Report for exercise 4 from group K

Tasks addressed:	4
Authors:	SONJA KRAFFT (03681252) LUDWIG-FERDINAND STUMPP (03736583) TIMUR WOHLLEBER (03742932)
Last compiled:	2023–10–27
Source code:	https://gitlab.lrz.de/00000000014A9334/mlcs-ex4-representation-of-data

The work on tasks was divided in the following way:

SONJA KRAFFT (03681252)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
LUDWIG-FERDINAND STUMPP (03736583)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
TIMUR WOHLLEBER (03742932)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%

1 Introduction

Goal of this exercise

Attempting to formalise and quantitatively describe the complexity of the real world often poses the challenge of analysing and interpreting large amounts of data. Not only is the analysis of these large amounts of data limited in terms of computing power, but high-dimensional data is often difficult for humans to interpret, as interrelationships are difficult to make visually tangible. For this reason, data science has long been concerned with the challenge how to transfer high-dimensional data sets into a space in which they have fewer features, while at the same time containing almost the same amount of information. One of the best known approaches to dimension reduction is Principal Component Analysis (PCA), which tries to approximate the best linear subspace. In this exercise, this will be carried out on various data sets. However sometimes, these lower-dimensional manifolds cannot be uncovered using a linear algorithm such as PCA. Therefore a diffusion map framework will be presented in the second part, which also works with non-linear relations. In the last part, the approach of the Variational Auto-Encoder (VAE) will be presented, which can enable the generation of data in addition to the reduction of the dimension.

Python dependencies

Due to the great selection of publicly available libraries, Python is chosen as the programming language for this exercise. The following is a listing of required software and *Python* packages in order to run and maintain the code. For plotting, we decided to use *matplotlib.pyplot*[1], since this seems to be the library to take for especially 3D plots. *SciPy*[2] and *NumPy*[3] are used for the core mathematical operations, where especially eigensolvers of *SciPy* are of great value for the implementation of the diffusion map algorithm. We use *Tensorflow*[4] and *Keras*[5] for the implementation of the Variational Autoencoder used in task three and four, as well as *scikit-learn*[6] for the application of PCA in task two.

python	3.9.5
vscode	1.63.2

Table 1: Python version and code editor.

datafold	1.1.6
matplotlib	3.4.3
numpy	1.21.3
pandas	1.2.5
scikit-image	0.19.1
scikit-learn	1.0.1
scipy	1.7.1
tensorflow	2.7.0
keras	2.7.0

Table 2: List of required Python packages to run the code of the repository. Find the corresponding `requirements.txt` file inside the root of the directory.

[1] `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html`
[2] `https://scipy.org/`
[3] `https://numpy.org/`
[4] `https://www.tensorflow.org/`
[5] `https://keras.io/`
[6] `https://scikit-learn.org/stable/`

2 Individual Tasks

Report on task 1/4: Principal Component Analysis

Notebook: `task_1.ipynb`
Code: `helpers/data.py`, `helpers/pca.py`, `helpers/plots.py`

Motivation

A variety of real-life scenarios and applications generate high-dimensional data such as trajectories of human crowds or images. However, in most applications, these data sets can be reduced to a much smaller dimension, taking into account an assumed loss of information. A well-known algorithm for this is principal component analysis. The relevance of this algorithm can be seen in how many different programming libraries it is implemented in. In the following section, a simple implementation of principal component analysis (PCA) using singular value decomposition (SVD) will be demonstrated.

Part I - PCA Dataset Analysis

For the decomposition of the dataset and the implementation, the SVD decomposition is applied to the previously centred dataset. Two processes are of particular importance:

Dimensionality Reduction

The centered datamatrix X is decomposed into $\bar{X} = USV^T$. The matrix S contains the singular values σ_i on the diagonal in decreasing order. The matrix V is an orthonomal matrix containing the direction of the principal components. The columns of matrix U contain the coordinates of the principal components. The dimensionality reduction can be achieved by only considering a certain number L of principal components and setting the remaining singular values to zero. With this modified singular value matrix the dimensional reduced matrix can reconstructed. The projection of the high dimensional data to a space which expresses the data with lower dimensions is called linear subspace.

Transformation

Sometimes it is also useful to represent the data in a coordinate frame which expresses the data in a simpler form. The base vectors of such a coordinate frame can be found by considering the direction of principal components in V. The transformation $\hat{X}$ of the dataset X can be achieved by $\hat{X} = XV$ or $\hat{X} = \hat{U}S$. Not explicitly required in this part of the task, but an example transformation using the PCA dataset is shown in Figure 1

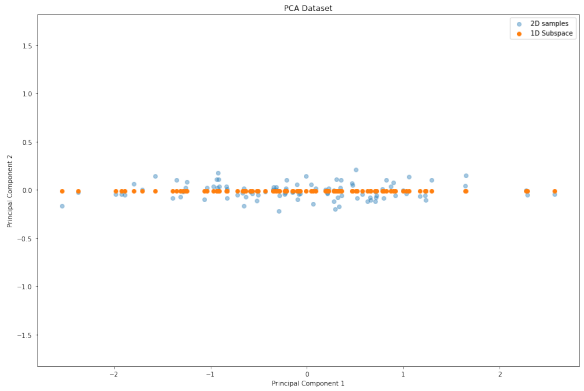


Figure 1: Shows an example of the transformed PCA Dataset. The new coordinate system is found by the direction of the two principle components.

PCA Dataset

Figure 2 shows the reconstruction of the provided two dimensional PCA dataset. The graph illustrates that the points gather around a diagonal from the bottom left to the top right. Seeing the plot one can deduce that the variance seems to be significantly greater in one direction than in the other. This becomes clearer when the two principal components are also displayed. Please note that the matrix V contains the normalised directions of the principal components, so they normally have the same length. In order to emphasise the significance of the respective direction, these vectors were multiplied by a constant that depends on the variance, resulting in the different lengths. So it can be clearly seen that the dominant principal component is the direction of the green line. This can also be seen in the, explained variance ("energy") for both components, where the first component has $E_1 = 0.9931$ and $E_2 = 0.0069$. Please note that the concept behind "explained variance" is covered in more detail in the accuracy measurement in Section 2.

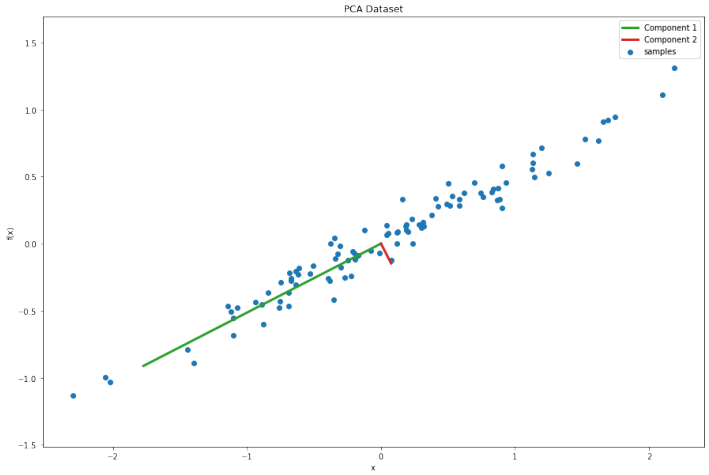


Figure 2: Scatter Plot of the two dimensional PCA Dataset with 100 samples illustrated in blue. Additionally the direction of the scaled principle components is illustrated with different colors. The length of the terse line indicates the explained variance of the corresponding component.

The reduction of dimension explained in the introduction of this chapter can also be approximated for the 2D dataset by projecting the data points in the direction of the strongest principal component. Figure 3 indicates the one dimensional linear subspace with an orange line.

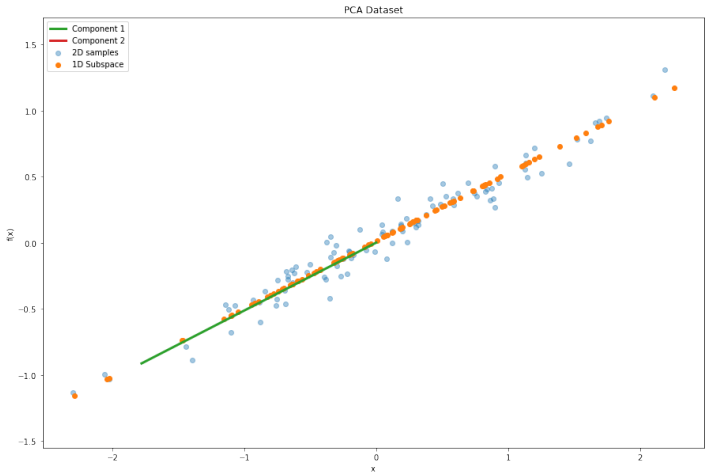


Figure 3: Approximation of the one dimensionel linear subspace in the direction of the largest principle component. Since the second principal value is set to zero the red vector is no longer visible.

Part II - Image Dataset Analysis

An other example of high-dimensional data can be found in pictures. For this purpose, an example photo of an racoon is to be converted into a grey image and scaled to the size of (249 x 185). The image is then to be reconstructed using the previously explained SVD-based PCA algorithm. For this purpose, four different numbers of components are to be plotted and evaluated. The reconstructions resulting from this are shown in Figure 4. If one only take a rough look at the images, one will notice that the images for the cases n=185 and n=120 (n=number of used principle components for reconstruction) look almost identical although 65 components less were used for the reconstruction. Only in the image with n=50, for example, slight streaks can be seen in the black background (blur). For the case with n=10 components, the stripes and the blurring become even clearer. What is remarkable about this case is that already n=10 components are sufficient to represent the basic features of the image. This and the fact that there are hardly any differences in the reconstruction between n=120 and n=185 components suggest that a few components provide the greatest amount of information.

Figure 4: Shows the reconstruction results of the four given principle components. The upper left is reconstructed using all n=185 principal components, the upper right uses only n=120. In the bottom are the cases shown for n=50 on the left and n=10 on the right side.

In order to take a closer look at this phenomenon, the information content per component is to be determined below. For this purpose, the information content that would be lost for each individual component was represented in a bar chart. this is shown in Figure 5. This shows that most of the 185 components of the image contribute little image information. You can see that the first n=25 components are responsible for most of the explained variance ("energy").

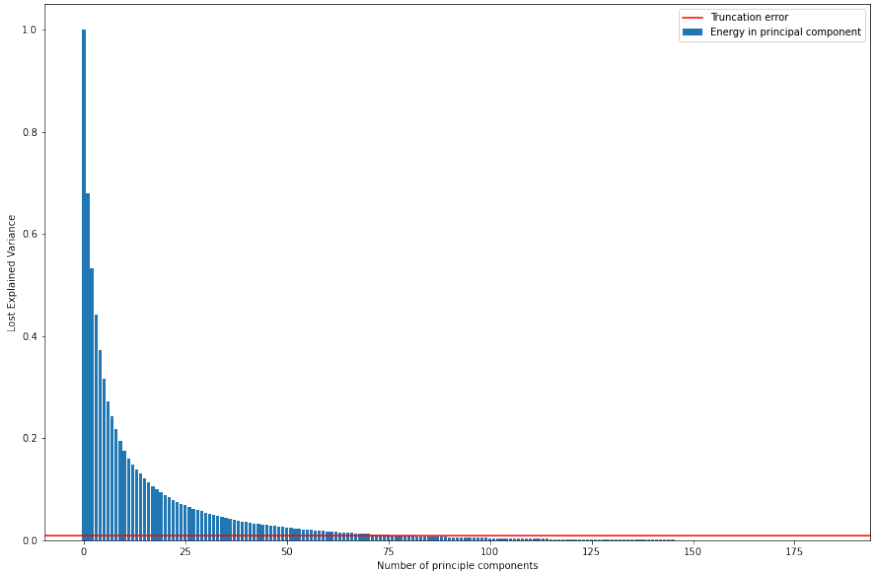


Figure 5: Bar chart over all n=185 principle components and their corresponding contribution to the information content of the image or loss in case of omission. The red line indicates the threshold from which the information content (truncation error during the reconstruction) is less than 1%

This becomes even clearer when you draw in the threshold value of 1% explained variance into the plot (highlighted by the red horizontal line) and zoom in. From the n=77 component onwards one gain no more than 1% information content (energy) or lose it if one would omit it.

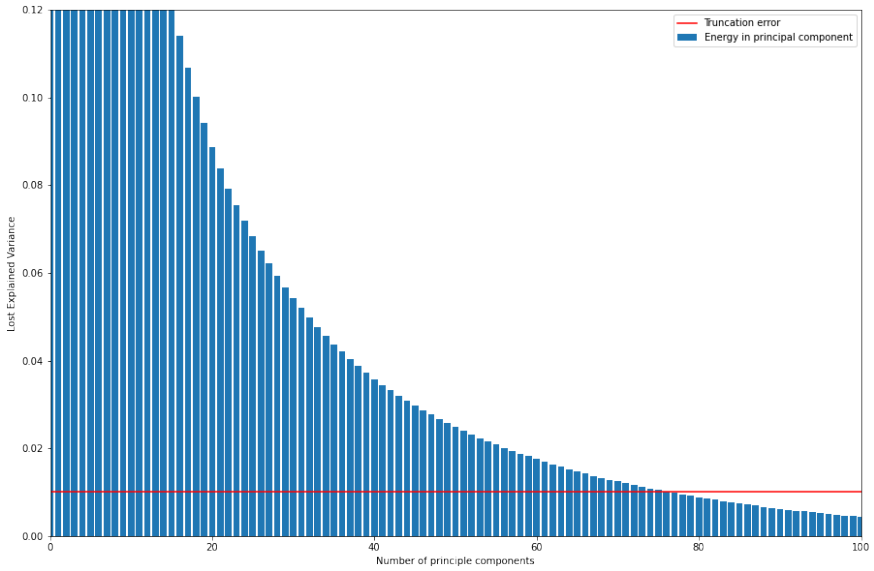


Figure 6: Detail breakout of bar chart in Figure 5

The experiment has thus shown that it is possible to compress images significantly, since only a few components contribute information.

Part III - DMAP Dataset Analysis

As in the previous section, this part will also deal with high-dimensional data. For this purpose, the implemented PCA algorithm will be used to analyse trajectory data of 15 pedestrians over 1000 time steps. The visualisation of the distance covered for the first two pedestrians can be seen in Figure 7.

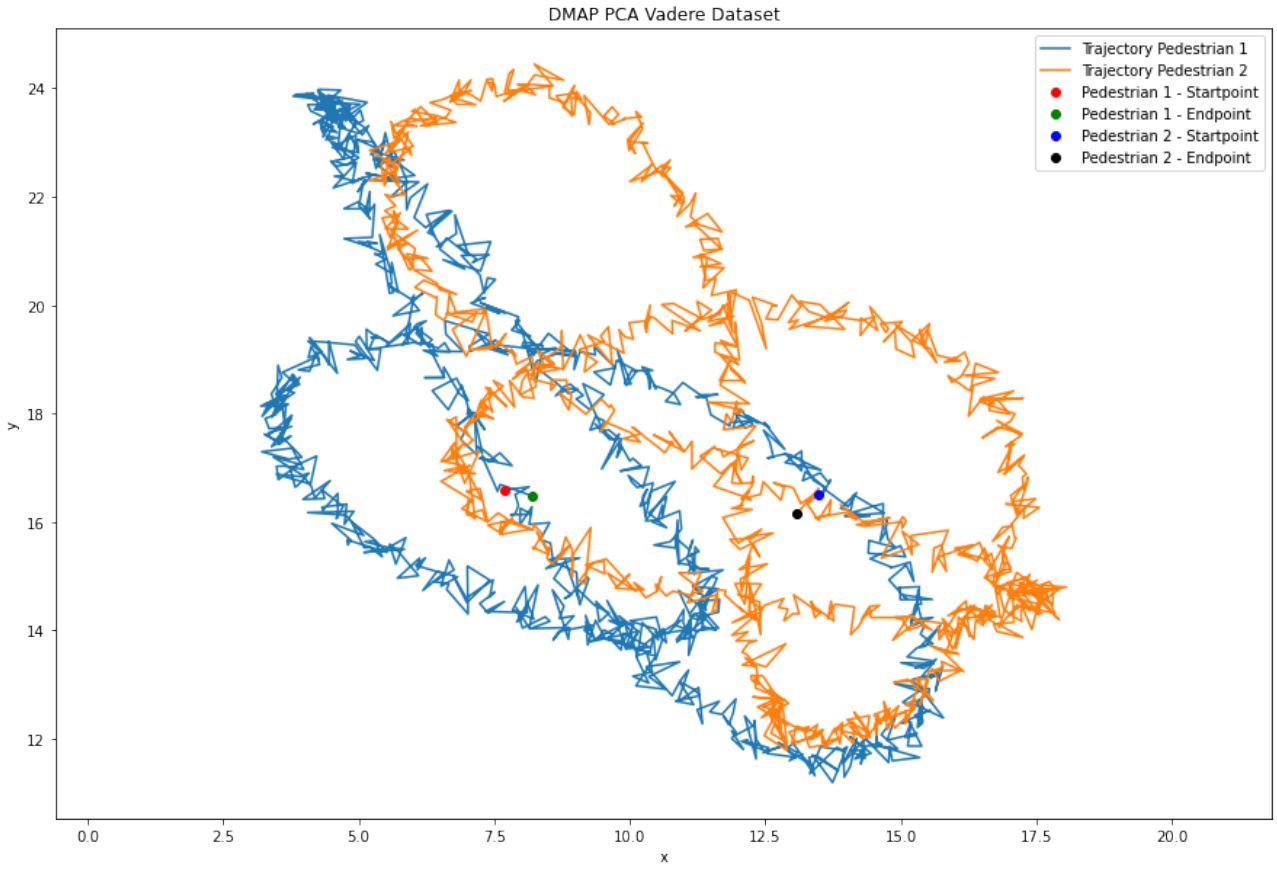


Figure 7: This figure shows the trajectory of two pedestrians in two dimensional space. The trajectory of the individual pedestrians has been highlighted in blue and orange. Furthermore, the respective start and end points of the trajectories are marked in colour.

It can be observed that the two pedestrians run different routes that cross at certain points. Further the two pedestrian trajectories exhibit a zigzag pattern across all time steps. The common feature of the two trajectories seems to be that the two pedestrians run in a loop, this can be deduced from the last time step being next to the corresponding initial position. The trajectory of the first pedestrian encoding also the timestamp as colors is shown in Figure 14b. This helps to understand recognising the direction of movement.

Dimensionality Reduction

In order to understand the effect of dimension reduction on trajectory dataset in more detail, the following section examines the linear subspaces of the 30-dimensional *DMAP PCA Vadere* dataset for n=2 and n=3 principal components. For the case with two principal components, the energy analysis results in $E_1 = 0.47331$ for the first principal component and $E_2 = 0.37594$. The overall energy of the two components is therefore $E_{12} = 0.84925$. For the use case it was defined that 90% of the explained variance ("energy") must be achieved. A comparison with the determined value and the definition thus shows that two components are not sufficient to achieve "most of the energy". The graphical representation of the two-dimensional subset is shown in Figure 8. *Please note: the energy values in this report are rounded to the fifth decimal digit.The exact values can be found in the notebook.*

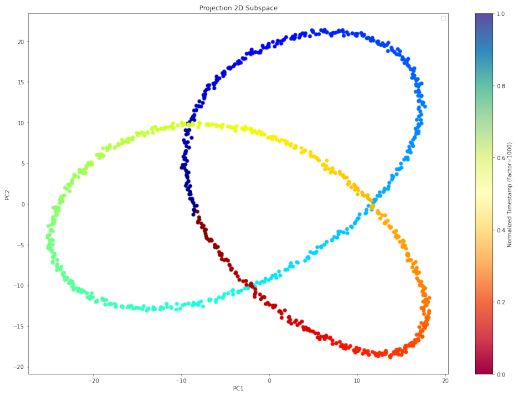


Figure 8: 2D Subspace projection with two principal components

The presence or absence of intersections does not indicate whether the minimum dimension has been found and whether the number of principal components used is sufficient to capture essential information of the data set as it might happen that also components with little explained variance hold valuable information. In order to find a representation that captures most of the energy the nearest approach is now to extend the dimension of the subspace by one to n=3 principal components. Graphically, this leads to Figure 9.

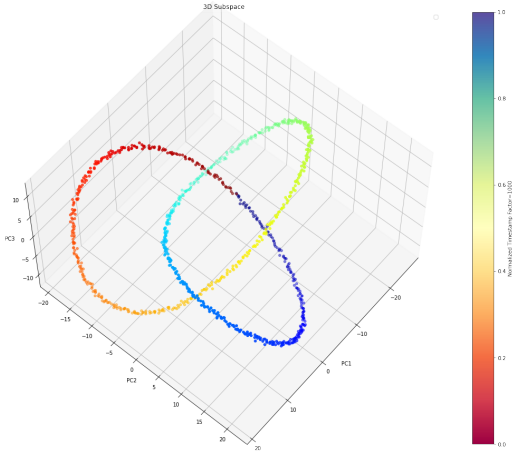


Figure 9: 3D Subspace projection with three principal components

For the case with three principal components, the energy analysis results in $E_1 = 0.47331$ for the first principal component, $E_2 = 0.37594$ for the second principal component and $E_3 = 0.14788$ for the third principal component. The overall energy of the three components is therefore $E_{123} = 0.99713$. A further comparison with the definition value of 90% shows that sufficient energy can be recovered with 3 principal components.

General questions

In this section, the approach, methodology and lessons learned will be briefly discussed.

Time estimate it took to implement code for this part

The development of the first functioning methods was possibly more quickly in this task than in the following tasks ($\sim$ 4 hours). However, this presupposes that one is already familiar with the two essential algorithms (SVD and PCA) and the corresponding processes such as the transformations, projections and reduction of the dimensions. In concrete terms, this resulted in another 4 hours to refresh and acquire knowledge. There were also another 4 hours to modularise and clean up the code which sums it up to 12 hours in total.

Accuracy of the method and evaluation criteria

In order to evaluate how meaningful the reconstructed data is despite the reduction of the actual dimensions, the explained variance ("energy") was used in this section. The energy can be calculated using the following formula:

$$\frac{1}{trace(S^2)} \sum_{i=1}^{L} \sigma_i^2 \tag{1}$$

In this formula $trace(S^2)$ is the sum over all the squared singular values and σ_i the corresponding singular values. In order to determine the number of necessary principle components for a required significance, the energy seems to be an a first indication of an assessment criterion, but not sufficient, as it can happen that components with little energy contain the information that is important for the use case.

Another evaluation criteria that can be used with such common algorithms is the comparison with existing programming libraries. For this purpose, the own implementation was compared with the PCA algorithm from sklearn. [7]

Learning outcomes about dataset and applied method

- applying SVD to a centered dataset allows a simple implementation of a PCA algorithm, but for practical use the own implementation is often not worthwhile, because it is not numerically optimized and some common procedures like energy analysis, projection or transformation are time and labor consuming to implement.
- high-dimensional data can often be projected onto a linear subspace. This approximation only needs a small number of essential principal components to capture most of the energy. Nevertheless this approach works only good for independent features, it is not really possible to capture non-linear correlations
- graphical representations of the linear subspaces are often not (directly) interpretable
- explained variance (energy) is a first indication criterion, but not sufficient, as it can happen that components with little energy contain the information that would be meaningful
- if images are interpreted as high dimensional data, even a few principal components are sufficient for a reconstruction of essential contents. Further, a large proportion of the principle components only provide a very small amount of information.

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`

Report on task 2/4: Diffusion Maps

Notebook: `task_2.ipynb`
Code: `helpers/data.py`, `helpers/diffusion_maps.py`, `helpers/plots.py`

Motivation

Whereas linear manifold learning techniques such as *Principle Component Analysis* (PCA) or *Singular Value Decomposition* (SVD) are known tools for every Data Scientist or Machine Learning Engineer, they reach their boundaries when it comes to uncovering non-linear manifolds. In the following part, we are going to take a closer look at one non-linear manifold learning technique, namely *Diffusion Maps*. As for PCA, Diffusion Maps is a technique to reduce the dimensionality of a dataset and project it into a manifold space where its topology is preserved. The concept of topology is directly built into the Diffusion Maps algorithm by incorporating the distance matrix on the dataset.

Introduction to Diffusion Maps algorithm used in this exercise

For this exercise, we are using an implementation proposed by Berry et al.[1, p.647] which was already given in the formulation sheet of this exercise. While there exists multiple approaches on how to implement a Diffusion Map algorithm, this implementation has the great advantage that it tries to be robust against the sampling density of the given dataset. Below list is copied from the exercise sheet and should serve as a reference to the reader and the afterwards shown results:

1. Form a distance matrix D with entries
$$D_{ij} = ||\mathbf{y}_i - \mathbf{y}_j||$$
where $i = 1, ..., N$ are the rows, $j = 1, ..., N$ are the columns, and $\mathbf{y}_i$, $\mathbf{y}_j$ 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: $\epsilon = 0.05(max_{ij} D_{ij})$.
3. Form the kernel matrix W with $W_{ij} = \exp(-D_{ij}^2/\epsilon)$.
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 ϕ_l of the matrix $T = Q^{-1}K$ by $\phi_l = Q^{-1/2}v_l$.

This algorithm is used and converted in a step-by-step approach in Python with the help of the libraries *NumPy*[8] and *SciPy*[9]. Here is a list of some implementation details, always in reference to the followed algorithm (2):

- Step 1 - distance matrix: *scipy.spatial.distance_matrix*.
- Step 4 & 6 - normalization: *np.diag(np.sum(kernel_matrix, axis=1))*.
- Step 7 & 10 - fractional matrix power: *scipy.linalg.fractional_matrix_power*.
- Step 8 - eigendecomposition: Since the matrix $\hat{T}$ is real symmetric by construction, one can use *scipy.sparse.linalg.eigsh* to compute only the $L+1$ eigenvectors corresponding to the largest eigenvalues.

Similarity of Diffusion Maps and Fourier analysis

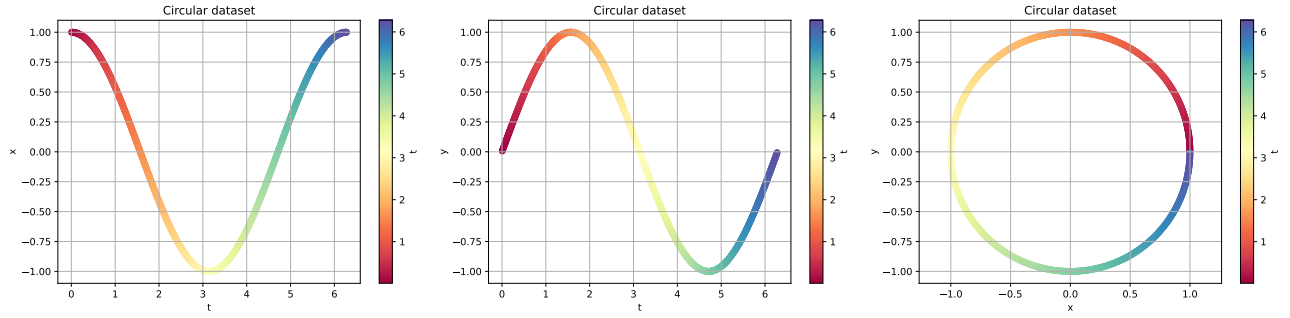
In this first section, we want to investigate the relation between Diffusion Maps and Fourier analysis as a tool for signal processing.

Introduction to the circular dataset

We are using a circular dataset with $N = 1000$ points given by:

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

Figure 10 shows the resulting dataset and how the coordinates x and y are defined by t.



(a) $x_t = cos(t_k)$ (b) $y_t = sin(t_k)$ (c) $\mathbf{x}_k = (cos(t_k), sin(t_k))$

Figure 10: Periodic dataset defined on the circle with radius $r = 1$. $N = 1000$ points are sampled based on equation (2) and colored based on the underlying value t_k. The coloring allows to inspect the topology of the data and later compare it with the transformed data to verify the results.

Applying the Diffusion Map algorithm and observing the projected data

Using the implementation of the above described algorithm, we compute the five eigenfunctions ϕ_l associated with the largest eigenvalues λ_l, already evaluated on the dataset X. This way, we directly get the projected coordinates in the new axis. Figure 11 shows the values of the eigenfunctions $\phi_l(x_k)$ against t_k (see equation 2).

[8] https://numpy.org/
[9] `https://scipy.org/`

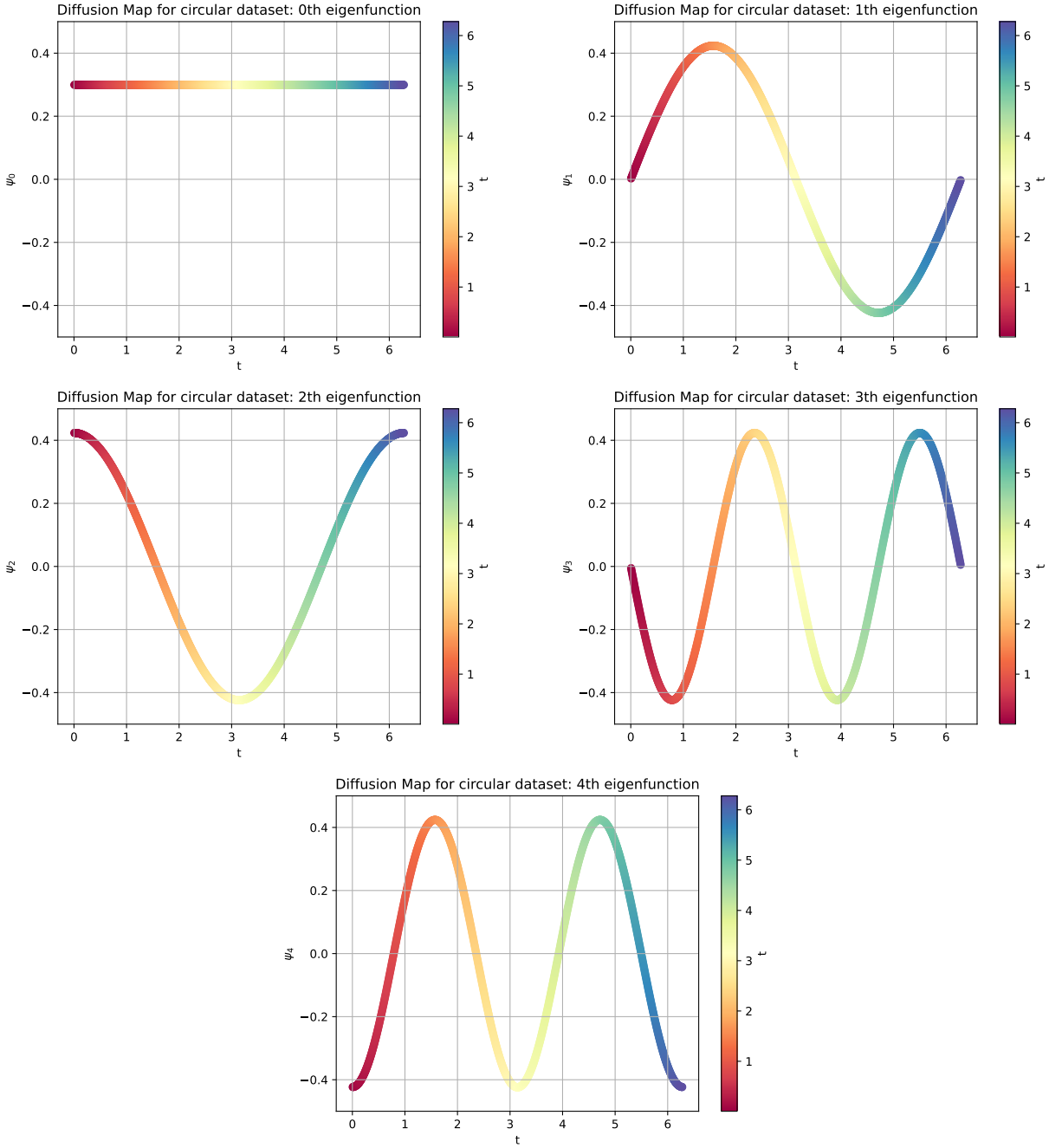


Figure 11: Results of Diffusion Map algorithm applied to circular dataset (2). Five eigenfunctions ϕ_l associated with the largest eigenvalues λ_l are calculated to get the projected coordinates ψ_l. Projections are plotted against t and colored based on their value t. Comparing the coloring to figure 10 indicates that the topology of the data is preserved. Note that the first projection ψ_0 is constant and that the projections ψ_1 and ψ_2 are qualitatively equivalent to the original coordinates x and y shown in figure 10a and 10b.

Comparing the results to the Fourier Analysis

When performing a Fourier Analysis on a signal X, one tries to decompose it as a sum of sine and cosine waves. This decomposition can also be seen in the plots of the evaluated eigenfunctions ψ_l in figure 11. The plot of t against the first (ψ_1) and the second evaluated eigenfunction (ψ_2) show the sine and cosine wave (= fundamental oscillation), the graph below shows the first harmonics. In a comparable fashion as the Fourier Analysis, our Diffusion Map algorithm opens one new subcoordinate ψ_l for each trigonometric function that represents the transformed data in the new projected space. As for the Fourier Analysis, these subsignals ψ_i are suffient to describe the original signal X and can be used to recover it back.

Diffusion Maps and PCA on nonlinear Swiss Roll dataset

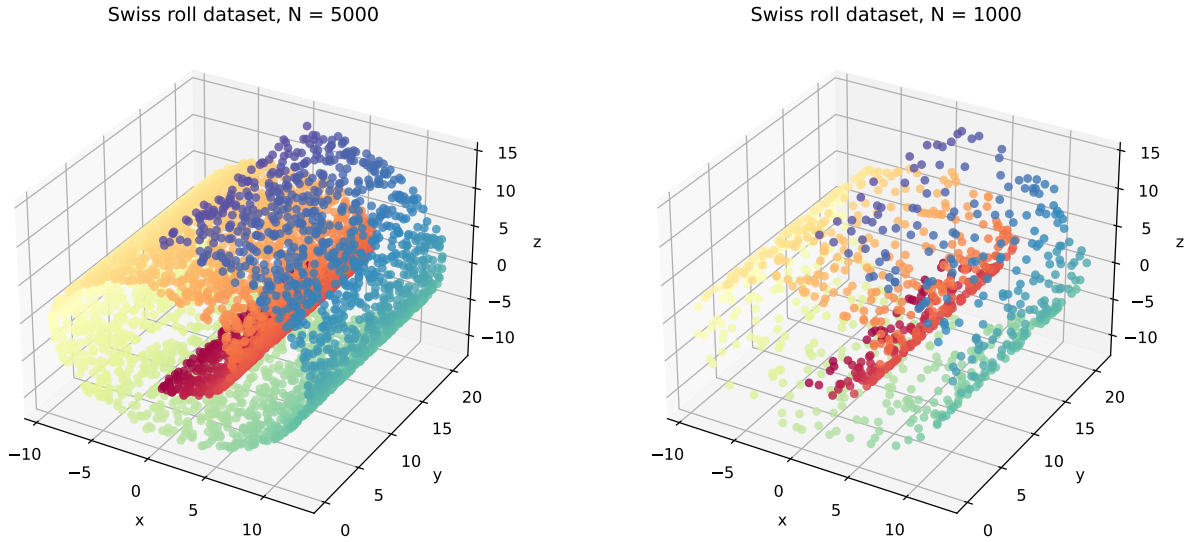
Next, we want to look at another non-linear dataset and compare the results of the non-linear Diffusion map algorithm to the linear PCA.

Introduction to the Swiss Roll dataset

The swiss roll dataset is a non-linear dataset, insipired by the the likewise named type of rolled sponge cake filled with whipped cream[10]. It is defined through:

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

where $(u, v) \in [0, 10]^2$ are chosen uniformly at random. The dataset can be loaded from the Python library *scikit-learn*[11] using *sklearn.datasets.make_swiss_roll*. Figure 12 shows the dataset for $N = 5000$ as well as $N = 1000$ sampled points. While the non-linear manifold is a lot sparser for $N = 1000$ sampled points, one can still see the topology of the roll like shape.



(a) $N = 5000$ (b) $N = 1000$

Figure 12: Swiss roll dataset. $N = 5000$ (a) and $N = 1000$ (b) points are sampled based on equation (3). The coloring of the data points shows the topology and later helps to verify the correctness of transformation methods.

[10] https://en.wikipedia.org/wiki/Swiss_roll
[11] https://scikit-learn.org/stable/index.html

Applying the Diffusion Map algorithm and observing the projected data

In order to transform the swiss roll into a lower dimensional manifold while preserving the topology, we compute the first ten eigenfunctions based on the Diffusion Map algorithm. Figures 15 and 16 show the projected coordinates where we plot the first non-constant projection ψ_1 against the other ones.

Here are the observations:

- The first eigenvector ψ_0 can usually be ignored since it is constant.
- For $N = 5000$, the eigenvector ψ_5 is the first one where ψ_l is no longer a function of ψ_1. For lower ψ_l, one can clearly see the functional dependence between ψ_1 and ψ_l.
- The eigenvector combination ψ_1 and ψ_5 shows to be a suitable low level representation for the swiss roll dataset, since it does not show a functional dependence and there are no intersections such that the representation is unique. It performs well in unrolling the roll into a meaningful 2D representation.
- Although the proposed Diffusion Maps algorithm tries to be robust against the sampling density, one can clearly see the qualitative differences in the projections between $N = 5000$ (figure 15) and $N = 1000$ (figure 16). The shapes of the resulting projections have a different number of intersections as well as local extrema. This concludes that the sampling density is an important property for manifold learning methods in general and one should make sure that the number of sample points of the underlying manifold is sufficiently high.

Applying the PCA algorithm and comparing the results to the Diffusion Map

In order to compare the non-linear Diffusion Maps method with the linear PCA, we also apply PCA on the swiss roll dataset, for both $N = 5000$ and $N = 1000$ sampled points. Figure 13 shows the results:

Here are the observations:

- As for the Diffusion Map algorithm, the results depend heavily on the sample size of the data. PCA applied on the sparser dataset with $N = 1000$ data points results in different principle components than with $N = 5000$.
- The swiss roll dataset requires to have 3 principle components to map onto in order to get a unique representation. For 2 principle components, all points are squashed on the plane and the 3D - structure of the data is lost and therefore cannot be recovered again.
- Unlike the Diffusion Map algorithm, the linear PCA algorithm fails to uncover the intrinsic structure of the data and the unfolded manifold.
- Both the Diffusion Map algorithm as well as PCA succeed in preserving the topology of the data.

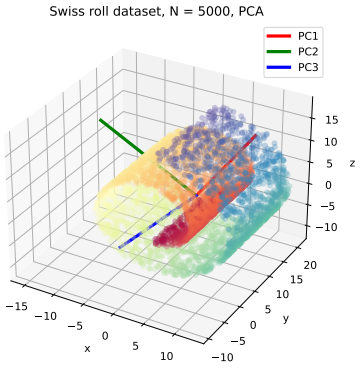
Analysing trajectory data with Diffusion Maps

In the previous subsections, we have been analyzing the application of the Diffusion Maps algorithm to some toy data. In the next subsection, we are going to analyze trajectory data recorded from the Vadere simulation software.

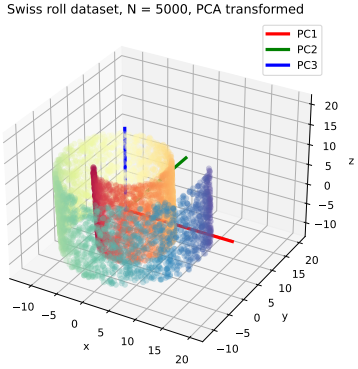
Introduction to the dataset

The trajectory dataset contains position data (x, y) of 15 pedestrians over 1000 discrete timesteps. It is of shape $(1000, 30)$ where each row contains the position information of all 15 pedestrians in a format of $(x_1,\ y_1,\ x_2,\ y_2,\ ...,\ x_{15},\ y_{15})$.

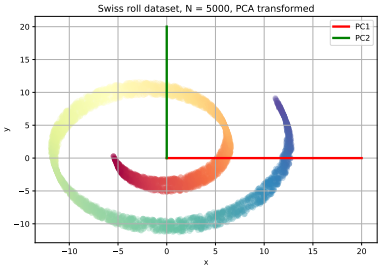
Figure 14 shows a visualization of the data. It is worth mentioning that the later applied Diffusion Map algorithm does not see the data as coming from different pedestrians (compare figure 14a), but much rather



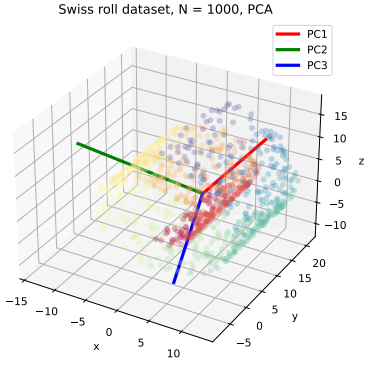
(a) $N = 5000$: Original data with principle components axes.



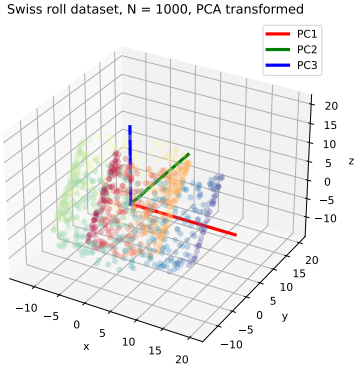
(b) $N = 5000$: Transformed data along principle components axes.



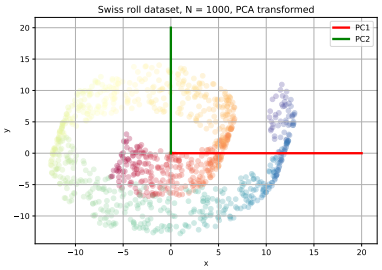
(c) $N = 5000$: Transformed data along first two principle components axes only.



(d) $N = 1000$: Original data with principle components axes.



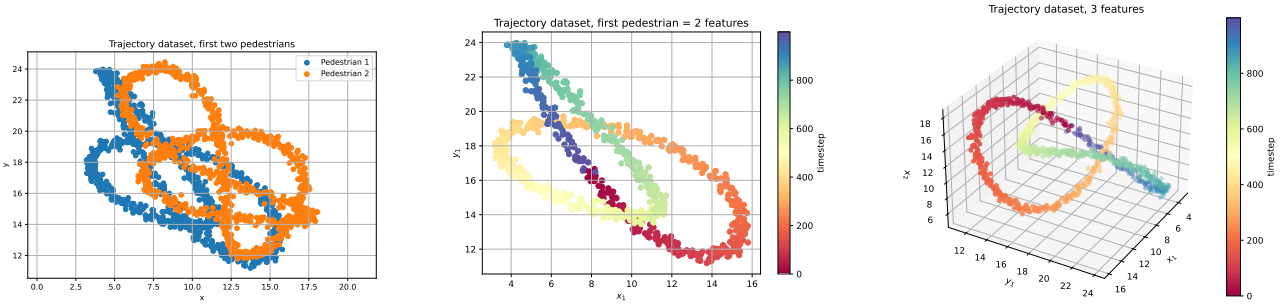
(e) $N = 1000$: Transformed data along principle components axes.



(f) $N = 1000$: Transformed data along first two principle components axes only.

Figure 13: PCA applied on swiss roll dataset of $N = 5000$ and $N = 1000$. The coordinate center of the eigenvector axis is equal to the mean of the of the dataset and lies within the roll. Unfortunately due to a bug in the 3D visualization of `matplotlib.pyplot`, this is not directly obvious from looking at the first two figures.

as 30 - dimensional data. Figure 14c shows this interpretation for three of the 30 features. Even in three dimensions, one can see the intertwining ring-like topology of the data, that should be preserved when performing the dimensionality reduction method.



(a) Trajectories of first two pedestrians.

(b) Trajectory of first pedestrian, colored by timestep.

(c) First three columns as features, colored by timestep.

Figure 14: Visualization of trajectory dataset of shape $(1000, 30)$ where each row contains the position information of all 15 pedestrians in a format of $(x_1,\ y_1,\ x_2,\ y_2,\ ...,\ x_{15},\ y_{15})$.

Applying Diffusion Map algorithm on trajectory data

We apply our Diffusion Map algorithm on the data and visualize the first nine eigenfunctions associated to the largest eigenvalues (figure 17). One can see that for the combination of the projections ψ_1 and ψ_2 we manage to get a unique ring-like representation of the low-dimensional manifold while preserving its topology. Since there are no intersections, we conclude that one can already accurately represent the data using two eigenfunctions.

Usage of datafold software on swiss roll dataset

In the last part, we want to compare our own implementation of the proposed Diffusion Map algorithm (2) with a proper implementation in the publicly available Python library `datafold`[12]. For this reason, we are using `datafold` to compute the eigenvectors of the swiss roll dataset to then compare it with the results of our own implementation.

Comparing the results of the `datafold` software (figure 18) with our own results (figure 15), one can see that the achieved projections are qualitatively equivalent. However during the application, we experienced that the `datafold` software is more optimized in terms of required computation, which leaves some room for improvements of our own implementation.

For computing the projections using the *datafold* library, the following steps are performed:

1. Get the swiss roll dataset using *sklearn.datasets.make_swiss_roll*.
2. Initialize the *datafold.pfold.PCManifold* class with with the dataset. *PCManifold* inherits from *np.ndarray*, holds the dataset as well as attaches a Gaussian kernel associated with the data to it. The Gaussian kernel describes the locality between point samples.
3. Call *pfold.PCManifold.optimize_parameters* to estimate a cutoff value below which the values of the Gaussian Kernel are considered to be zero, as well as a corresponding scale epsilon of the Gaussian. The idea of the cutoff is to allow for a sparse calculation that speeds up the computation for larger datasets. So far, all calculation happened on the original representation of the data only.
4. Instantiate the *datafold.dfold.DiffusionMaps* class with a Gaussian kernel using the just calculated cutoff and scale epsilon, as well as specifying the number of largest eigenpairs to compute.

[12] `https://datafold-dev.gitlab.io/datafold/index.html`

5. Fit the *DiffusionMaps* object to our swiss roll data by calling *datafold.dfold.DiffusionMaps.fit* and passing our dataset.
6. Retrieve the computed eigenvectors and corresponding eigenvalues as properties *eigenvectors_* and *eigenvalues_* of the fitted *datafold.dfold.DiffusionMaps* instance.
7. Plot the first non-constant eigenfunction against the others in 2D plots.

General questions

Finally, we want to reflect about our findings and the general working approach.

Estimated time it took to implement the code

Implementing the diffusion map algorithm as well as creating the datasets and the plots took a total time of approximately 12 hours. However most of this time comes from proper software engineering in terms of modularization and readability.

Accuracy of the method and evaluation criteria

In order to verify the results of our Diffusion Maps implementation, we:

- compare our results qualitatively with the results of the `datafold` software.
- make use of colored data points in order to verify the topology-preserving behaviour.
- Choose a set of eigenfunctions such that there are no overlaps or a squashing of datapoints for the embedding since this would indicate a non-accurate representation from which one cannot simply recover the original data due to a loss of topological information.

Using these pseudo-metrics, we come to the conclusion that our results are qualitatively good, without any quantitive proof however.

Learning outcomes about dataset and applied method

- Diffusion Maps is a non-linear manifold learning technique that can be used to uncover manifolds of non-linear structures such as the swiss roll.
- Linear manifold learning methods such as PCA fail to uncover the intrinsic structure of non-linear datasets.
- Both Diffusion Maps and PCA highly depend on the sampling density of the data and require a sufficiently high density of data points. While there exist methods to make the algorithms more robust against this, one should still be careful about this.
- Diffusion Maps and PCA are both techniques that preserve the topology of the data (= notion of local neighborhood) whereas the geometry (= metrics in space) might change.
- Intrinsic structure of trajectory dataset is of a ring like nature where two eigenfunction suffice to get a accurate non-overlapping representation.

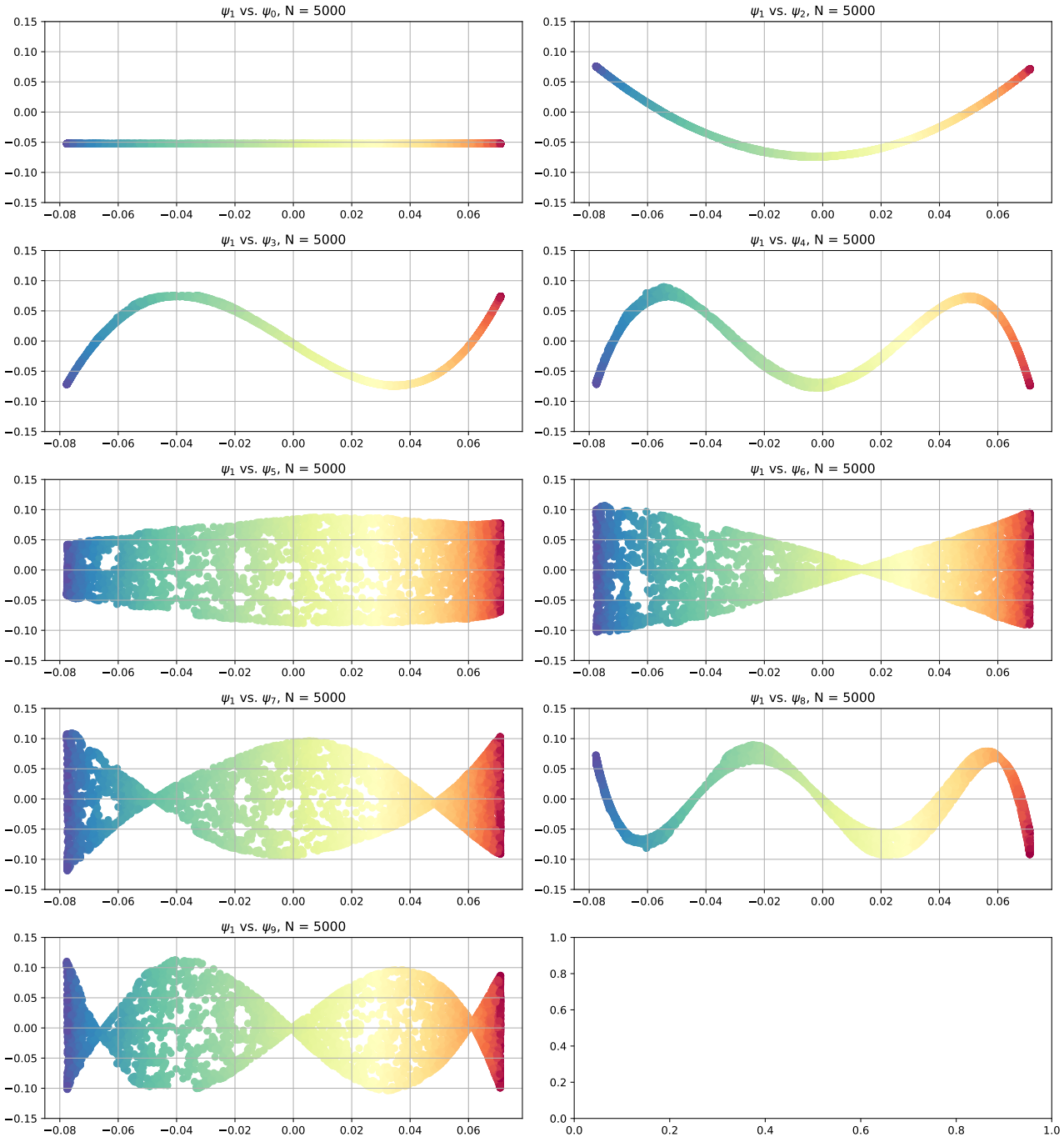


Figure 15: Diffusion Map embedding on the swiss roll dataset with $N = 5000$ points. Shown are the projections on the first ten eigenfunctions associated to the largest eigenvalues. Data points are colored based on the original topology of the swiss roll (compare with figure 12).

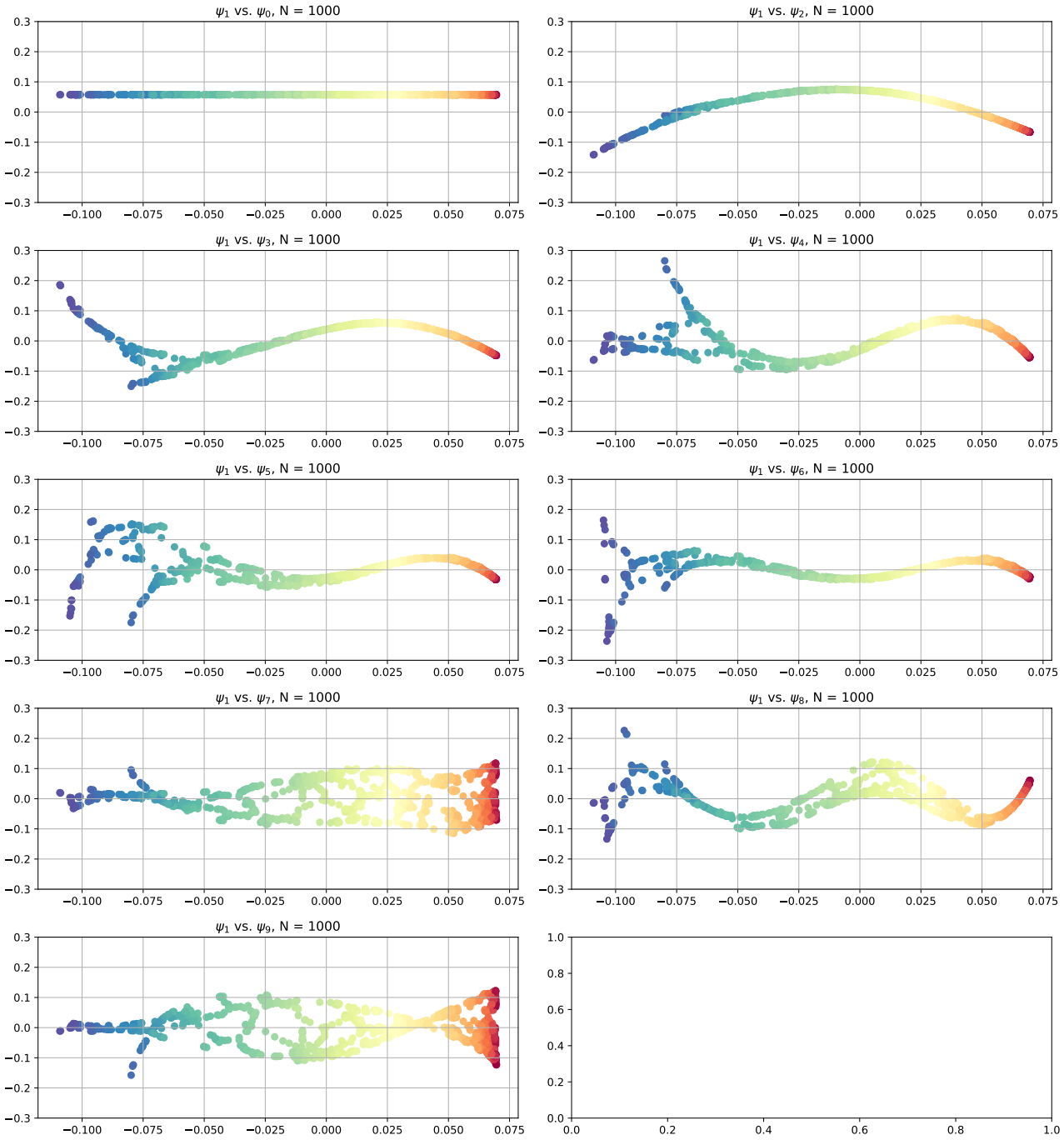


Figure 16: Diffusion Map embedding on the swiss roll dataset with $N = 1000$ points. Shown are the projections on the first ten eigenfunctions associated to the largest eigenvalues. Data points are colored based on the original topology of the swiss roll (compare figure 12).

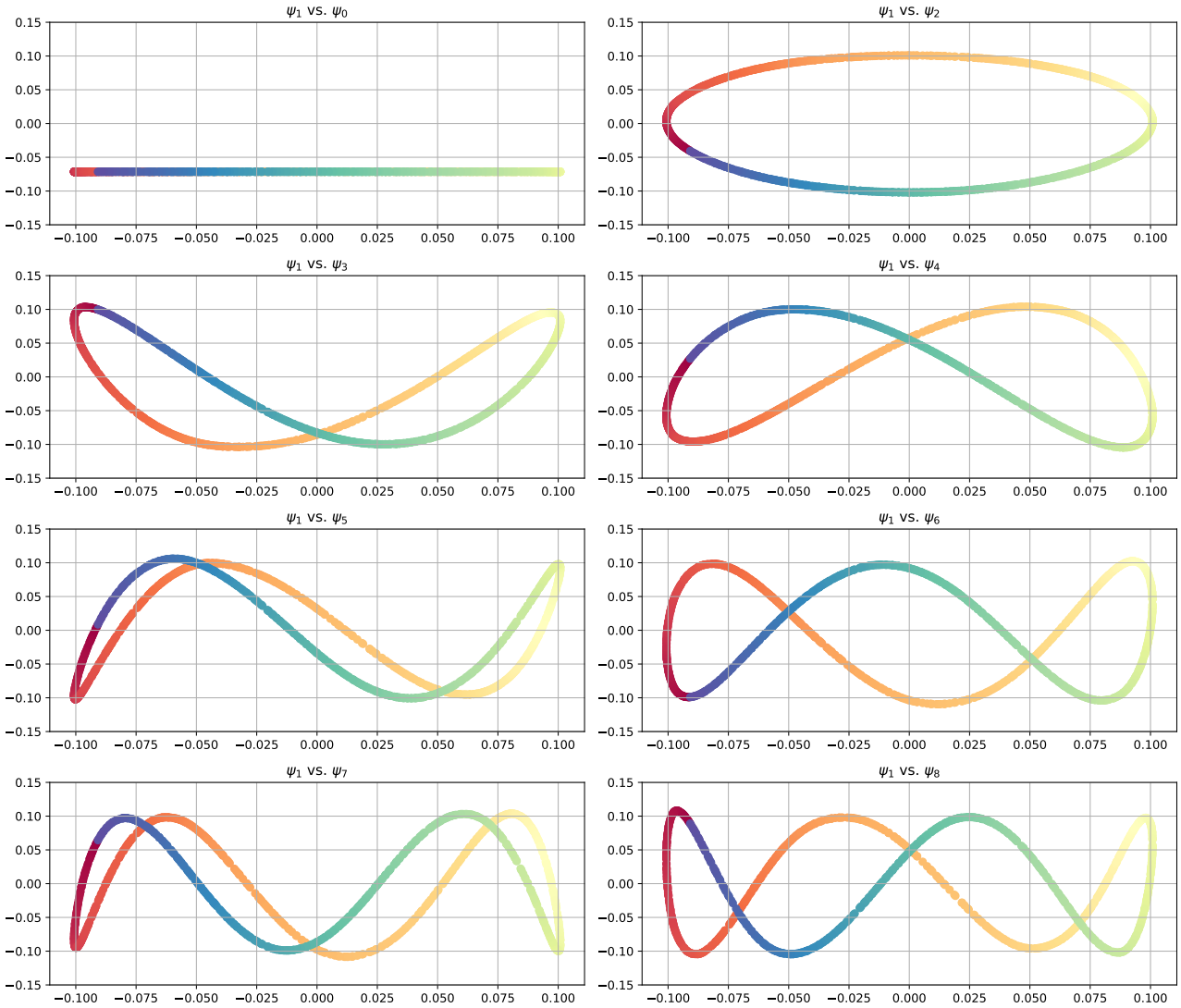


Figure 17: Diffusion Map embedding on the trajectory dataset where the x and y position of all pedestrians act as features of the original data. Shown are the projections on the first nine eigenfunctions associated to the largest eigenvalues. Data points are colored based on their index in the original dataset.

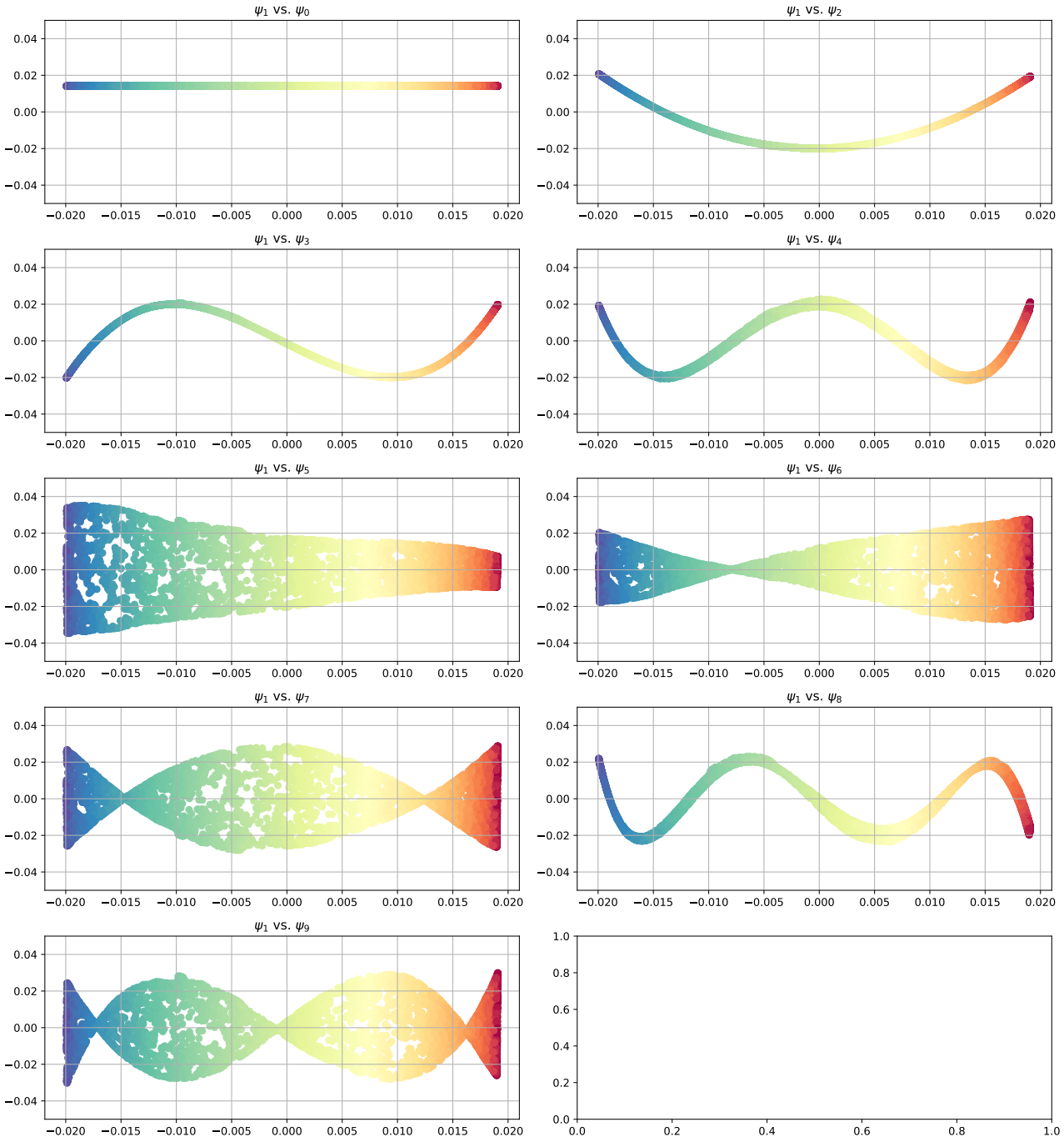


Figure 18: Diffusion Map embedding using the `datafold` software on the swiss roll dataset with $N = 5000$ points. Shown are the projections on the first ten eigenfunctions associated to the largest eigenvalues. Data points are colored based on the original topology of the swiss roll (compare figure 12).

Report on task 3/4: Training a Variational Autoencoder on MNIST

Notebook: `task_3.ipynb`
Code: `helpers/vae.py`

Motivation

Probabilistic inference and learning of models with difficult posterior distributions is not a simple task and requires a special application of Machine Learning. By using a *Variational Autoencoder (VAE)*, one solves a unsupervised learning task with the help of probability theory. This task implements a *VAE* in order to efficiently optimize the encodings of the *MNIST* dataset. In particular, it presents how a latent space representation of the *MNIST* dataset is learned, how the reconstruction and generation of digits differ in quality and how the dimension of the latent vector impacts the results.

Data Preparation

This task demonstrates how a *VAE* can be used to reconstruct known input and to generate new digits. Therefore, the *MNIST* dataset consists of input and target data for training and for testing and is downloaded by:

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

The training data consists of 60000 samples and the test data includes 10000 samples. A data point of the input data has shape $(28, 28)$ that represents an image of a digit and the target data includes the label for the associated image. The input data is not binarised, but normalised by dividing it with the maximal value 255 and reshaped to the form $(28, 28, 1)$.

Defining the Network

Theoretical Background

Autoencoders usually take high dimensional inputs, convert them to low dimensional feature vectors, and reconstruct the input data with tolerable loss of information. While the dimension reduction is done by an *encoder*, the input data is reconstructed by a *decoder*. *Variational Autoencoders* additionally learn distributions of the data instead of encoding samples independently in order to improve generalisation and reconstruction of unseen data.

In order to learn the distributions, we fix a *prior* and approximate a *posterior*. A common approach chooses a *multivariate diagonal Gaussian* distribution for *prior* and *posterior* [2]. However, the number of parameters drastically increases with the size of the input data, unseen data requires a new computation of local variational parameters and we assume strong factorization with the multivariate diagonal Gaussian. That is why, Louis Tiao [4] proposes to use an inference network that approximates the local variational parameters.

Defining the Loss

Because of the fact that the network is supposed to handle probability distribution, the standard loss functions that are implemented for *Tensorflow* do not suffice. That is why the approximate posterior and the true posterior are compared with the so-called *Kullback-Leibler (KL) divergence* in order to minimize their difference. This *KL divergence is defined as:*

$$KL(P||Q) = \sum P(X) * log(\frac{P(X)}{Q(X)}) \tag{4}$$

where P(X) & Q(X) are probability distributions [5]. For the implemented *VAE*, an adapted version is used:

$$KL = \frac{1}{2}\sum 1 + \sigma - \mu^2 - e^{\sigma} \tag{5}$$

Because the *KL divergence* depends on the marginal likelihood if we compute it for the approximate and the true posterior, we use an alternative loss function that is called *evidence lower bound (ELBO)*. This *ELBO loss* also uses the *KL divergence* but it takes the approximate posterior and the true prior. The *ELBO loss* is defined by:

$$ELBO(q) = \mathbb{E}_{q(z|x)}[log(p(x|z)) - KL(q(z|x)||p(z))] \tag{6}$$

In order to use minimisation for optimisation, this loss is negated. Since the input data is not binarised, the first term is computed by *mean squared error loss* and added to the *KL* loss.

Implementation

The *VAE* is implemented by using *TensorFlow 2*[13]. The implementations of the custom sampling layer, the encoder, the decoder and the *VAE* are found in the separate file `helpers/vae.py` in order to make it reusable for other datasets. In addition to that, all parameters that define the input, output, and intermediate shapes can be set to prevent limitations because of the input shapes. For the *MNIST* dataset, the input shape is $(28, 28, 1)$. In the following, we demonstrate how the *VAE* is implemented with two dimensions of the latent space ($latent\_dim = 2$).

The structure of the encoder is visualised in Tab. 4. First, the input is flattened in order to have a shape with a single dimension. Then, there are two Dense layers with 256 units and *ReLU* activation functions. The same output of the last Dense layer is fed into two Dense layers where one outputs the mean and the other outputs the variance. The last layer is a custom layer that computes a sample for the input z of the decoder. This is necessary because we have a a more complex loss function that deals with a unknown probability and might not be differentiable [5]. Kingma and Welling [3] introduce a *reparameterisation trick* that we will use by computing $z = \mu + e^{\sigma/2} * \epsilon$ where ϵ is chosen randomly from the normal distribution and has the same shape as μ. This way, it is still possible to use gradient descent with a loss function that involves a distribution that it is unknown a priori.

Layer (type)	**Output Shape**
(Dense)	(256,)
(Dense)	(256,)
(Dense)	(784,)
(Reshape)	(28,28,1)

Table 3: Structure of model of decoder

The decoder (see Tab. 3) mirrors some layers of the encoder, but directly starts with the two Dense layers with 256 units and ReLU activation functions that receive the computed latent vector z. Then, another Dense layer increases the size such that it can be reshaped to the shape of the original input.

Layer (type)	**Output Shape**
(Flatten)	(784,)
(Dense)	(256,)
(Dense)	(256,)
latent_mu (Dense)	(2,)
latent_sigma (Dense)	(2,)
sampling (Sampling)	(2,)

Table 4: Structure of model of encoder

The structure of the *VAE* model is illustrated in Tab. 5. It combines the encoder and the decoder by taking the encoder's input and feeding it into the decoder.

[13] `https://www.tensorflow.org/`

Layer (type)	Output Shape
encoder (Encoder)	(2,)
decoder (Decoder)	(28,28,1)

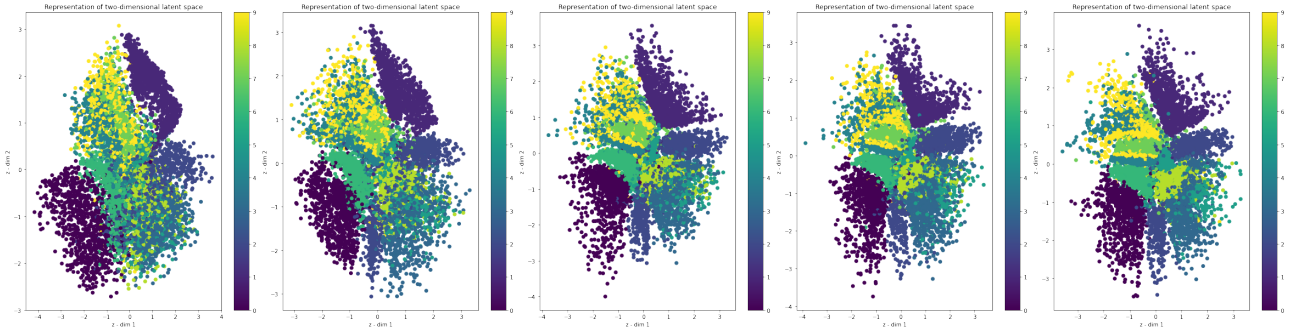
Table 5: Structure of the model of the Variational Autoencoder

The *Adam* optimiser is used for the optimisation of the *VAE*. The learning rate is chosen as $\alpha = 0.001$, the batch size is 128, and the train dataset is used for training while the test dataset is used for validation. The maximum number of epochs is set to 200 and early stopping is used with a patience of 10 in order to prevent overfitting.

Observations

The quality of the system is tested by visualising the latent representation, the reconstruction of images, the generation of new digits, and the development of the loss for a two dimensional latent space. Additionally, reconstruction, generation and loss are illustrated for a latent space of dimension 32 in order to compare the results to the lower dimensional latent space.

Latent_dim = 2



(a) Epoch 1 (b) Epoch 5 (c) Epoch 25 (d) Epoch 50 (e) Epoch 120: after convergence

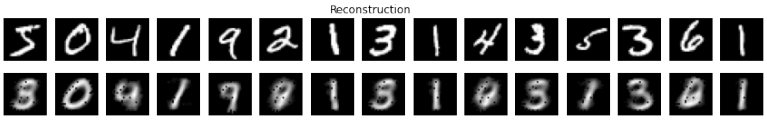
Figure 19: Representation of two dimensional latent space after different numbers of epochs where the labels of input data are indicated by a colour bar

The development of the latent space for a two dimensional latent space is illustrated in Fig. 19. One is able to observe that the system learns to separate the data according to the different labels and performs differently well for different digits. In particular, the numbers **0** and **1** perform well because there is a clear cluster and the numbers **3** and **5**, as well as, **4** and **9** are more difficult to separate.

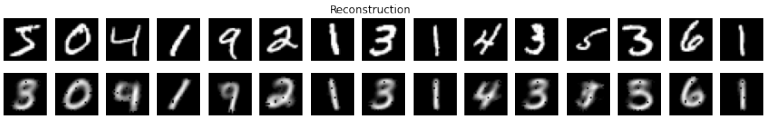
The same results are to be observed when analysing the reconstruction of images. While some numbers look very similar to their original image, others form the wrong digits like the first data point where a **5** becomes a **3**. The quality of the images improves with the number of epochs where images become less blurry with less noise. However, one is able to already identify wrong reconstructions after few epochs.

For the generation of new digits, the input of the decoder is chosen randomly. That is why, one is not able to compare two individual images. But it is obvious that the quality of the images improves. While the images after the first epoch are very blurry and have a lot of wrong black pixels, the images are very clear after convergences with a little bit of noise. Analysing the input vectors that are responsible for good results reveals that they are inside one of the clear clusters if they are mapped to our latent space representation. Input vectors close to zero particularly perform badly.

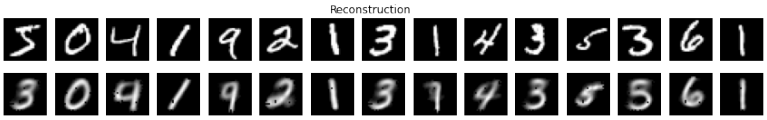
Another metric is the loss curve of the network. The loss curve of the validation loss that is calculated for the test set is illustrated in Fig.22). Because a high *ELBO* loss indicates a good approximation and the optimisation



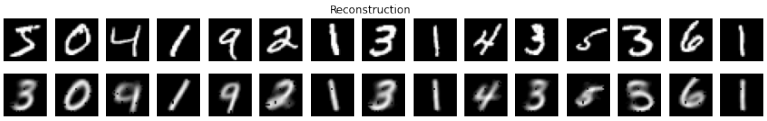
(a) Epoch 1



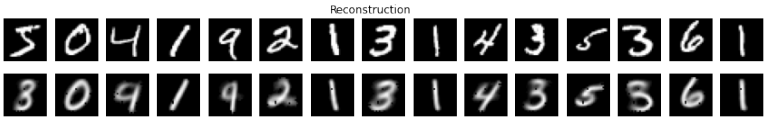
(b) Epoch 5



(c) Epoch 25



(d) Epoch 50



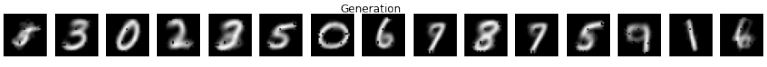
(e) Epoch 120

Figure 20: Reconstruction of digits compared to original input after different numbers of epochs

(a) Epoch 1

(b) Epoch 5

(c) Epoch 25



(d) Epoch 50

(e) Epoch 120

Figure 21: Generation of new digits with a randomly chosen mean after different numbers of epochs

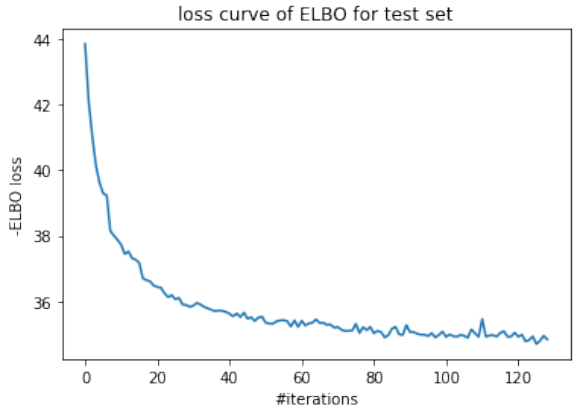


Figure 22: Development of loss for negated evidence lower bound (ELBO) until convergence for $latent\_dim = 2$

minimises losses, the *ELBO* loss is negated and the loss curve decreases. This indicates that the learning works as expected. Furthermore, the validation loss is higher like the training loss and the training loss does not increase which might indicate overfitting. All these metrics combined let us assume that the implemented *VAE* has a good performance with adequate results.

Latent_dim = 32

While it is possible to visualise the latent space representation for $latent\_dim = 2$, the remaining metrics need to be sufficient for higher dimensional latent spaces.

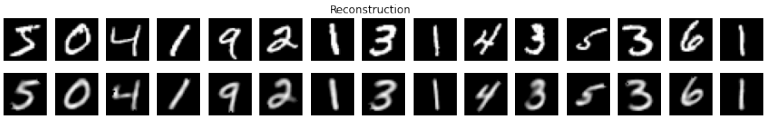


Figure 23: Reconstruction of digits compared to original input after convergence for $latent\_dim = 32$

Fig. 23 and 24 illustrate the reconstruction of the original data and the generation of new digits. Particularly, the reconstruction has convincing results. There are no images that form the wrong digit, there is minimal bluriness and noise. However, the generation performs not too good. Some images do not even represent any known digit. This might be a result of the fact that the decoder's input is chosen randomly. For two dimensions, this still leads to good results but not for a latent space with 32 dimension. The input space becomes very large with increasing dimensions. This makes it more difficult to choose input vectors that lead to good results because it is close to one of the clusters. The probability to receive a output of the encoder that is a good input for the decoder is higher for reconstructed digits because they already come from a distribution that represents an existing digit. If we know outputs of the encoder that lead to good results for the reconstruction, we can reuse and slightly modify them to achieve better results for generation. One is able to assume, it is sufficient for data generation to have a low dimensional latent space for the input data that only represents 10 digits. However, if the goal is the reconstruction, one could use the higher dimensional latent space or simply use a regular *Autoencoder*.

Figure 24: Generation of new digits with a randomly chosen mean after convegence for $latent\_dim = 32$

The loss curve for a latent space with dimension 32 (see Fig. 25) looks very similar to the loss curve for a two dimensional latent space. Only the loss converges to a smaller number and the converges takes a higher number of epochs. The latter is easily explained by the fact that a higher dimensional latent space also results in a higher number of parameters.

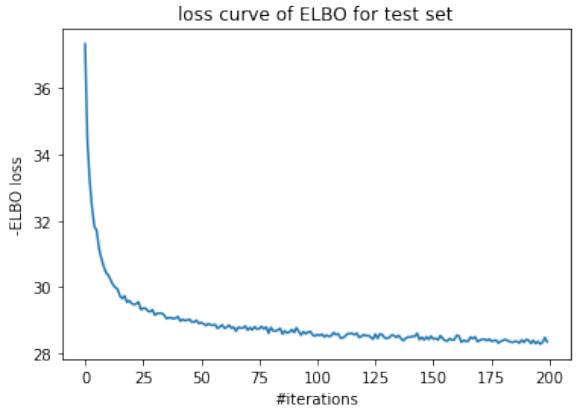


Figure 25: Development of loss for negated evidence lower bound (ELBO) until convergence for $latent\_dim = 32$

Discussion

The presented *Variational Autoencoder* seems to achieve good results on the *MNIST* dataset. But it deviates from the proposed system by using a inference network instead of explicitly implementing probability distributions. Futhermore, there are no activation functions for the Dense layers that output the mean and the variance. It is possible to argue why no activation function is chosen for the mean or the variance of the approximate posterior and the likelihood. The posterior represents the output of the encoder and the likelihood represents the output of the decoder. In case of the decoder, we do not need any activation function for the mean and the standard deviation because we need the real values between zero and one in order to produce a digit. The posterior samples $z = \mu + e^{\sigma/2} * \epsilon$ where epsilon is chosen randomly from a Normal distribution. Therefore, we should not modify the mean because it determines which digit we will receive. The standard deviation has less of an effect, and it might be useful to avoid that the second part of the sum gets to small in order to provide samples that differ from the mean. A possible activation function is ReLU. But at the same time, we do not want to deviate too much such that we still classify the right digit. So, softplus might be an alternative for an activation function for the standard deviation. However, we have chosen to use no activation function for posterior and likelihood because we receive the best results without them and most of the literature abstinates from it.

General questions

Finally, we want to reflect about our findings and the general working approach.

Estimated time it took to implement the code

The implementation and the testing of the *VAE* combined with the training and testing with the *MNIST* dataset took 25 working hours. Approximately additional 20 hours were spent on research and refactoring the *VAE* such that it is reusable for different datasets.

Accuracy of the method and evaluation criteria

In order to test the quality of the system four metrics were used. The latent space representation, the reconstruction of original data, the generation of new digits and the loss curve were compare to other systems with the same task. However, no numerical values have been compared and the system only was tested for two different values of the latent space because of the lack of literature for other values.

Learning outcomes about dataset and applied method

- high dimensional data like the *MNIST* dataset that consists of images can be reconstructed without major information loss even though the dimensionality is reduced from 784 to 32.
- with *Variational Autoencoders* one is able to generate data that is similar to the input data and has the same underlying probability distribution
- there are a lot of different ways of construction a *VAE* by using different layers, activation functions, or probability distributions

Report on task 4/4: Fire Evacuation Planning for the MI Building

Notebook: `task_4.ipynb`
Code: `helpers/vae.py`

Motivation

In the following, the implementation of the Variational Autoencoder of task three should be used to learn the underlying data distribution of a dataset containing positional information of students in a building scenario. The estimated findings should be used for enhancements of the fire evacuation planning.

Introduction to the FireEvac dataset

The FireEvac dataset contains 3600 positions (x, y) of students inside the Mathematics & Informatics building of the Technical University Munich. It is split into 3000 rows for training, and 600 rows for testing. The data was recorded by tracking the position of 100 random students and employees throughout the busiest hour on different days. The recorded data aims to estimate the underlying distribution of people inside the MI building. For the learning process and also the evaluation, this dataset was then further subdivided into a test and training dataset. This is visible in Figure 26.

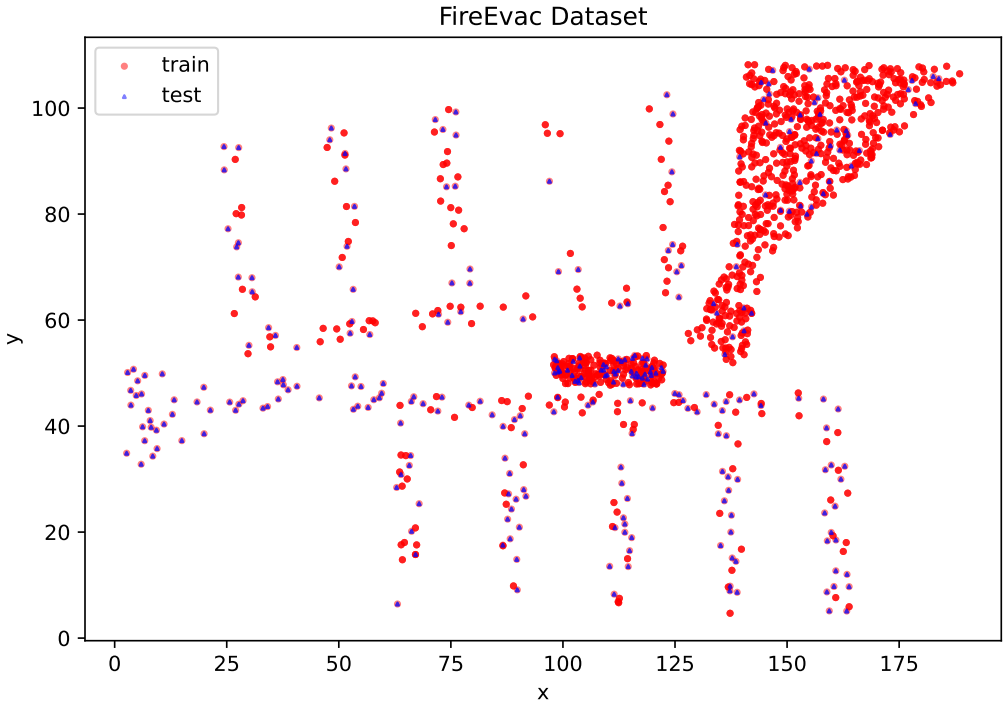


Figure 26: FireEvac dataset split into *train* and *test*. Contains positional data of students and employees inside the Mathematics & Informatics building of the Technical University Munich. Single data points are shown half transparent to easily indicate possible overlaps of positions in the data.

Learning the underlying data distribution $p(\mathbf{x})$ using a VAE

Implementation

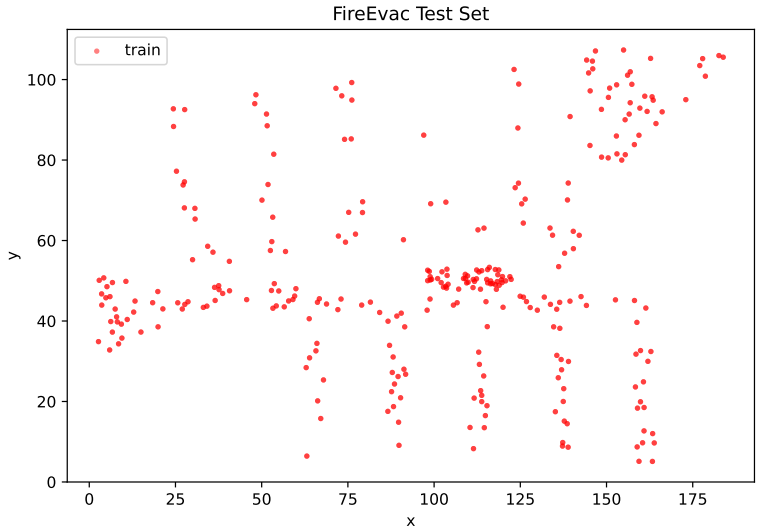
In order to learn the underlying probability distribution $p(x)$, the following section will use the implementation from task three to reconstruct the test. The structure of the VAE again includes the encoder and the decoder with a flat layer, two dense layers and dense layers for the mean and variance, as well as a sampling layer (see Fig. 4), three dense layers and a reshape layer (see Fig. 3). The entire structure can be taken from Fig. 5. Contrary to the recommendation of the problem definition that the input value x should be scaled to the interval $[-1, 1]$, our evaluations have shown that the best results are obtained for a normalised input in the interval $x \in [0, 1]$.

The following section will briefly describe the procedure and the newly identified hyperparameters. In the following section, the best hyperparameters are briefly described. in contrast to the implementation in task three, the reduction of the batch size seemed to make sense here, since the initial data set is also smaller. for this purpose, different batch sizes between 2 and 128 were evaluated. the best result with regard to this parameter was obtained for $n_{batch} = 64$. Another hyperparameter is the latent dim. Here, systematic tests were carried out for the usecase in the interval $[2, 32]$. The best result was obtained for latent dimension $= 2$. In accordance to this systematic hyperparameter search approach, the hidden size and number of dense layers were also tested. This resulted in a hidden size of 64 and a number of 2 dense layers. Beyond that, no structural adjustments were made, so that both the loss function and the optimiser (ADAM) used in the optimisation step remained the same. To prevent overfitting of the autoencoder, early stopping was again used as the regulation technique. To prevent divergence during the learning process, a small step size of $\alpha_{LR} = 0.005$ was chosen.

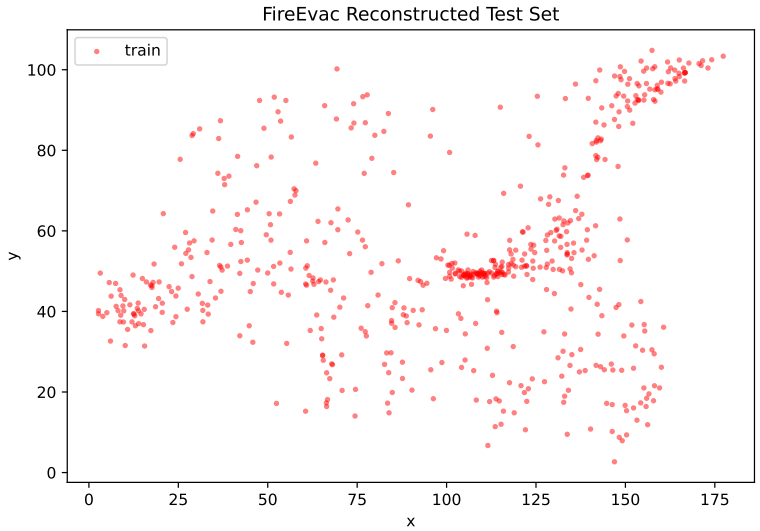
Observations

One possible result of the reconstruction of the test set can be seen in Figure 27. Observable is that the ordered structure from Figure 26 has been broken up. Particularly, the samples are no longer arranged along parallel corridors in the MI building, but also fill the white spaces in between. Nevertheless, the VAE was able to reconstruct dense areas of the building, such as the central corridor or the entrance area and is therefore capable to at least capture essential structures of the initial dataset.

Despite the fact that essential characteristics of the data set could be reconstructed, a critical examination of the structure and the respective parameters to be learned of the VAE is still necessary, since certain structures were not taken into account (see missing aisles) in the reconstruction. For this purpose, the learning process will be examined in more detail in the following section.



(a) Original test set



(b) Reconstruction

Figure 27: Original FireEvac *test* dataset and reconstruction containing the positional data of students and employees inside the Mathematics & Informatics building of the Technical University Munich.

Again, the ELBO Loss is utilised within this model. The progression over 200 epochs can be seen in Figure 28. The rapid drop-off in the first 10 iterations is positive and desirable, which indicates fast learning and convergence. After that, there is a plateau until about the 55th iteration, where no improvement can be observed. Afterwards, the loss function drops again and stagnates at a constant value of about 4.8. In the considered time window of n=200 epochs, the model could not reach a significantly lower loss value.the magnitude of the loss suggests, similar to the consideration of the results above, that there is still a need for optimization of the model, but this will not be pursued further due to the limited time and resources for this report.

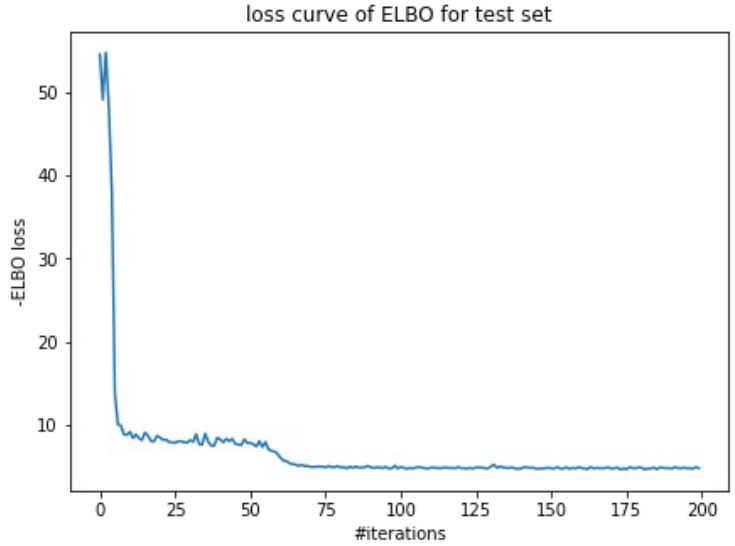


Figure 28: Development of loss for negated evidence lower bound (ELBO) until convergence for latent dim= 2

After assessing the learning process of the autoencoder by evaluating the structure of the loss curve, the VAE should be used to generate 1000 samples. The result of the process is shown in Figure 29. The outcome seems to have very little in common with the initial structure of the building. One common feature could be found, for example, in the dense crowding at the entrance area. Nevertheless, it seems that the arms leading away from the central aisle in particular are difficult to learn. Here the distribution looks very arbitrary and the actual structure is no longer recognisable.

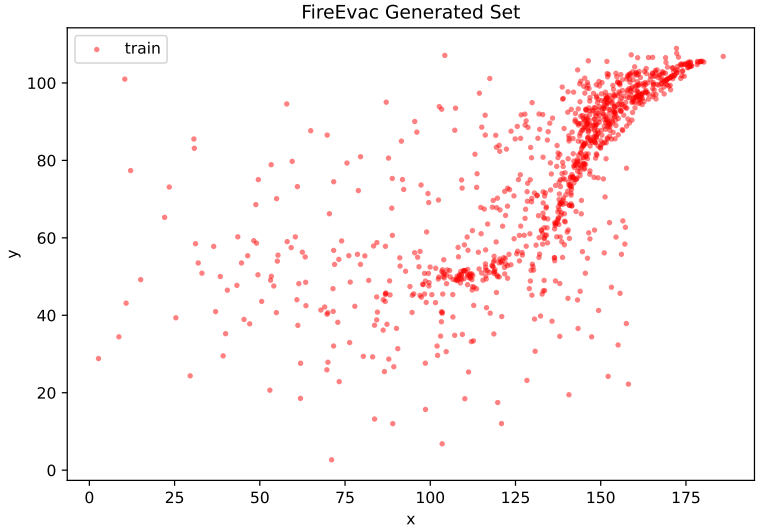


Figure 29: Generated FireEvac dataset containing the positional data of students and employees inside the Mathematics & Informatics building of the Technical University Munich.

Estimating the critical number of people for the MI building

A possible real world application for the VAE could be that one would like to determine a maximum critical quantity of visitors in the fire protection design of buildings. For this purpose, the critical number of students and employees in a sensitive area of the MI building is to be determined in the following.

Therefore, a sensitive area at the entrance of the building is chosen at the location [130¡x¡150, 50¡y¡70]. Since approximately 10% of all generated samples are located in this area, the number of people is calculated for $n \in [950, 1100]$. Because the input for the generation is chosen randomly, the number of generated samples, for which the critical number of 100 people is reached, varies. If the only goal is to count the number of people when there are 100 people in the area, then the number of samples is around 1000. However, if the number is calculated because of reason regarding safety then often the number already is reached for 900-950 people and less people should enter the building.

Bonus: Crowd simulation

In order to simulate the crowd, the fire department wants to use the learned distribution. Therefore, positions of 100 people were sampled (see Fig. 30). Due to time reason, an application with the Vadere simulation software is not included.

Figure 30: Reconstructed FireEvac *test* dataset containing the positional data of students and employees inside the Mathematics & Informatics building of the Technical University Munich.

General questions

Finally, we want to reflect about our findings and the general working approach.

Estimated time it took to implement the code

After the basic structure of the encoder was already worked out in task three, the time needed for the pure implementation was not very high. Nevertheless, the systematic hyperparameter search clearly exceeded the scope of this exercise, since, among other things, interactions between hyperparameters are difficult to see and assess with only basic machine learning knowledge. Furthermore, the previous and this task part seem to be based on old tutorials, which hardly seem to be implementable in a meaningful way with current frameworks.

Accuracy of the method and evaluation criteria

The overall evaluation of the VAE has shown that the implementation does not perform convincingly in the aspects of reconstruction and generation of new samples. The poor learning of the fingers leading away from the side of the building is one of the reasons for this. Therefore, this implementation in its current state is only conditionally suitable for a safety assessment of buildings. One possible reason can be found in the higher estimation of entrances than they correspond to reality. Therefore, further rethinking of the entire setup as well as an extended systematic hyperparameter tuning is needed. Remarkable was the optimal distribution when the sensitive area is the point of interest.

Learning outcomes about dataset and applied method

- The FireEvac dataset has a complex underlying structure
- One should not rely on the implemented *VAE* in case of questions regarding safety because data generation works poorly for small datasets with 2 dimensions
- The implemented *VAE* suffices for rough approximations and might improve if the system is adapted in terms of normalisation of the input data, number and form of layers and hyperparameters.

References

[1] T. Berry. Time-scale separation from diffusion-mapped delay coordinates — siam journal on applied dynamical systems — vol. 12, no. 2 — society for industrial and applied mathematics, 12/18/2021.

[2] Danijar Hafner. Building variational auto-encoders in tensorflow. Blog post, 12/12/2021.

[3] Diederik Kingma and Max Welling. Auto-encoding variational bayes. 2013.

[4] Louis Tiao. How to create a variational autoencoder with keras?, 12/16/2021.

[5] Christian Versloot. How to create a variational autoencoder with keras?, 12/12/2021.