Exercise sheet 4
Representations of data

Due date: 2019-12-12 (2 weeks)
Tasks: 4

In this exercise, you will learn how to represent high-dimensional data with lower-dimensional representations. There is a large number of methods available, and we cannot cover all of them here. The algorithms you will study in this exercise are

1. Principal Component Analysis,
2. Diffusion Maps, and
3. Variational Auto-Encoders.

Important: For each of the tasks 1–3, in addition to the usual description of your findings, you have to report

(a) an estimate on how long it took you to implement and test the method,

(b) how accurate you could represent the data and what measure of accuracy you used, and

(c) what **you** learned about both the dataset and the method (which is probably different from what the machine learned).

The mathematical theory underlying most of the algorithms discussed here is mostly studied in the field of differential geometry. A founding father of the field is Carl-Friedrich Gauss, the Princeps mathematicorum (Latin for “the foremost of mathematicians”). While working on a geodetic survey of the Kingdom of Hanover, he discovered the Theorema Egregium (“remarkable theorem”, [4, 5])—with the corollary that it is impossible to create a (flat) map for any part of a sphere that does not distort length and angles (in his case, the Kingdom of Hanover). One of the most important concepts for differential geometry, and in also for the representation of data, is a *manifold* [7]. It will be very difficult for you to understand certain restrictions on the representation of data if you do not understand this basic notion[1].

1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most fundamental techniques for data representation and manifold learning. It linearly decomposes the data, and works under the assumption that the data points are samples of a multi-variate normal distribution. More generally, PCA works well if the given dataset is close to a hyperplane, that is, a plane with arbitrary dimension (a line, a two-dimensional plane, a cube, etc.).

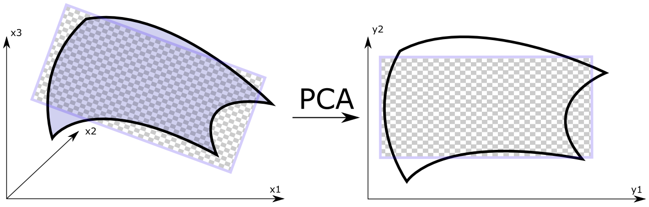

Figure 1: An embedding of a manifold in two-dimensional Euclidean space, using Principal Component Analysis. The manifold on the left is already embedded into three-dimensional Euclidean space, but PCA is able to find a two-dimensional embedding, because the manifold is almost planar.

As PCA is such a fundamental idea, it has been discovered in many different areas and hence has many different variations and names: Proper Orthogonal Decomposition (POD) in mechanical engineering, Karhunen-Loéve-Transformation in signal processing, factor analysis in statistics, and many more[2] The work horse of the

[1] You can take a look at my informal introduction on my website.
[2] Take a look at the wikipedia article `https://en.wikipedia.org/wiki/Principal_component_analysis`.

PCA algorithm is an eigendecomposition of the (properly modified) data matrix. In its most basic form, the PCA algorithm to decompose a data set $\{x_i \in \mathbb{R}^n\}_{i=1}^N$ into its principal components can be written down as follows:

1. Form the data matrix $X \in \mathbb{R}^{N \times n}$ with rows x_i from points (observations) in the data set.
2. Center the matrix by removing the data mean $\overline{x} = \frac{1}{N}\sum_{i=1}^N x_i$ from every row (every data point):
$$\overline{X}_{ij} = X_{ij} - \overline{x}_j.$$
3. Decompose the centered data matrix into singular vectors U, V and values S, such that
$$\overline{X} = USV^T,$$
where $U \in \mathbb{R}^{N \times N}$, $S \in \mathbb{R}^{N \times n}$, and $V \in \mathbb{R}^{n \times n}$. Assuming the singular values on the diagonal of S are in decending order, the columns of U contain the coordinates of the principal components of each data point with decreasing importance. These vectors are normalized to length one. The matrix V is orthonormal ($V^T V = I$) and can be used to project new, centered data points $\hat{X}$ onto their (properly scaled) principal components $\hat{U}S$, because
$$\hat{X}V = (\hat{U}SV^T)V = (\hat{U}S)(V^T V) = \hat{U}S. \tag{1}$$
Note that only the coordinates U change to $\hat{U}$ for the new data points, the lengths S and direction V of the principal components are not supposed to change.
4. The "energy" (explained variance) of the i-th principal component is contained in the singular value σ_i on the diagonal of the matrix S. The percentage of the total energy explained by using a certain number L of principal components to describe the data can be computed through
$$\frac{1}{\text{trace}(S)}\sum_{i=1}^{L}\sigma_i,$$
where trace(S) is the sum over all singular values (not just L).

2 Diffusion Maps

Diffusion Maps are a family of functions that map data points into the set of eigenfunctions of the Laplace-Beltrami operator on a manifold describing the data. The theoretical foundation of the algorithm and the Laplace-Beltrami operator can be found in the literature [6, 2, 3, 8]. From an abstract point of view, Diffusion Maps are not so different from Principal Component Analysis (that is also why they have similarity with "kernel-PCA"). The main idea in Diffusion Maps is to represent each data point not in its given coordinates, but in coordinates of basis functions of the function space on the data. In that function space, an eigendecomposition is used to find the optimal basis, as in PCA. The basis functions extracted in Diffusion Maps are the eigenfunctions $\phi_k : M \to \mathbb{R}$ of the Laplace-Beltrami operator Δ on a manifold M close to the data, so that

$$\Delta\phi_k = \lambda_k \phi_k, k \in \mathbb{N}.$$

A critical part of the Diffusion Map framework is to algorithmically remove the influence of the sampling density of the data points from the estimation of the optimal basis. This is typically not done in PCA, although some versions (robust PCA, outlier detection) try to accomplish a similar task.

There are several different versions of the Diffusion Maps algorithm, we focus on the description in [1]. The following adaption is a good start:

1. Form a distance matrix D with entries
$$D_{ij} = \|y_i - y_j\|,$$
where $i = 1, \dots, N$ are the rows, $j = 1, \dots, N$ are the columns, and y_i, y_j are the i-th and j-th data points (rows of the data matrix X).
2. Set ϵ to 5% of the diameter of the dataset: $\epsilon = 0.05(\max_{i,j} D_{i,j})$.

3. Form the kernel matrix W with $W_{ij} = \exp\left(-D_{ij}^2/\epsilon\right)$.
4. Form the diagonal normalization matrix $P_{ii} = \sum_{j=1}^{N} W_{ij}$.
5. Normalize W to form the kernel matrix $K = P^{-1}WP^{-1}$.
6. Form the diagonal normalization matrix $Q_{ii} = \sum_{j=1}^{N} K_{ij}$.
7. Form the symmetric matrix $\hat{T} = Q^{-1/2}KQ^{-1/2}$.
8. Find the $L+1$ largest eigenvalues a_l and associated eigenvectors v_l of $\hat{T}$.
9. Compute the eigenvalues of $\hat{T}^{1/\epsilon}$ by $\lambda_l^2 = a_l^{1/\epsilon}$.
10. Compute the eigenvectors of the matrix $T = Q^{-1}K$ by $\phi_l = Q^{-1/2}\phi_l$.

The eigenvalues λ and the eigenvectors ϕ_l are the objects we are interested in. Note that the eigenvector ϕ_0 is constant if the data set is connected for the given value of ϵ, so only the eigenvectors $\phi_1, \phi_2, \ldots$ are of interest.

Figure 2 shows a data set with points on the unit sphere. The eigenfunctions of the Laplace-Beltrami operator on the unit sphere have explicit formulas, but here, they are computed with the Diffusion Map algorithm. The right side of the figure shows the first few, evaluated over the data set.

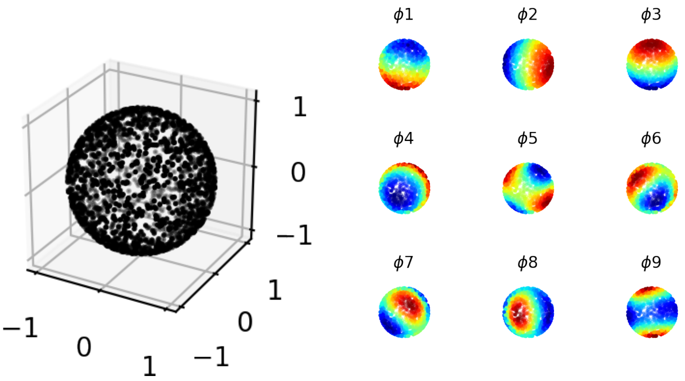

Figure 2: Left: a sphere, embedded in three-dimensional Euclidean space and sampled at the black dots. Right: Eigenfunctions of the Laplace-Beltrami operator, evaluated at the positions of the black dots on the left. The color shows the function value.

3 Variational Auto-Encoder

Variational autoencoders (VAEs[3]) are a class of deep generative models. They can learn a latent representation and the underlying distribution of our data in an unsupervised fashion. We recommend to use Python and TensorFlow 1.15 to implement a VAE. The advantage of TensorFlow is that only the objective function

$$\mathcal{L}_{\text{ELBO}}(\theta, \phi) = \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}\Big[\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big] - \mathrm{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\big)\Big]$$

needs to be implemented. The optimization—including computing the gradients, applying the reparametrisation, etc.—is handled by TensorFlow. A short tutorial can be found in the link below[4]. You are free to use other machine learning frameworks, for example PyTorch.

Note: the number of points per exercise is a rough estimate of how much time you should spend on each task.

[3] VAE paper: `https://arxiv.org/abs/1312.6114`

[4] Tutorial: `https://danijar.com/building-variational-auto-encoders-in-tensorflow`

Task 1/4: Principal component analysis Points: 30/100

In the **first part** of this task, you are supposed to implement principal component analysis using a library method for singular value decomposition. Then, approximate the one-dimensional, linear subspace of the two-dimensional data set `pca_dataset.txt` on Moodle that is optimal in the sense of variance reduction. How much energy is contained in each of the two components? Plot the data set as shown in the figure, and add the direction of the two principal components by drawing lines starting from the center of the data set.

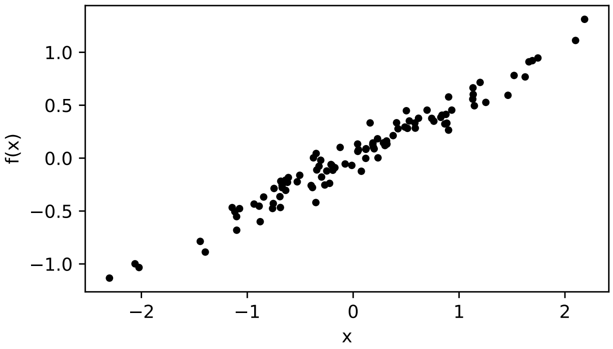

Figure 3: Dataset to analyze with PCA.

For the **second part**, apply PCA to the image below. You can download the image on its original website[5], or generate it through the `scipy.misc.face()` command in Python. Note that this image is the spiritual successor to "Lena", which was removed due to copyright issues. The 249 columns (or rows, you choose) of the image should be considered as the data points in the analysis.

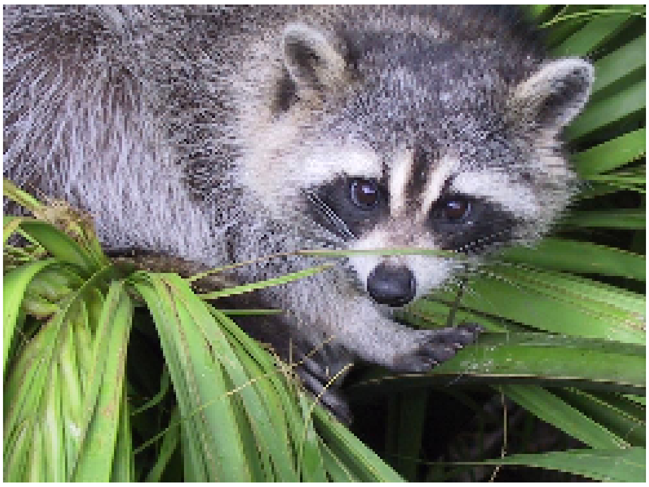

Figure 4: Image of a racoon.

Visualize reconstructions of the image with (a) all, (b) 120, (c) 50 and (d) 10 principal components. At what number is the information loss visible? To obtain a reconstruction with a certain number L of principal components, just set the singular values to zero that are smaller than the L-th singular value and then reconstruct using $\hat{X} = U\hat{S}V^T$. At what number is the "energy" lost through truncation smaller than 1%?

The **third and last part** of this task concerns the trajectory data in the file `data_DMAP_PCA_vadere.txt` on Moodle. The file contains position data of 15 pedestrians over 1000 time steps (in the form $x_1, y_1, x_2, y_2, \ldots$ for the x, y positions of all pedestrians).

1. Visualize the path of the first two pedestrians in space. What do you observe?

[5] Public domain, `https://pixnio.com/fauna-animals/raccoons/raccoon-procyon-lotor`.

2. Analyze the data set by projecting the 30-dimensional data points (one in each row of the file) to the first two principal components. Discuss your findings, in particular: are two components enough to capture most of the energy of the data set? Why, or why not? How many do you need to capture most of the energy?

Task 2/4: Diffusion Maps **Points: 30/100**

I expect you to implement the Diffusion Map algorithm yourself, in the language of your choice, using only basic library support. You do not need to implement eigensolvers, matrix multiplication routines, etc., but you are also not supposed to use existing algorithms for Diffusion Maps.

Part one: Use the algorithm to demonstrate the similarity of Diffusion Maps and Fourier analysis. To do this, compute five eigenfunctions ϕ_l associated to the largest eigenvalues λ_l with Diffusion Maps, on a periodic data set with $N = 1000$ points given by

$$X = \{x_k \in \mathbb{R}^2\}_{k=1}^N,\ x_k = (\cos(t_k), \sin(t_k)),\ t_k = (2\pi k)/(N+1). \tag{2}$$

Plot the values of the eigenfunctions $\phi_l(x_k)$ against t_k. Bonus (5 points): explain the relation of your results to Fourier analysis of the "signal" X.

Part two: Use the algorithm to obtain the first ten eigenfunctions of the Laplace Beltrami operator on the "swiss roll" manifold, defined through

$$X = \{x_k \in \mathbb{R}^3\}_{k=1}^N,\ x_k = (u\cos(u), v, u\sin(u)), \tag{3}$$

where $(u, v) \in [0, 10]^2$ are chosen uniformly at random. The `sklearn` Python library has a simple routine to generate the swiss-roll data set[6]. Use this method to create the data points in three-dimensional space, with no additional noise. Use the Diffusion Map algorithm to obtain approximations of the eigenfunctions, and plot the first non-constant eigenfunction ϕ_1 against the other eigenfunctions in 2D plots (the function ϕ_1 on the horizontal axis). At what value of l is ϕ_l, $l > 1$ no longer a function of ϕ_1? Compute the three principal components of the swiss-roll dataset. Why is it impossible to only use two principal components to represent the data?

Part three of this task again concerns the trajectory data in the file `data_DMAP_PCA_vadere.txt` on Moodle. Perform the same analysis you did with PCA. **Note:** Diffusion Maps do not have exactly the same energy interpretation of the eigenvalues (singular values) than PCA, so you cannot make statements about it here. How many eigenfunctions do you need to accurately represent the data set?

Task 3/4: Training a Variational Autoencoder on MNIST **Points: 30/100**

In this exercise, we want you to implement and to train a VAE.

Dataset

Use the MNIST dataset. You can download it using tensorflow, using

```
tensorflow.keras.datasets.mnist.load_data()
```

The tutorial to VAE linked above also describes how to obtain the data. **Note:** do not binarise MNIST but normalise between 0 and 1, and split it into a training and a test set.

Model

Use a two-dimensional latent space. Use a multivariate diagonal Gaussian distribution as approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$. The encoder neural network, which outputs the mean and standard deviation of $q_\phi(\mathbf{z}|\mathbf{x})$, should consist of 2 hidden layers (dense/fully connected) with 256 units each and ReLU activation functions. Use also a multivariate diagonal Gaussian distribution as likelihood $p_\theta(\mathbf{x}|\mathbf{z})$. The decoder neural network, which outputs **only** the mean of $p_\theta(\mathbf{x}|\mathbf{z})$, should consist of 2 hidden layers (dense/fully connected) with 256 units each and ReLU activation functions. Implement the standard deviation as a trainable global variable. Use a multivariate diagonal standard normal distribution as prior $p(\mathbf{z})$. **Note:** using a Bernoulli likelihood, as it is the case in most tutorials, is only possible for binarised versions of MNIST.

Optimisation

[6] See `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html`.

Use the Adam optimiser with a learning rate of 0.001. Use a batch size of 128 for training. **Note:** when training VAEs, there are—in addition to the loss—typically three sources of information, which indicate whether the model trains properly: the latent representation, reconstructed data, and generated data. You will get points for this task if you answer the questions and document the experiments.

1. What activation functions should be used for the mean and standard deviation of the approximate posterior and the likelihood—and why?
2. What might be the reason if we obtain good reconstructed but bad generated digits?
3. Train the VAE (training set) and do the following experiments (test set) after the 1st epoch, the 5th epoch, the 25th epoch, the 50th epoch, and after the optimisation converged:
   (a) Plot the latent representation, i.e., encode the test set and mark the different classes.
   (b) Plot 15 reconstructed digits and the corresponding original ones.
   (c) Plot 15 generated digits, i.e., decode 15 samples from the prior.
4. Plot the loss curve (test set), i.e., epoch vs. $-\mathcal{L}_{\text{ELBO}}$.
5. Train the VAE using a 32-dimensional latent space and do the following experiments after the optimisation converged:
   (a) Compare 15 generated digits with the results in 3.2
   (b) Compare the loss curve with the one in 4.

Task 4/4: Fire Evacuation Planning for the MI Building Points: 10/100

Due to the increasing amount of students at the Technical University of Munich, the fire evacuation plan for the MI building needs to be reconsidered. An important information is the distribution of people within the MI building $p(\mathbf{x})$.

In a hypothetical scenario, the fire department decided to track 100 random students and employees during the busiest hour on different days. The idea is to use this data for learning $p(\mathbf{x})$. As a first experiment, the fire department wants to estimate the number of people that is critical for the building. To simplify the task, it defined a sensitive area in front of the main entrance—marked by the orange rectangle (130/70 & 150/50)—where the number of people should not exceed 100. You will get points for this task if you document the experiments.

1. Download the FireEvac dataset (training: 3000 x-y-positions; testing: 600 x-y-positions) and make a scatter plot to visualise it.
2. Train a VAE on the FireEvac data to learn $p(\mathbf{x})$. You can reuse your VAE implementation from the previous task. **Note:** the observation space for this dataset is two-dimensional (instead 784 in MNIST). Although it is a low-dimensional dataset, it is not a simple one. It might be that you will need to train the VAE for quite a few epochs or to tune hyperparameters.
3. Make a scatter plot of the reconstructed test set.
4. Make a scatter plot of 1000 generated samples.
5. Generate data to estimate the critical number of people for the MI building: how many samples (people) are approximately needed to exceed the critical number at the main entrance?

Bonus points (5):
Later the fire department wants to use $p(\mathbf{x})$ as the initial distribution for a precise crowd simulation.

1. Create a new Vadere scenario for the MI building with the appropriate dimensions, some of the walls (similar to the figure) and a target in the top right corner.
2. Use $p(\mathbf{x})$ to sample the positions of 100 people and simulate their trajectories.
3. Report what you did and discuss the results.

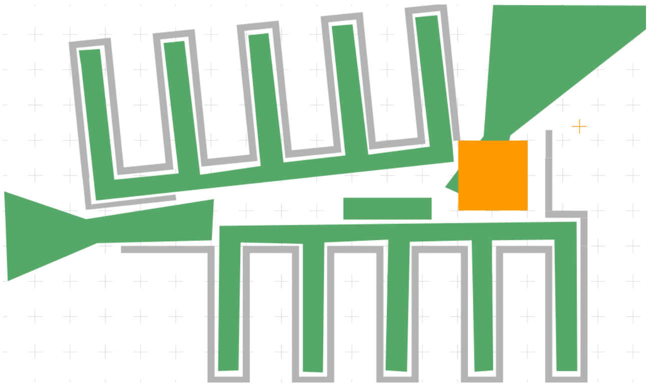

Figure 5: Schematic of the MI building in Garching, including the areas where pedestrians can be located (green) and are measured (orange).

References

[1] T. Berry, J. R. Cressman, Z. Gregurić-Ferenĉek, and T. Sauer. Time-scale separation from diffusion-mapped delay coordinates. *SIAM Journal on Applied Dynamical Systems*, 12(2):618–649, 1 2013.

[2] Ronald R. Coifman, Stćphane Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21):7426–7431, 2005.

[3] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 7 2006.

[4] Carl Friedrich Gauss. *Disquisitiones generales circa superficies curvas.* Royal Society of Göttingen, 1827.

[5] Carl Friedrich Gauss. *General Investigations of Curved Surfaces.* Princeton University Library, 1902.

[6] Stéphane S. Lafon. *Diffusion Maps and Geometric Harmonics.* PhD thesis, Yale University, 2004.

[7] John M. Lee. *Introduction to Smooth Manifolds.* Springer New York, 2012.

[8] Steven Rosenberg. *The Laplacian on a Riemannian Manifold.* Cambridge University Press, 1997.