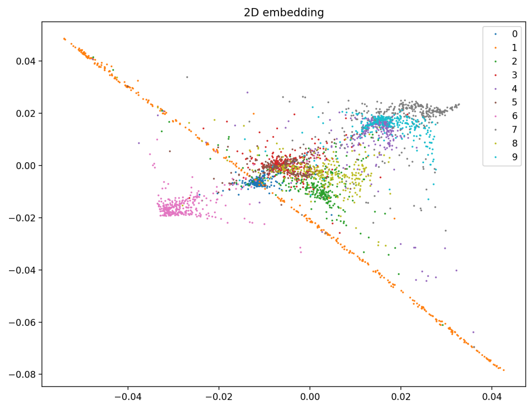


(c) $k = 111$

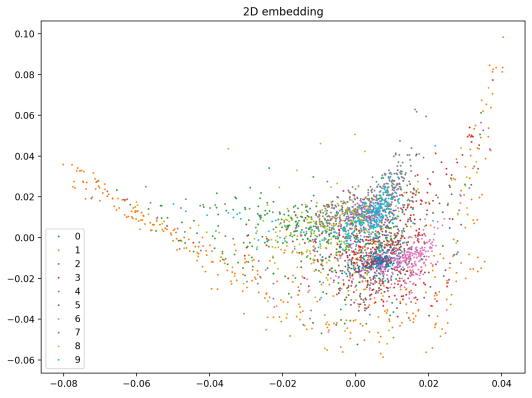
Figure 4: Effect of number of neighbours on embeddings. Notice that the first plot is a normal consequence of having less neighbours than actually desired number of dimensions, here 2. The second plot shows a good clustering compared to all other numbers of neighbours. Last figure is representative of almost any $k > 20$. The reasons is discussed in the last section.



(a) L1 distance

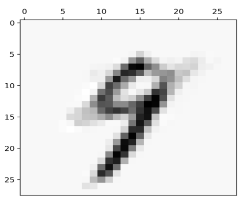


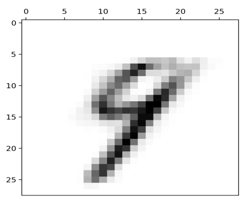
(b) L2 distance

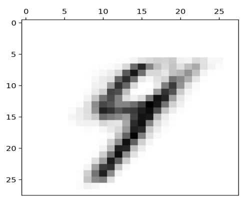


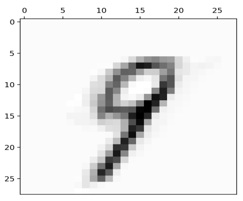
(c) Hamming distance

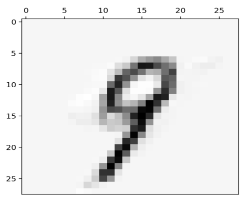
Figure 5: Effect of distance metric on the clustering.


(a) Taken from the manifold


(b) Outside of the manifold


(c) Outside of the manifold


(d) Outside of the manifold


(e) Outside of the manifold

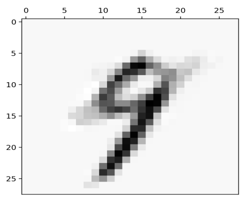
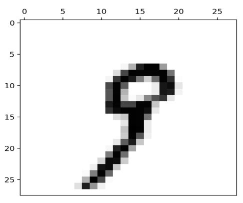
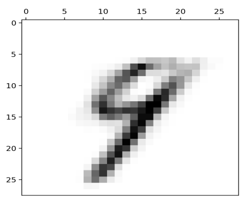
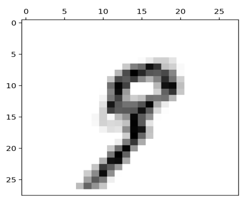
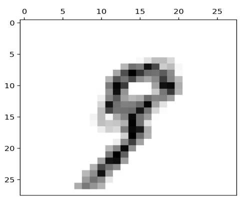

(f) Taken from the manifold

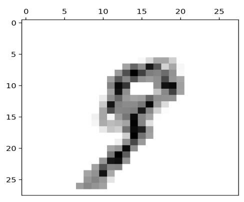
Figure 6: Reconstructed vectors from the 2D embeddings. The first one and the last one are taken from the manifold, other four are on the line between those two (They are hence not on the manifold).


(a) From the dataset


(b) On the line


(c) On the line


(d) On the line


(e) On the line

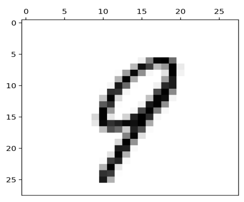

(f) From the dataset

Figure 7: Except the first and last one, others are outside of the original data points, obtained from the line between the first vector and last one. Note that there is no reconstruction here, only results of interpolation is shown.

The Implementation

In the implementation section you give a concise insight to the practical aspects of this coding exercise. It mainly mentions the optimization methods used to solve the model equations. Did you encounter numerical or efficiency problems? If yes, how did you solve them? Provide the link to your git branch of this coding exercise.
Hard limit: One page

Many built-in functions of scipy, numpy and sklearn have been used in the code. The entire code is hugely inspired by the function "sklearn.manifold.locally_linear_embedding".
We may divide the implementation in following three parts:

1. Finding neighbours: For this part the function sklearn.neighbors.NearestNeighbors is used.
2. Finding weights: The error function for a particular x can be rewritten as

$$\epsilon = |x_i - \Sigma_j W_{ij} x_j|^2 = |\Sigma_j W_{ij}(x_i - x_j)|^2 = \Sigma_{jk} W_{ij} W_{ik} C_{jk}$$

   where $C_{jk} = (x_i - x_j)(x_i - x_k)$ is the local covariance matrix. The optimal weights are then given by $W_{ij} = \frac{\Sigma_k C_{jk}^{-1}}{\Sigma_{lm} C_{lm}^{-1}}$. A simple way avoid this inverse operation is to solve the linear system $\Sigma_j C_{jk} W_{ik} = 1$ and then rescale the weights so that they sum to one (This is easily done by the "solve" function in scipy). The only problem that can arise here is when the covariance matrix is singular, and this can be solved by multiplying a very small regularisation value ($reg = 1e-3$) to the trace of $\mathbf{C}$.
3. Last step is to find coordinates in the new low-dimensional space. In the exercise session we have seen that the second error function can be rewritten as

$$\Theta(\mathbf{Y}) = \Sigma_i |y_i - \Sigma_j W_{ij} y_j|^2$$

$$\Theta(\mathbf{Y}) = \mathbf{Y}^\intercal \mathbf{M} \mathbf{Y}$$

   where $\mathbf{M} = (\mathbf{I} - \mathbf{W})^\intercal(\mathbf{I} - \mathbf{W}) = \mathbf{I} - \mathbf{W} - \mathbf{W}^\intercal + \mathbf{W}^\intercal\mathbf{W}$.
   Then by Rayleitz-Ritz theorem the optimal embedding is found by computing the bottom $d+1$ eigenvectors of $\mathbf{M}$ where we drop the first one and keep the d remaining. This is also simply done by the use of function "eigh" of scipy.

The entire code can be found at `https://gitlab.vis.ethz.ch/vwegmayr/slt-coding-exercises/tree/11-801-495/1_locally_linear_embedding`

Your Page

Your page gives you space to include ideas, observations and results which do not fall into the categories provided by us. You can also use it as an appendix to include things which did not have space in the other sections.
No page limit.

As Plots were taking too much place, all the explanations are put here.

MINST dataset is a very large dataset of 60000 pictures of 28×28 dimension. As calculations were very time consuming, I had to reduce the size of dataset to at least 20000. Some other analysis (like finding the most appropriate number of neighbours) has been performed on a dataset of only 3000 pictures. At every part of this question, it is explicitly mentioned whether the dataset of 20000 data points or 3000 point has been used.

Figure 1 shows both 2D and 3D embeddings. In both embeddings we can see a clustering, although not perfect. One strange thing is that the 3D embedding does not use all the space available to make better clusters.
Plotting the $\mathbf{M}$ matrix is extremely hard as it is of 20000×20000 dimension which makes it almost impossible to have a look at it. In order to make it easier to see, only a very tiny fraction of matrix is plotted: $\mathbf{M}_{1:50,1:50}$. Figure 2 shows this small block of the matrix which can be representative of the entire matrix, as it is very sparse, with large values on the secondary diagonal and mostly 0s everywhere else (but not everywhere).
Figure 3 shows the singular values of the $\mathbf{M}$ matrix. If I understood well, the most appropriate number of dimensions to take in this case would be 5, as the singular values are still very small (and thus the objective value of our optimisation stays small) but we get more dimensions to embed the data.

Both effects of number of neighbours and distance metrics have been studied. For this part only 3000 data points are used. First embeddings for $k\ in\ [1, 11, 21, ..., 131, 141]$ have been compared. Visually it could be concluded that $k = 11$ results in clusters that are better separated from each other. Figure 4 compares the embeddings for only few of them (Note that in this case the metric distance is Euclidean, for not any particular reason). One interesting result was that for very big ks (aprox. $k > 20$) it seems that the clustering doesn't change that much and it actually doesn't improve at all. This result could come from the fact that when the number of neighbours are larger compared to the curvature of the original space, then we don't get any good resolution; in other words, the neighbourhood becomes so large that it can't be learned with out linear system of weights anymore, as then we try to fit a linear model for a non linear structure. In a Next step, embeddings for $k\ in\ [8, 9, 10, ..., 19]$ were compared in order to find out whether there is any "optimal" clustering for a specific k. Results was that they were mostly all visually similar regarding how well clusters are separated and thus they are not plotted here any more. That's why for other analysis $k = 10$ is chosen.
In order to see the effect of distance metric, 2D embeddings for distances L1, L2 and Hamming (Intentionally an almost irrelevant distance metric) are compared. Figure 5 shows the result. The result of L1 distance and L2 distance are almost the same (one rotated compared to other, which is actually inherent in solving the $\mathbf{M}$ matrix) and this is simply because both distances have the same nature. We can also see the clusterings resulted by the hamming distance are worse as there are more overlapping points from different clusters. This is also an expected result as the hamming distance has no meaning in the context of image vectors. (page continues...)

An easy way which leads to almost correct reconstructions is to take a vector from the embedding space, find its neighbours in the same space, find the weights that fit the vector as a linear combination of those neighbours, and finally find the original vector by multiplying those weights to the corresponding vectors in the original space. Figure 6 shows the result of reconstruction for two vectors from the embedding space (first one and the last one) and for four other points out of the embedding space (which are on the line that connects those two vectors in the space). We can see that the result is at least visually appealing! Finally figure 7 shows images corresponding to the points on the line between two vectors in the original space just for comparison (the first one and last one are the original vectors, others are the images on the line between those points).