Report for exercise 3 from group F

Tasks addressed:	4
Authors:	Jianzhe Liu (03751196) Hao Chen (03764817) Yang Cheng (03765398) Pemba Sherpa (03760783)
Last compiled:	2023–05–29
Source code:	https://github.com/Chuck00027/MLCMS-GroupF

The work on tasks was divided in the following way:

Jianzhe Liu (03751196) **Project lead**	Task 1	0%
	Task 2	25%
	Task 3	0%
	Task 4	100%
Hao Chen (03764817)	Task 1	50%
	Task 2	0%
	Task 3	50%
	Task 4	0%
Yang Cheng (03765398)	Task 1	50%
	Task 2	0%
	Task 3	50%
	Task 4	0%
Pemba Sherpa (03760783)	Task 1	0%
	Task 2	75%
	Task 3	0%
	Task 4	0%

Report on task TASK 1, Principal component analysis

Part1 Implement principal component analysis

1.Briefly explain PCA PCA is a classical method for dimensionality reduction. It aims to transform a dataset with a high-dimensional feature space into a lower-dimensional space while retaining the most important information or patterns present in the data.

2.Plot the data set as in the figure, add the direction of the two principal components In step1 and step2, we load the dataset as shown in Figure 1

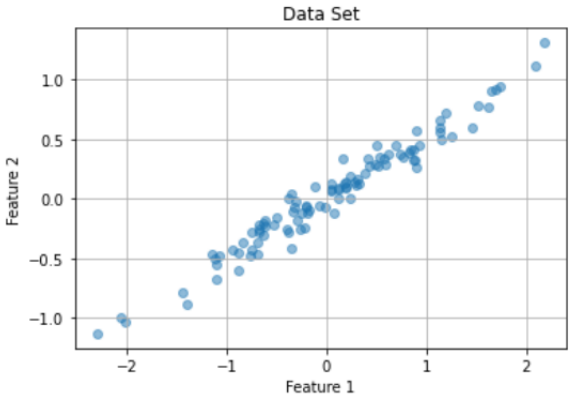


Figure 1: Plot original dataset

Step 3: Preprocess the data: In this step, the dataset data is centered by subtracting the mean of each feature from the corresponding feature values. This process ensures that the data is centered around the origin.

Step 4: Compute the Singular Value Decomposition (SVD): The SVD is calculated on the centered data using the NumPy function np.linalg.svd(). It decomposes the centered data matrix into three matrices: U, s, and Vt. U represents the left singular vectors, s contains the singular values, and Vt contains the right singular vectors.

Step 5: Determine the principal components: The principal components are determined by extracting the first and second column vectors from the right singular vectors matrix, Vt. These column vectors represent the directions in which the data varies the most.

Step 6: Calculate the energy in each component: The energy of each principal component is calculated by dividing the squared singular values by the sum of all squared singular values. This energy represents the proportion of total variance explained by each principal component.

Step 7: Plot the data set with principal components: A scatter plot is created to visualize the original dataset data using the first two features(As shown in Figure 2). The principal components are represented as arrows originating from the mean of the dataset. The length and direction of the arrows indicate the contribution and direction of each principal component.

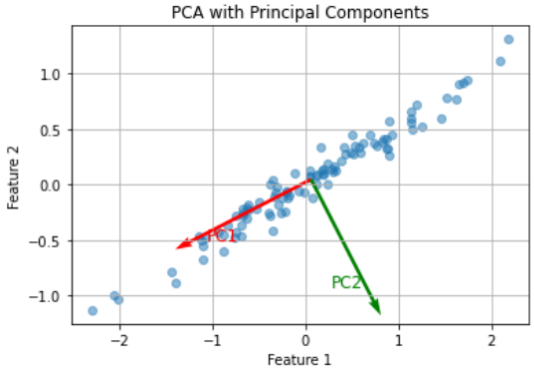


Figure 2: 2 principle components in the scatter plot of the data set

3.How much energy is contained in each of the two components? To calculate how much energy

is contained in each of the two principal components,we first need to compute the total energy.The total energy is determined by summing the squared singular values obtained from the SVD.Then we calculate the energy in the first principal component (PC1).The energy in PC1 is the proportion of the total energy contributed by the squared singular value associated with PC1. And similar to PC1, the energy in PC2 is the proportion of the total energy contributed by the squared singular value associated with PC2. The corresponding code in shown in the table below

```
# Step 6: Calculate the energy in each component
total_energy = np.sum(s ** 2)
energy_pc1 = (s[0] ** 2) / total_energy
energy_pc2 = (s[1] ** 2) / total_energy

...

print("Energy contained in each component:")
print("Component 1: {:.2%}".format(energy_pc1))
print("Component 2: {:.2%}".format(energy_pc2))
```

Edited code from Task 1.ipynb

As the result in the file **Task-1**,We get that the first principal component captures 99.31% of the energy and the second principal component captures 0.69% of the energy. They already capture almost all of the energy, so only these two principal components are needed.

Part2 Apply PCA to the image

1.Load and resize the image In the first step, we need to load the image and reesize it to 249 × 185.We download the image named "PIXNIO-28860-1536x1152.jpeg".Then we open this image in grayscale. Then according to the request, we resize the original image to 249 × 185.The output is shoen in Figure 3.

Figure 3: resized image

2.Visualize reconstructions of the image

Step1:First we convert the resized image to NumPy Array and Transpose.The resulting array is transposed using .T to ensure that the shape is compatible with the PCA algorithm.

Step2:We perform PCA on the image data.The scikit-learn library provides an implementation of PCA through the PCA class, which allows users to perform PCA on their data. Here we appy this method to be more effective.The transformed components are stored in the components variable.

Step3:Visualize reconstructions with different numbers of principal components.We aim to demonstrate how different numbers of principal components affect the reconstruction of an image using PCA. By setting certain singular values to zero in the PCA components, the code progressively reduces the number of components used for reconstruction, resulting in images with varying levels of detail. The purpose is to visually compare and understand the trade-off between the amount of information retained and the quality of the reconstructed images.

Step4:Visualize the reconstructions.In this step, we plot the result in terms of 'All Components', '120 Components', '50 Components', '10 Components'.(Figure4)

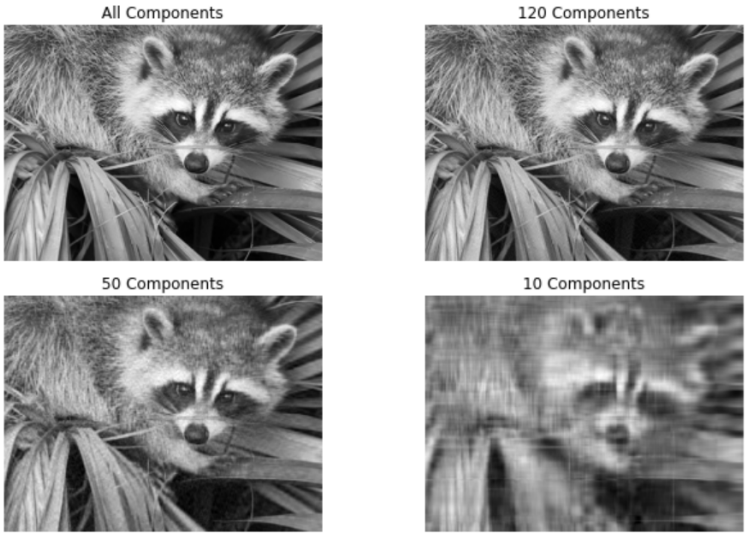


Figure 4: Reconstruction of the image using different number of principle components

3.Analyse In this part, we need to estimate at what number is the information loss visible and at what number is the energy lost through truncation smaller than 1%. We start by computing the explained variance ratio for each principal component, which represents the proportion of variance explained by that component. The cumulative variance ratio is then calculated by taking the cumulative sum of the explained variance ratios.We estimates the number of principal components at which the information loss becomes visible by finding the index where the explained variance ratio drops below a certain threshold (in this case, 0.01). The code determines the index at which the cumulative variance ratio exceeds 0.99, indicating the number of principal components required to explain at least 99% of the variance. This index value is stored in *threshold_index*. Similarly, the code finds the index at which the explained variance ratio falls below 0.01, representing the number of principal components where the energy lost through truncation is less than 1%. The corresponding index is stored in *visible_loss_index*. By adding 1 to both index values, the actual number of principal components where information loss becomes visible (*visible_loss_num*) and where energy loss is below 1% (*energy_loss_num*) is obtained. Finally, the code prints the result shown in Figure5.

```
Number where information loss becomes visible: 13
Number where energy lost through truncation < 1%: 89
```

Figure 5: output result

```
# Calculate information loss and energy loss thresholds
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

threshold_index = np.argmax(cumulative_variance_ratio > 0.99)
visible_loss_index = np.argmax(explained_variance_ratio < 0.01)

visible_loss_num = visible_loss_index + 1
energy_loss_num = threshold_index + 1

print("Number where information loss becomes visible:", visible_loss_num)
print("Number where energy lost through truncation < 1%:", energy_loss_num)
```

Code part for calculate information loss and energy loss thresholds

Part3 Analyse the trajectory data

1.Visualize the path of the first two pedestrians in 2D space At first we load the corresponding dataset . In this part we need to visualize the first two pedestrians' path in $two-dimensional$ space.As the

reslut shown in FIgure 6 , we could find out their trajectory patterns are somehow same.They both perform the character '8' trajectory.But their trajectories are only roughly the same. Their respective amplitudes have been scaled and their periods are switched.Furthermore, although there are differences in their specific trajectories, they are all concentrated in an almost identical area

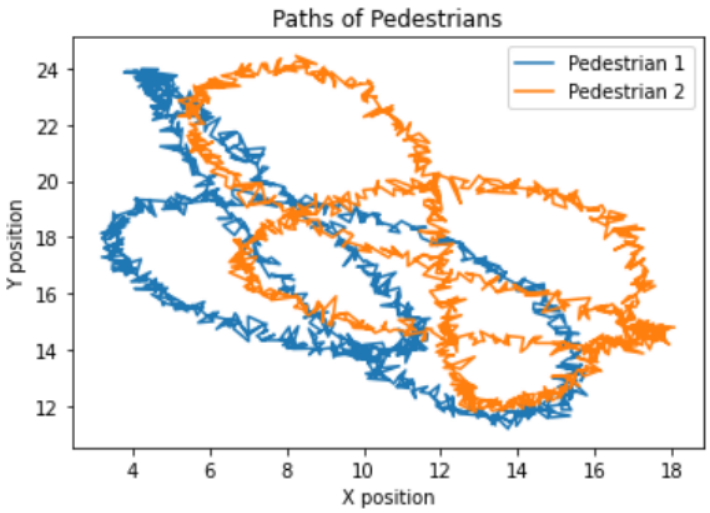


Figure 6: The first two pedestrians' path

2.Are two components enough to capture most of the energy of the data set? In order to answer the question, we need to calculate the energy captured by two components. Based on the code part,we could find out that the energy captured by the first two components is 84.92%. Bacause 84.92% ¡ 90%,then two components alone are not sufficient, and more components would be needed to capture most of the energy.

```
# Perform Principal Component Analysis (PCA)
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data)

# Calculate the percentage of energy captured by the first two components
energy_percentage = np.sum(pca.explained_variance_ratio_) * 100

print(f"Energy captured by the first two components: {energy_percentage:.2f}%")
```

Code part for calculate the captured energy

3.Why, or why not? How many do you need to capture most of the energy? Because two components are not enough to capture most of the energy of the dataset, it means that the information in the dataset is spread across more than just two dimensions. The data may have complex patterns, correlations, or structures that cannot be adequately represented or captured solely by two principal components.

In such cases, using only two components for dimensionality reduction may result in a loss of important information and a reduction in the overall fidelity of the dataset representation.

To capture most of the energy, we need to include more principal components in the analysis. By including more components, we could account for additional sources of variability and capture more intricate patterns or structures present in the data.

So that based on the previous code, we just need to edit the number of components from 2 to 3. The result is 'Energy captured by the three components: 99.71%'.Because the value is more than 90%, we could say that three components are enough to capture most of the energy of the data set.

Answer the three questions from the first page of the exercise sheet.

(a) an estimate on how long it took you to implement and test the method In Task 1, the running time is so fast that it is almost negligible. This is because in Task 1, the data set is relatively small, and the calculation of PCA is relatively simple. We wrote the code and answered the question for about 3 hours.

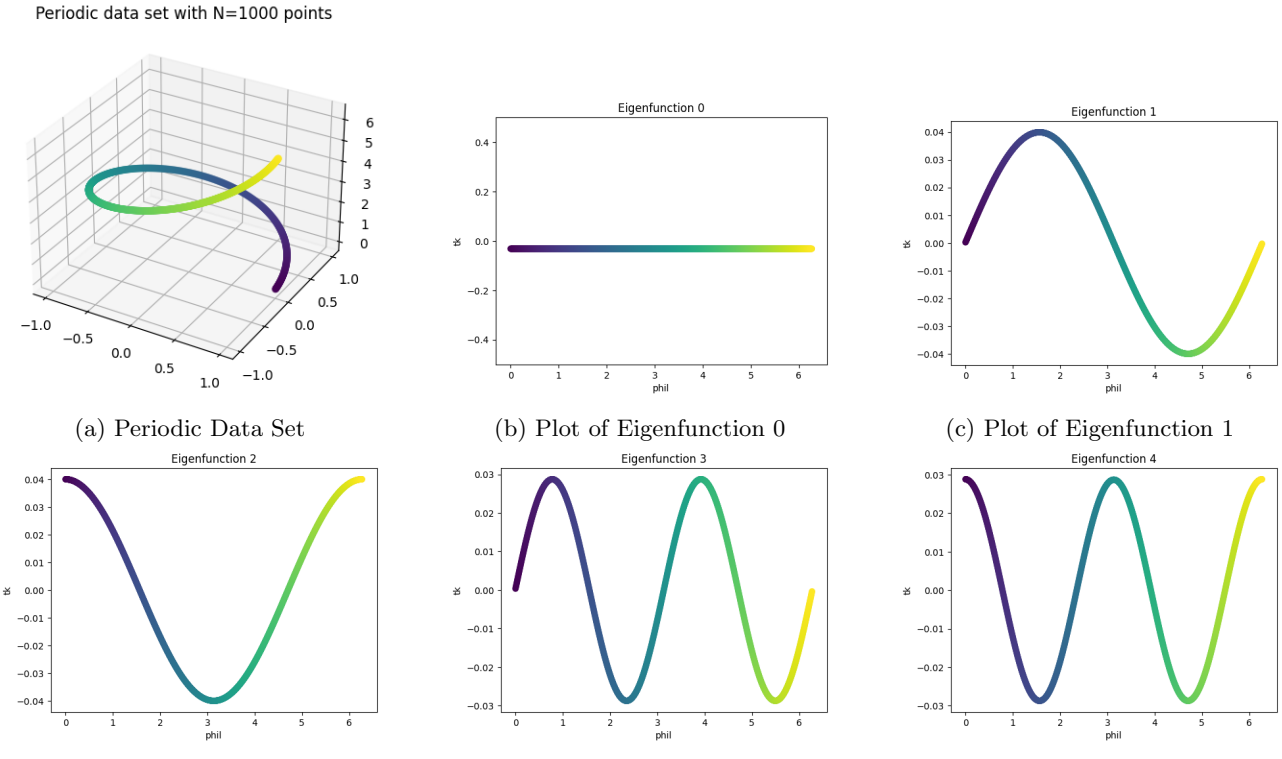
(b) how accurate you could represent the data and what measure of accuracy you used The accuracy of representing the data using PCA can be measured by energy captured by the principal components. A higher captured energy indicates a more accurate representation of the data.

(c) what you learned about both the dataset and the method Implementing and using PCA helps us understand how it can be applied and its limitations. It shows how the number of principal components we choose affects the accuracy of representing the data. We might find that even with fewer principal components, we can still capture a big part of the data's variation. This suggests that some dimensions in the data are repetitive or not very informative. On the other hand, if we need a larger number of principal components to capture most of the energy or variation in the dataset, it indicates that there are complex or multidimensional patterns in the data.

Report on task 2. Diffusion Maps

Part one: Diffusion Maps and Fourier Transform

By using the diffusion algorithm provided in the exercise sheet as a basis, a diffusion map function was implemented in a python notebook. The input periodic data set was created using a numpy array as shown in fig.7 part a. The diffusion coordinates created by using the diffusion map was then used to plot the five eigenfunctions associated to the largest eigenvalues respectively. The eigen functions in diffusion map displays information regarding underlying structure in the data. Higher the eigen value of the eigenfunction, the higher the contribution of that eigenfunction for the diffusion map. As the eigen values get lower, the information provided by the eigenfunction becomes less relevant and hence can be discarded. As only few eigenfunctions can sufficiently represent the dataset, this diffusion map method is very useful in dimensionality reduction.



(a) Periodic Data Set (b) Plot of Eigenfunction 0 (c) Plot of Eigenfunction 1

(d) Plot of Eigenfunction 2 (e) Plot of Eigenfunction 3 (f) Plot of Eigenfunction 4

Figure 7: Plot of Dataset and the Eigenfunctions for Diffusion Map of a periodic dataset

Bonus: Fourier Transform: Fourier transform is a mathematical tool which decomposes the input data into a set of sinusoidal components. The formula for Fourier transform is given below:

$$`F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i\omega t} dt`$$

Mathematically, both fourier transform and diffusion map rely on summation over a exponential time series with fourier transform having a complex (real part and imaginary part) form. But, both Diffusion Map and

Eigenfunction are used for spectral analysis the goal is to decompose the input data into a sum of basis functions. In Diffusion map, the eigenfunctions is used as a basis function and in fourier transform, the sinusoidal components are used as basis function. In the case of periodic data set, both of the mathematical functions returns similar sinusoidal basis functions.

Part2 Swiss roll manifold

First, we need to use the algorithm to obtain the first ten eigenfunctions of the Laplace Beltrami operator on the "swiss roll" manifold, which is defined through the formula below, where $(u, v) \in [0, 10]^2$ are chosen uniformly at random.

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

By using a simple routine in the sklearn Python library named "make_swiss_roll", we can easily generate the swiss-roll data set as shown in Figure 8

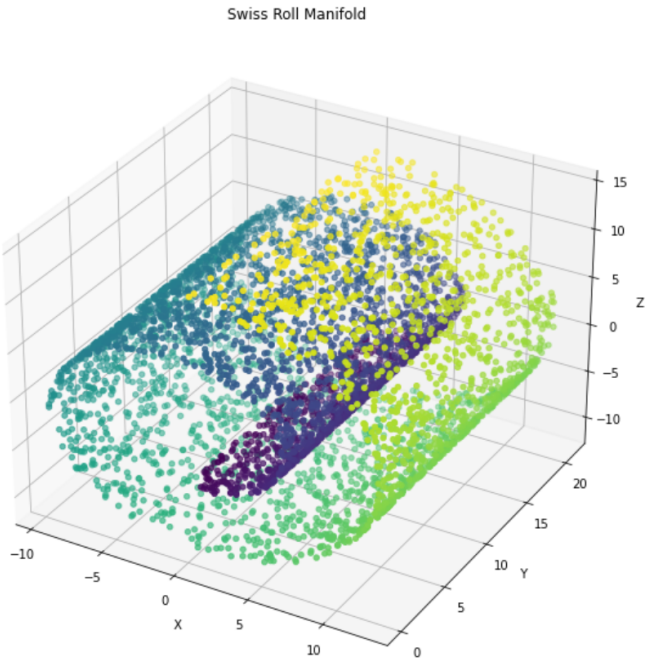


Figure 8: Swiss Roll with 5000 data

After generating the Swiss Roll, we will need to obtain the first ten eigenfunctions of the Laplace Beltrami on the Swiss roll data set and plot them in figures. By plotting them, $\Phi 1$ will always be on the horizontal axis, yet all other Φi are shown on the vertical axis.

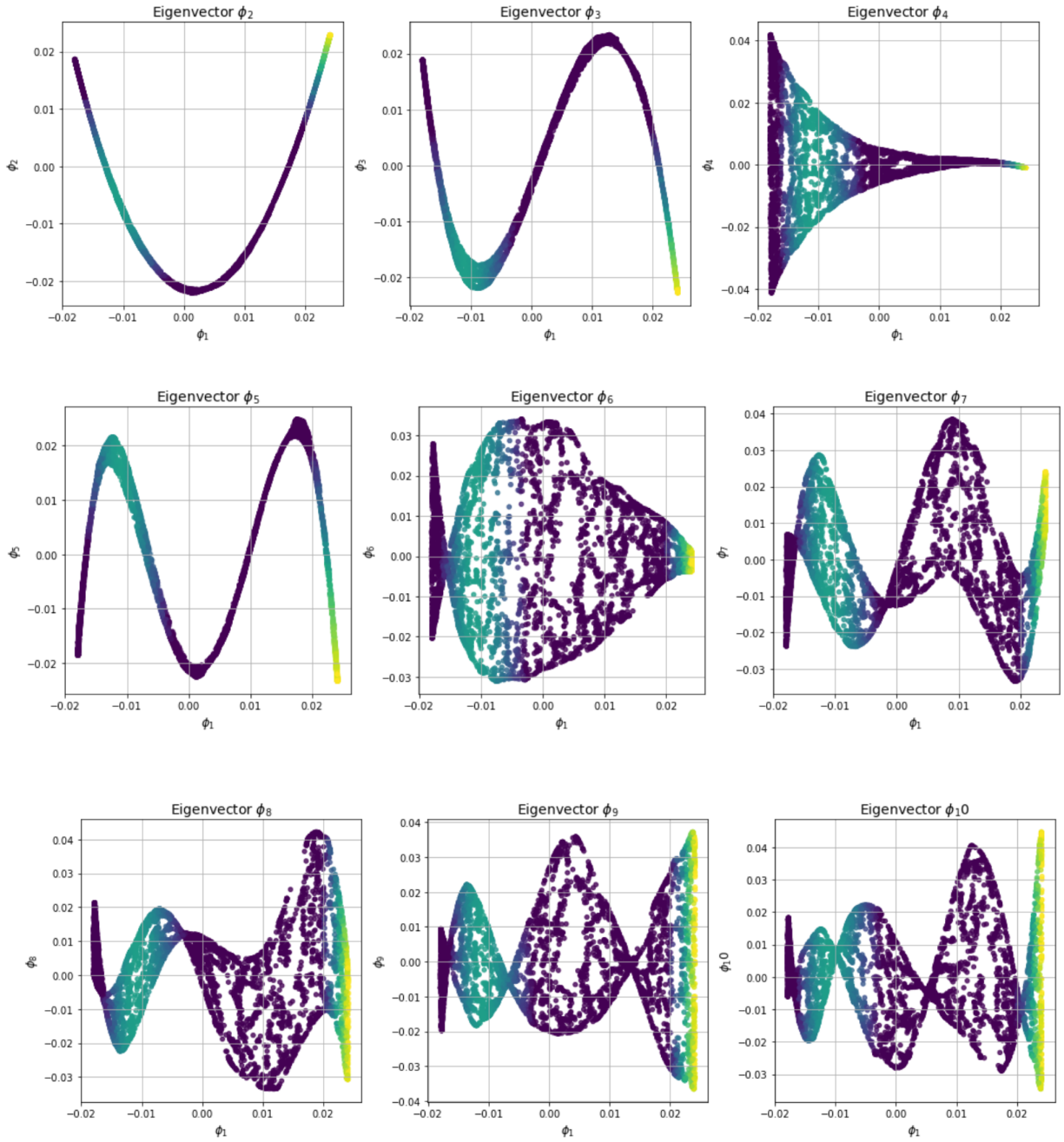


Figure 9: Eigenfunctions from 1-10 with 5000 data

As shown in the Figure 9, eigenfunctions from Φ1 - Φ9 are painted in scatter plots. Then to answer the question: At what value of l is l (l > 1) no longer a function of Φ1, we can try analysing these figures.

Pics of Φ2 and Φ3 are obviously still functions of Φ1, though Φ3 has a few turbulence, we believe that it is because of noises out of sampling. And then from Φ4, it's clearly no longer a function of Φ1. It's also worth to mention that although Φ5 looks similar to Φ3, it actually has more turbulence on curve, which can still be seen as non-functional to Φ1.

So the final answer is, from l=4 is l no longer a function of Φ1.

```
# Perform PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)

# Print the three principal components
print("Principal Components:")
for i in range(3):
    print(f"PC{i+1}: {pca.components_[i]}")
```

Executed at 2023.05.29 17:31:42 in 15ms

```
Principal Components:
PC1: [ 0.47417965 -0.01401482  0.88031656]
PC2: [-0.87493439 -0.11902486  0.46938566]
PC3: [-0.0982012   0.99279236  0.06870122]
```

Figure 10: the three principal components

The next Figure 10 we are showing here is to calculate the three principal components of the swiss-roll dataset. Here we use the method of pca to achieve that and print the result one by one.

It is also asked, why is it impossible to only use two principal components to represent the data. well, generally speaking, a Swiss roll dataset is inherently three-dimensional, for having three coordinates (x, y, z) that determine its position in 3D space. So if performing it in a 2 or lower-dimensional space, it's inevitable to lose some information.

if we reduce the dimensionality to two by using only two principal components, the information about the Swiss roll on the third dimension will be lost. The result would be like a flattened roll in 2D space.

Then the last task of this part is to build a Swiss Roll with only 1000 data. It is shown in Figure 11

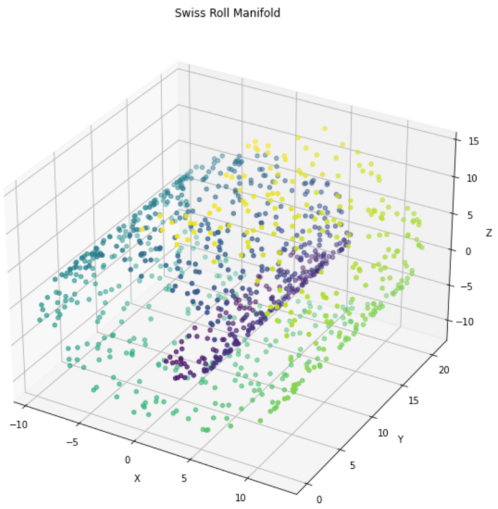


Figure 11: Swiss Roll with 1000 data

Here again, we also place all the eigenfunctions in scatter plots (in Figure 12).

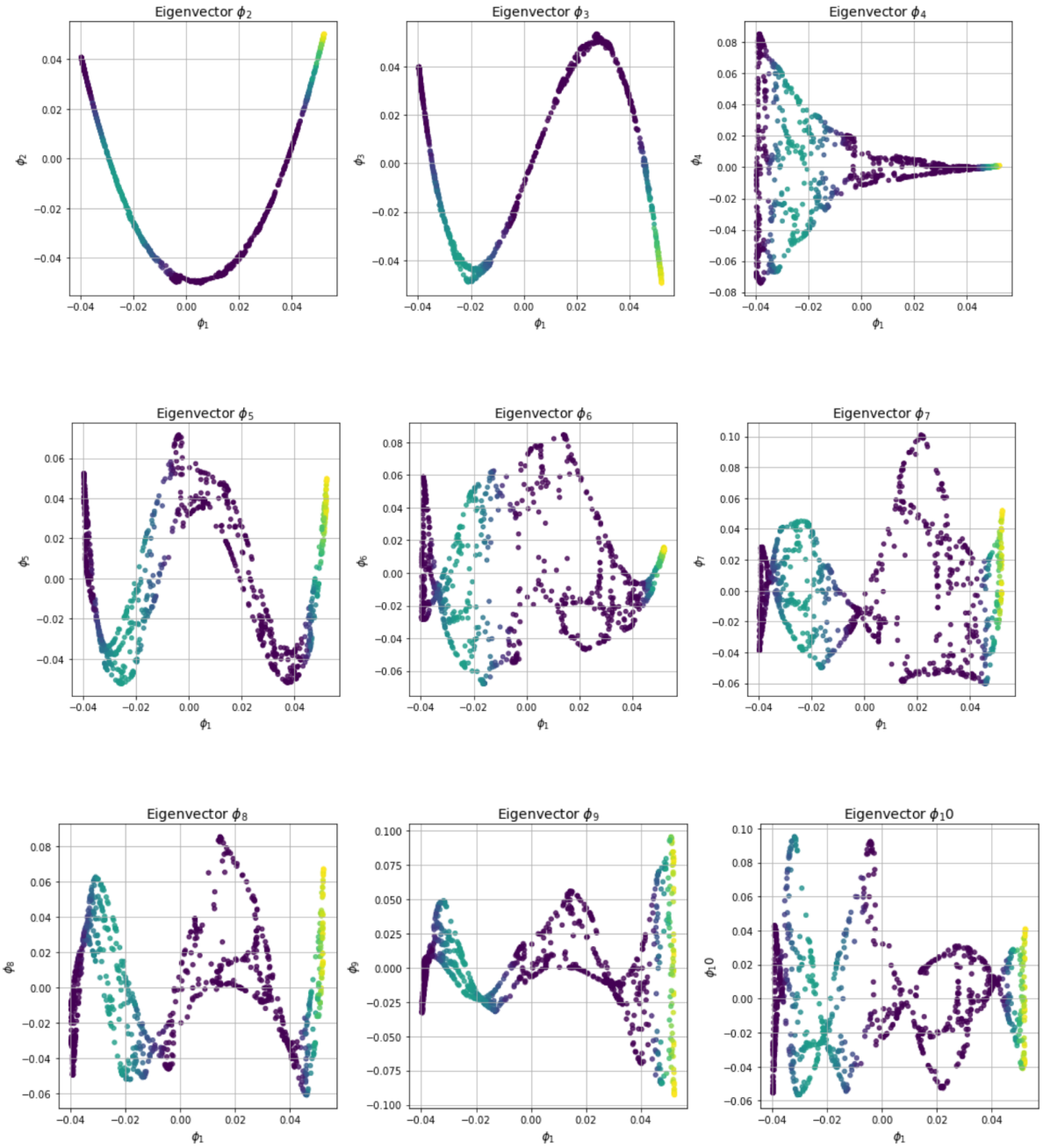
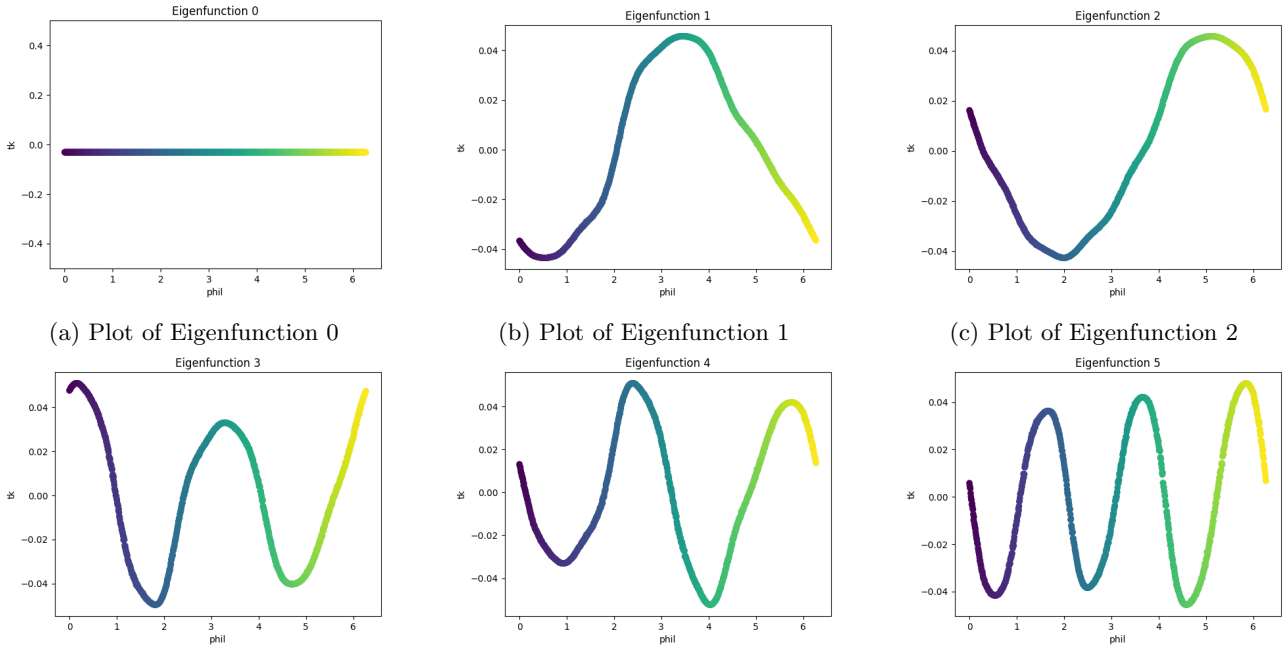


Figure 12: Eigenfunctions from 1-10 with 1000 data

It is to be observed that as the amount of data being reduced, the shapes of the Swiss Roll and all plots of eigenfunctions don't change too much, but the complexity and structure of the original dataset can not be fully captured. it's also to be seen that the noises are indeed getting more obvious. Here even the curve of Φ2 is no longer smooth. But it's interesting to find out that the difference between Φ3 and Φ5 is also getting clearly, which validates our conclusion, that Φ5 is no longer a function to Φ1.

Part three: Vadere Trajectory data and Diffusion Map

Since diffusion maps do not have exactly the same energy interpretation of the eigenvalues as PCA in Task1, a new way must be formulated to determine the necessary amount of eigenfunctions. The trajectory data for two pedestrains are presented in the figure 13. There are also the plot of the largest eigenvalues for the diffusion map of vadere data. To determine the number of eigenfunctions required to accerately represent the vadere data, without any intersecting curves, we can employ various methods. One way is by plotting the eigen values and checking for sudden drops. This will suggest that the eignefunction no longer is necessary to represent the dataset.



(a) Plot of Eigenfunction 0 (b) Plot of Eigenfunction 1 (c) Plot of Eigenfunction 2

(d) Plot of Eigenfunction 3 (e) Plot of Eigenfunction 4 (f) Plot of Eigenfunction 5

Figure 13: Plot of Dataset and the Eigenfunctions for Vadere dataset

Bonus: Swiss Roll and Diffusion Map

Figure 14 shows the 3d Image and the 2d diffusion map of the swiss roll dataset acquired from sklearn dataset.

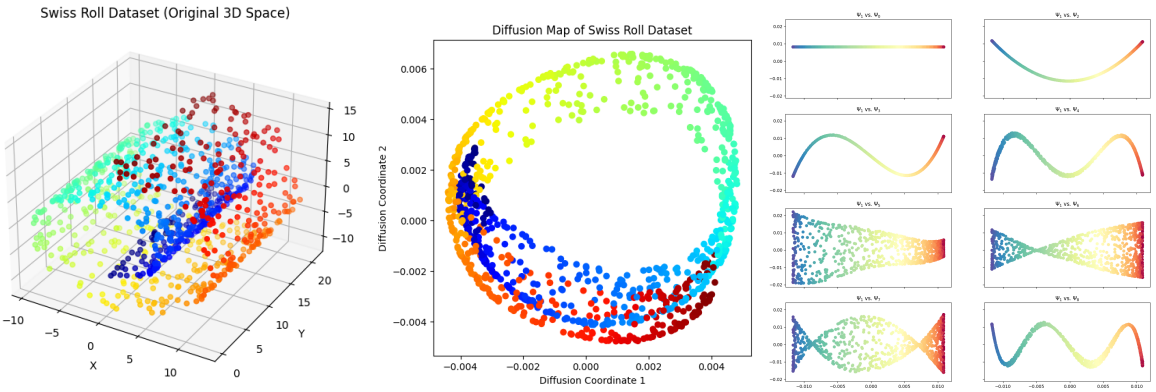


Figure 14: SwissRoll dataset and Diffusion Map with 1000 points

Report on task TASK 3, Training a Variational Autoencoder on MNIST

In this task, we mainly performed the following steps:

1. Select and preprocess the dataset: We selected the MNIST dataset, preprocessed it, including normalizing pixel values, and divided it into training and testing sets.

2. Model building: We build a variational autoencoder (VAE) using a 2D latent space. In this model, we use a multivariate diagonal Gaussian distribution as the approximate posterior q(z—x) and the likelihood p(x—z), and a multivariate diagonal standard normal distribution as the prior p(z).

3. Model training: We train the model using the Adam optimizer and a learning rate of 0.001. The batch size for training is 128.

4. Evaluate and optimize the model: We evaluate and optimize the model according to the following aspects: - After 1st, 5th, 25th, 50th iterations and optimization converged, draw the latent representation, i.e. encode the test set and label the different classes. - At these key points, draw 15 reconstructed figures and the corresponding original figures. - Decode 15 samples from the prior, i.e. generate 15 numbers. - Plot the loss curve, i.e. the number of training iterations vs. ELBO.

5. Training the model with a higher dimensional latent space: We used a 32-dimensional latent space to train the VAE and compared the resulting numbers and loss curves after the optimization converged.

The above steps are mainly to understand and practice the performance of variational autoencoders in image reconstruction and generation, and to observe the changes in model performance in different training stages and different latent space dimensions. Next, let's answer all the questions on exercise sheet:

-What activation functions should be used for the mean and standard deviation of the approximate posterior and the likelihood—and why? For the mean of the approximate posterior, a linear activation function should be used, that is, no explicit activation function is used, because the mean value of the posterior distribution (represented by the encoder) is a parameter whose value can vary in the entire real number domain. This is because we want the encoder to be able to map the input data anywhere in the latent space, positive or negative. Therefore, to maintain this flexibility, we generally do not use explicit activation functions on the mean output of the posterior distribution. For the mean of the likelihood, we used the sigmoid

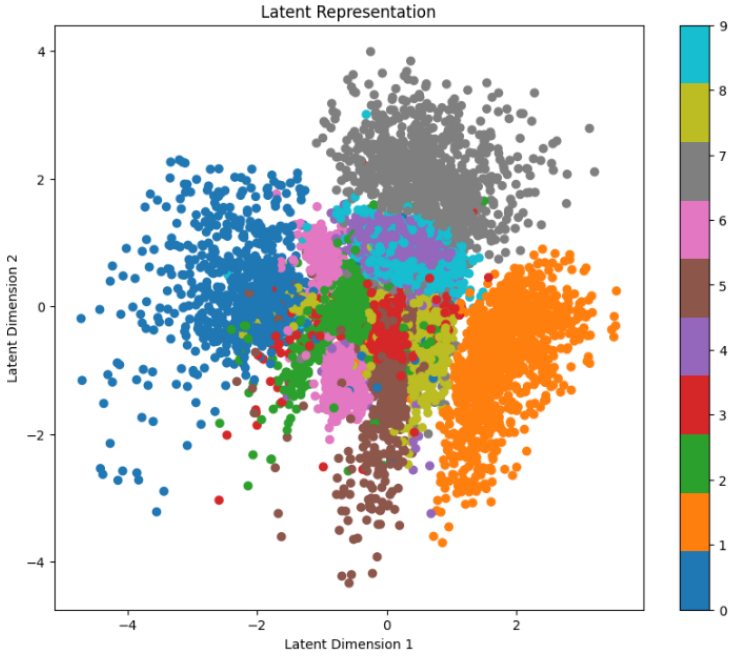


Figure 15: Latent space of 10 digits under the setting of 2D

activation function. Since we are dealing with image data, where pixel values range from 0 to 1, it is appropriate to use the sigmoid function as the activation function. (Of course, TanH may also be used). For the standard deviation of the approximate posterior, the activation function we use is $exp(0.5 * x)$. In this way, we can ensure that the standard deviation is always positive because the output of the exponential function is always positive. For the standard deviation of the likelihood, there is no need to use an explicit activation function, because the

standard deviation of the likelihood function is a learnable parameter and does not need to be derived from the input data through some explicit functional form (ie, the activation function) inferred. This is because in actual image data, the noise level of pixels (i.e. standard deviation) is usually within a certain range and does not change with changes in the image. Therefore, we can set it as a fixed, learnable parameter. In this case, we don't need to dynamically adjust the standard deviation based on the input data, so we don't need an explicit activation function.

-What might be the reason if we obtain good reconstructed but bad generated digits? There are two possible reasons I can think of. One is overfitting on the training set. The VAE has learned to map the inputs to the latent space and back to reconstruct them effectively, but when it comes to generating new digits from random points in The latent space, it fails to generalize because it's relying too much on the specifics of the training data. Another is the incomplete coverage of the latent space. During training, the model may fail to fully explore and understand all possible regions in the latent space. In other words, some places may not be sufficiently represented in the training data, and these regions may be points sampled from the prior distribution. This leads to poor performance of the model in these regions.

-Plot the latent representation, i.e., encode the test set and mark the different classes, as shown in figure15

-Plot 15 reconstructed digits and the corresponding original ones, also plot 15 generated digits, i.e., decode 15 samples from the prior, as shown in figure16,where the first,second and third row show the original,reconstructed and generated digits,respectively.

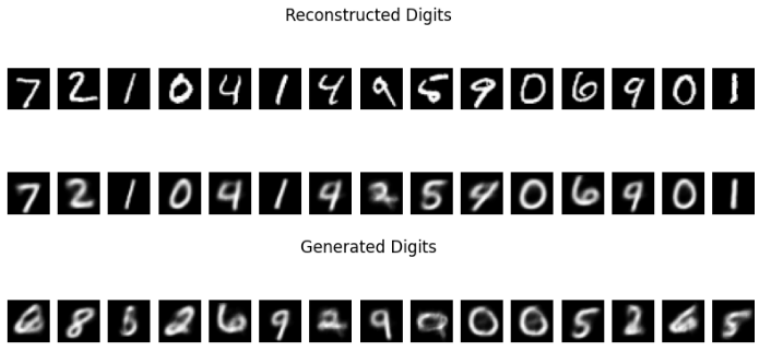


Figure 16: Reconstructed,oringinal and generated digits under the setting of 2-dimensional latent space

-Plot the loss curve (test set), i.e., epoch vs. ELBO, as shown in figure17.

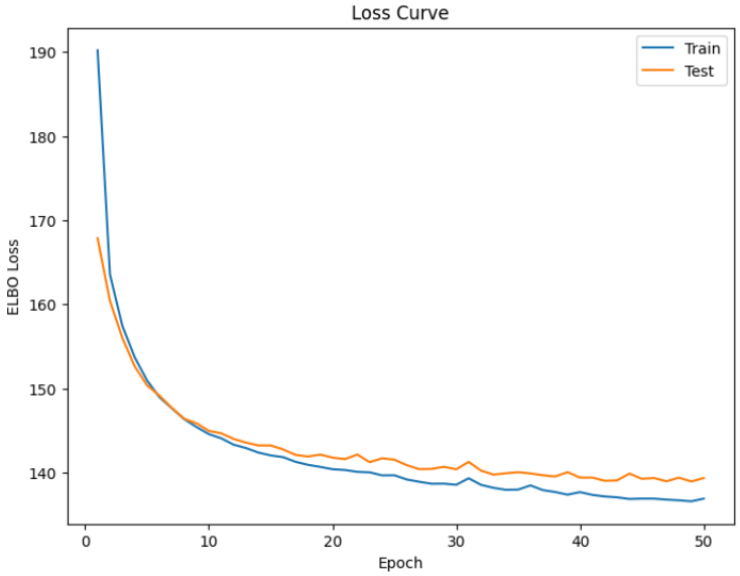


Figure 17: Loss curve under the setting of 2-dimensional latent space

-32-dimensional latent space: Compare 15 generated digits with the results in 3.(b). The 15 generated digits are as shown in figure18.When we compare them to the previous ones, we found that the generated digits from the 32-dimensional latent space are more clear than those from 2-dimensional latent space. This is because that When increasing the dimensionality of the latent space (say to 32 dimensions), VAEs have more space to learn complex representations of the data. This can lead to higher quality images with greater detail, while preserving key features of the digits.

Figure 18: 15 generated digits under the setting of 32-dimensional latent space

-32-dimensional latent space: Compare the loss curve with the one in 4. The loss curve under the setting of 32-dimensional latent space as shown in figure19. As we can see, the loss curve of test set is lower and smoothier than the previous one.The reason is that When using a higher-dimensional latent space, the model can capture more complex data representations. Specifically, more dimensions can enable models to capture more and more complex features and patterns in the data. Therefore, this makes it possible for the model to reconstruct the input image more accurately, thus reducing the reconstruction loss. At the same time, a more complex representation may also make the latent variable distribution closer to its prior distribution, thereby reducing the KL divergence loss. This is why a higher dimensional latent space may result in a lower loss.However, this is not always true, as the increased dimensionality of the latent space can also lead to overfitting, where the model overfits the training data and underperforms on the test data.

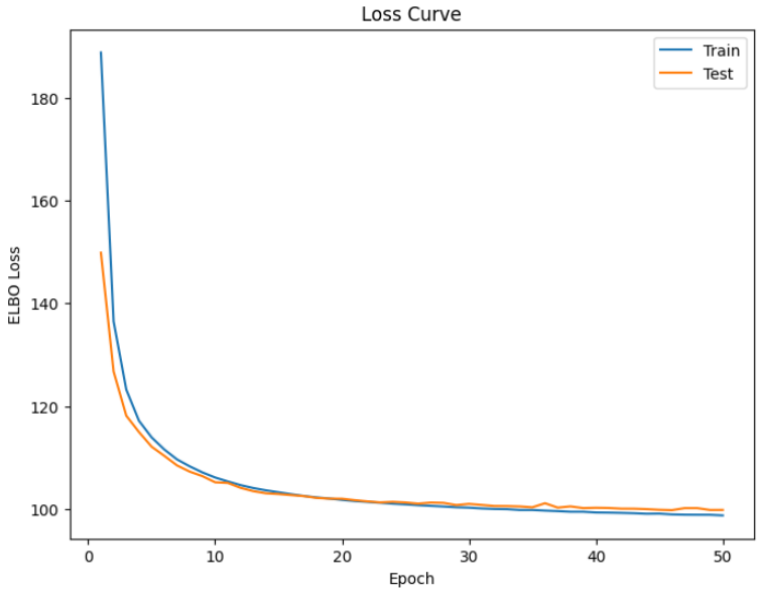


Figure 19: Loss curve under the setting of 32-dimensional latent space

Finally,let's answer the three questions on Page 1.

(a) An estimate on how long it took you to implement and test the method. We spent around 6 hours to implemt the method,and spent around 40 minutes to test it.

(b) How accurate you could represent the data and what measure of accuracy you used. In the setting of 2(and 32)-dimensional latent space,we get 140(and 100) in terms of ELBO loss.We use reconstruction loss and KL divergence loss (together called ELBO loss) to measure the performance of the model. The reconstruction loss measures the difference between the reconstructed image and the original image, while the KL divergence loss measures the difference between the distribution of the latent variable and the prior distribution.

(c) What you learned about both the dataset and the method (which is probably different from what the machine learned). I learned about that although the MNIST dataset is small, it contains

handwritten digits in various styles, which provides opportunities for the model to learn and understand the inherent changes in the data. And the VAE algorithm we use can just learn these internal changes.

Report on task TASK 4, Fire Evacuation Planning for the MI Building

In this task, we want to learn the distribution of people within the MI building p(x). The training data is obtained by tracking 100 random students and employees during the busiest hour on different days. Also, in this hypothetical scenario, a sensitive area in front of the main entrance is defined, where the number of people should not exceed 100.

First, since our data are preserved in two different .npy files, we need to visualize them.(in Figure 20)

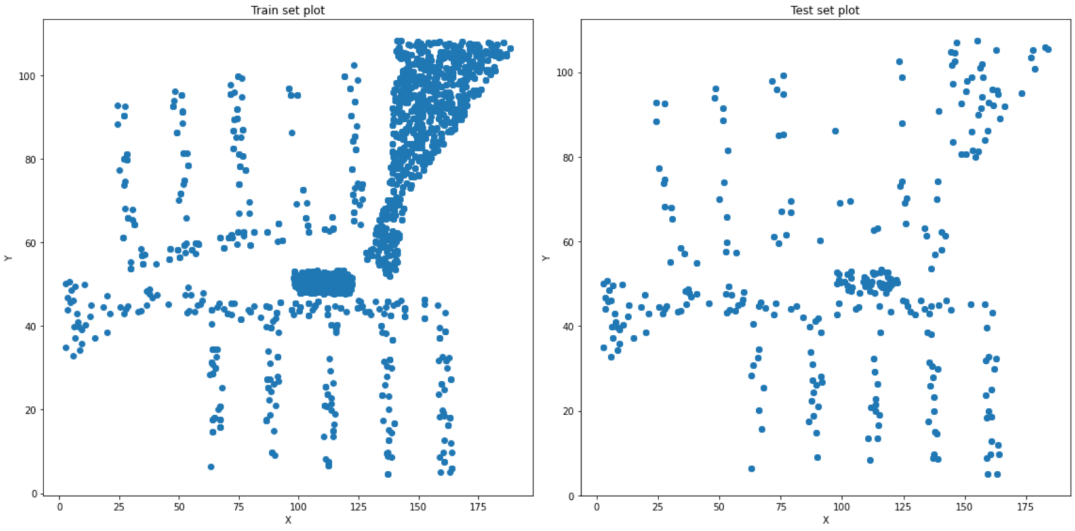


Figure 20: Visualization of data

These two pictures can give us a rough concept of the distributions of the coordinates. Then we reconstruct the VAE in Task3, letting it fit our task. Specifically, we followed part of the instructions in the given exercise sheet, and set the architecture of the encoder to be 2 - 32 - 64 - 128, which means it has 3 hidden layers. Along with it we have two-dimensional latent space with learning rate 0.001 (Adam optimizer) and batch size 128, training for 200 epochs. As for data pre-processing, we normalize the data to have zero mean and unit variance. The following figures 21 and 22 show the loss during training and the results of reconstruction.

```
Epoch 188/200, Train Loss: 0.7041
Epoch 189/200, Train Loss: 0.6861
Epoch 190/200, Train Loss: 0.6961
Epoch 191/200, Train Loss: 0.6879
Epoch 192/200, Train Loss: 0.6992
Epoch 193/200, Train Loss: 0.7021
Epoch 194/200, Train Loss: 0.6964
Epoch 195/200, Train Loss: 0.6959
Epoch 196/200, Train Loss: 0.6847
Epoch 197/200, Train Loss: 0.6861
Epoch 198/200, Train Loss: 0.6831
Epoch 199/200, Train Loss: 0.6836
Epoch 200/200, Train Loss: 0.6994
Test Loss: 0.8938969373703003
```

Figure 21: Loss of the normalized training

Figure 22: Reconstruction of the test set

As shown in the figure 22, the reconstructed data has roughly followed the distribution of original data, especially along the x-axis. But the shape is quite strange since it has a relatively small range of y. By following such a model, the generated data is also shaped like this(Figure 23).

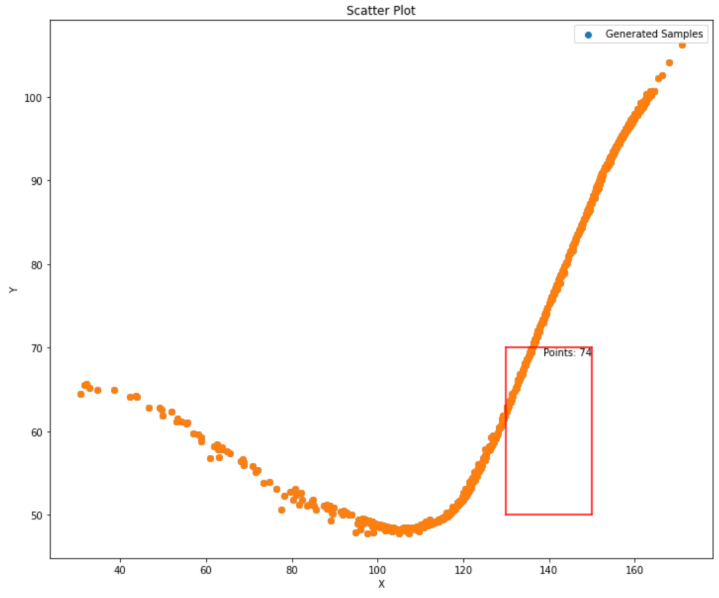


Figure 23: 1000 generated data

Regarding the result like this, we decided to do a comparison experiment. This time, the raw data is used, instead of being pre-processed. This time the loss seems to be a little bit higher(in Figure 24), but the reconstructed data performs so well(in Figure 25), that it over-fits the test data.

```
Epoch 188/200, Train Loss: 6.2409
Epoch 189/200, Train Loss: 6.4918
Epoch 190/200, Train Loss: 6.1624
Epoch 191/200, Train Loss: 7.4425
Epoch 192/200, Train Loss: 10.8618
Epoch 193/200, Train Loss: 8.0543
Epoch 194/200, Train Loss: 6.3003
Epoch 195/200, Train Loss: 6.1587
Epoch 196/200, Train Loss: 6.5113
Epoch 197/200, Train Loss: 6.0650
Epoch 198/200, Train Loss: 6.0624
Epoch 199/200, Train Loss: 6.2771
Epoch 200/200, Train Loss: 6.5147
Test Loss: 7.254914283752441
```

Figure 24: Loss of the raw training

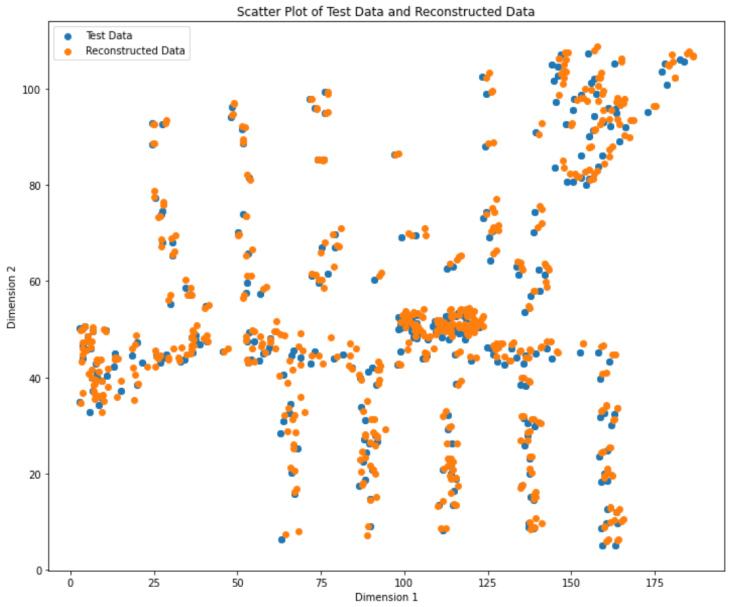


Figure 25: Reconstruction of the raw data

And there is no doubt that with this over-fitting model, the new generated data looks really bad.(Figure 26)

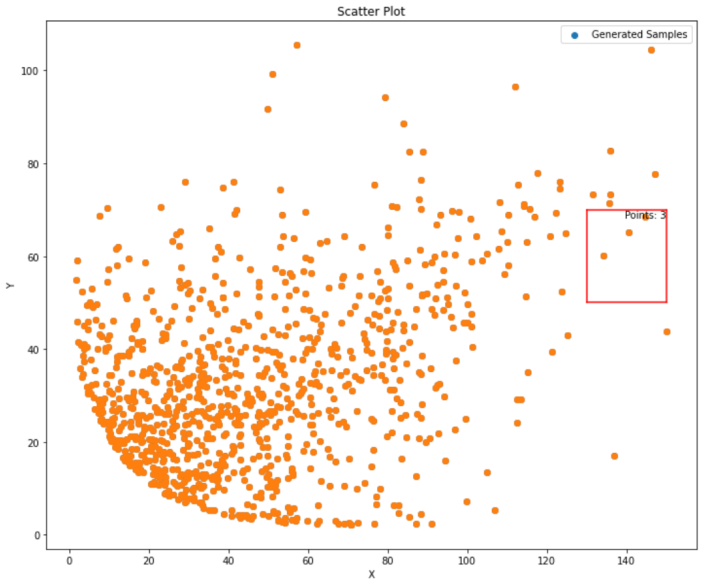
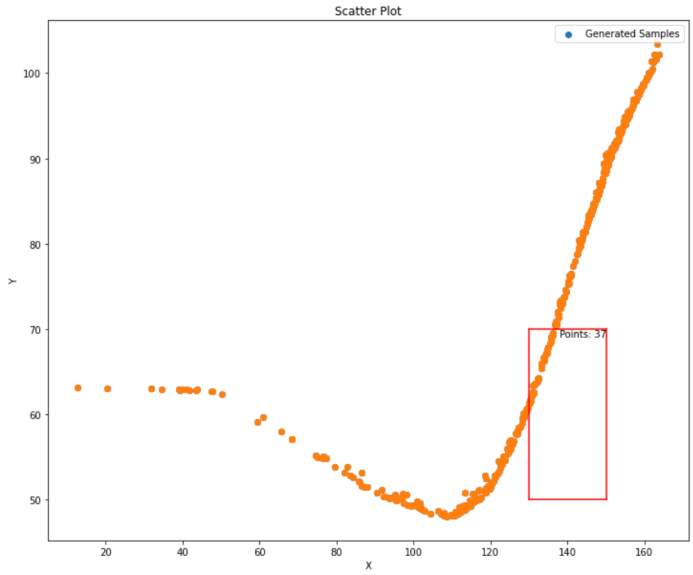


Figure 26: Over-fitting on generated data

So the conclusions of this part are, firstly, the data must be pre-processed, otherwise the training result will be bad, and secondly, there should still be better methods of processing those raw data. But here, we can first go on with our normalized data and results.

Next question is about the threshold of the amount of people in MI building. According to the given conditions, we created a measuring area in the plot, and generated 3 different plots with different amount of people: 500, 1500 and 2500. The results are shown below(Figure 27)



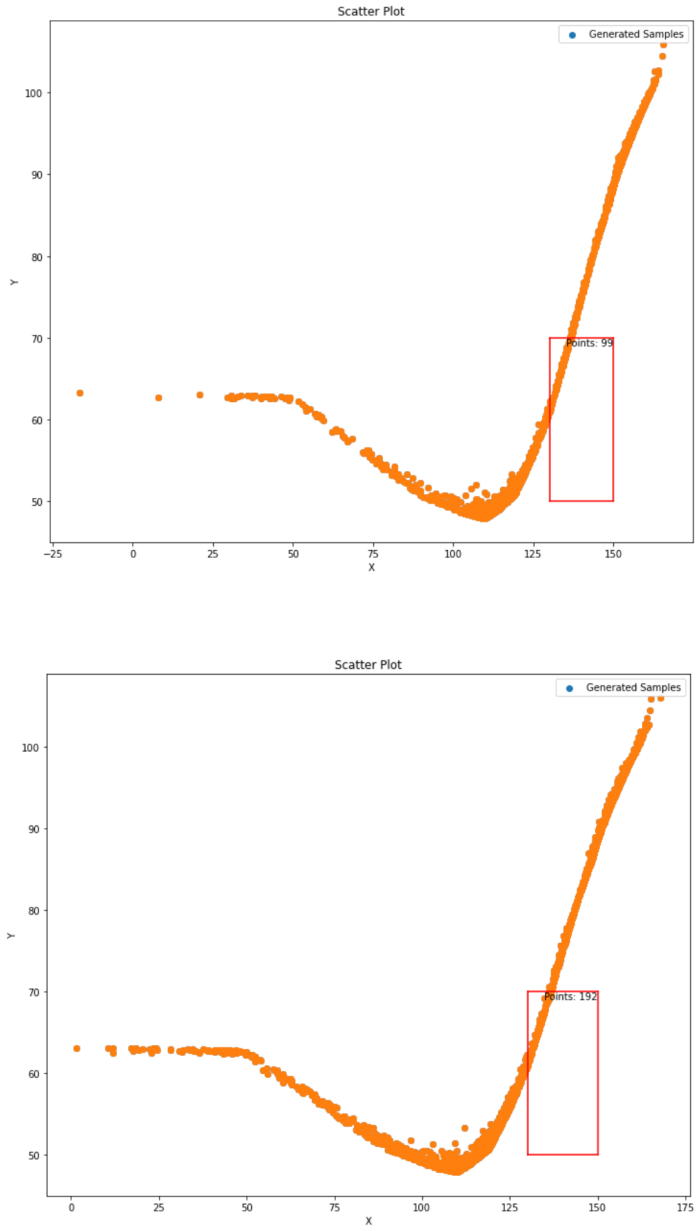


Figure 27: results of measuring area

From them we can tell that approximate 1500 people is the max capacity of MI building in this Fire-Eva-Scenario. (It is also to be seen that our generated data is indeed following some kind of learned distribution, just don't know how accurate it can be.)

Bonus MI Simulation in Vadere

In the bonus part, the learned distribution p(x) is supposed to be used to generate 100 samples, and use them to simulate the Fire Evacuation Scenario in MI building.

It's not hard to generate 100 samples, as shown in task 4, we can generate as many samples as we want and export them into a .CSV file. But when it comes to import it to Vadere, problem shows up. The tutorial on the internet says that we can import .CSV file directly into the Vadere and set them as pedestrians. Unfortunately, we didn't manage to do that in our version of Vadere, maybe it's because we are still not that familiar with it.

So we decided to first build a similar scene in Vadere, and manually split those data in different sources, as pedestrians, and simulate it. The simulating result is in Figure 28

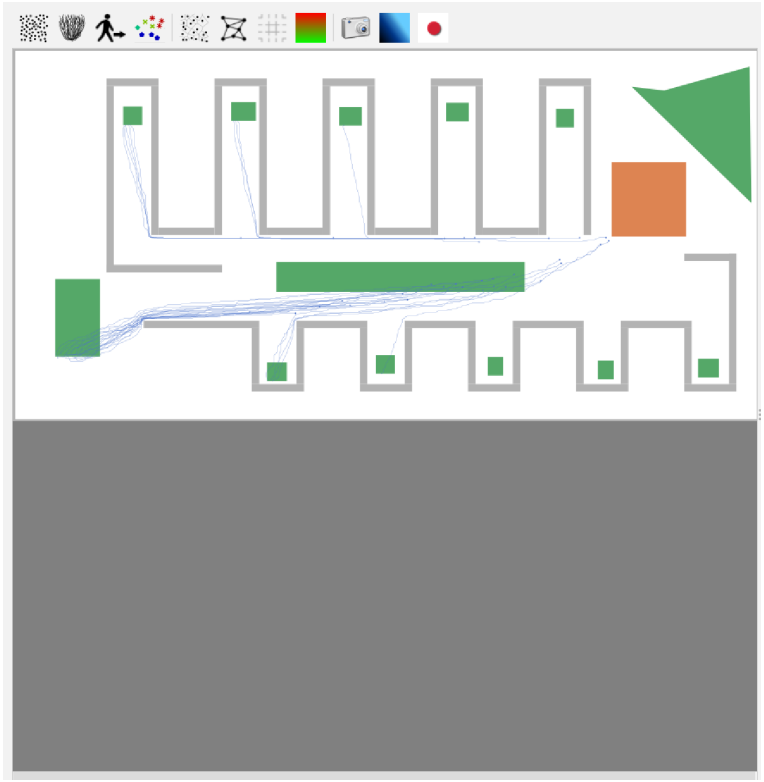


Figure 28: Simulation in Vadere

According to the data distribution we observed in the dataset, the density of people in the upper right and lower part of the scene is higher. In the simulation, it can be seen by observing the trajectory map that the crowd can easily reach the target point due to the open terrain and no obstacles on the upper right. On the other hand, due to the dense obstacles in the lower left of the building, most of the trajectories of the crowd overlap there. What's more, they are farther away from the target point. So from a practical point of view, a building must have multiple exits to meet the evacuation standards. (In fact, the MI building does have multiple exits)