Report for exercise 3 from group E

Tasks addressed:	3
Authors:	Zhitao Xu (03750803) Çağatay Gültekin (03775999) Gaurav Vaibhav (03766416) Minxuan He (03764584) Vatsal Sharma (03784922)
Last compiled:	2023–11–30
Source code:	https://gitlab.lrz.de/mlcms-ex-group-e/mlcms-ex-group-e

The work on tasks was divided in the following way:

Zhitao Xu (03750803)	Task 1	67%
Çağatay Gültekin (03775999) **Project lead**	Task 3 Task 4	20% 100%
Gaurav Vaibhav (03766416)	Task 2	67%
Minxuan He (03764584)	Task 3	80+task 4bonus%
Vatsal Sharma (03784922)	Task 1 Task 2	33% 33%

Report on task 1, Principal Component Analysis

Real world data often has many features, which means they are high-dimensional data in many cases. Except that the high-dimensional data is hard to visualize, however, high dimension can also be very challenging and cause problems in many other aspects. For example, similarity search or distance computation will require expensive resource if we have high-dimensional data. Additionally, high-dimensional data could always lead to correlations between different dimensions, posing challenges for algorithms such as Naive Bayes. This algorithm operates on the fundamental assumption that features are conditionally independent given the class label. When this assumption does not hold due to feature correlations, the performance of Naive Bayes can be adversely affected. Besides, highly correlated dimensions will also result in that the data lies on a low-dimensional manifold embedded in a high-dimensional space, which we can make use of. We can first transform the data and then remove the dimensions with low variances. Through this way, we can reduce the dimension while avoiding too much information loss. We can achieve this goal by using Principal Component Analysis (PCA), which is introduced in detail with an specific example in the following.

1.1 A simple example of PCA

In this part, we will use PCA to process the provided two-dimensional data in the `pca_dataset.txt` which contains 100 data points. We first load the data and store them as a two-dimensional Numpy array. We define a function to implement PCA, in which we center the data by subtracting the mean from the data points to make the center of the data located at the origin. After that, we apply the singular value decomposition (SVD) to the centered data matrix, obtaining left singular vectors `U`, singular values `S`, and right singular vectors `Vh`, respectively. The right singular vectors `Vh` contain the new bases after the transformation or **the principal components**. The left singular vectors `U` represent the new coordinates of each data point of the principal components. The singular values `S` give the variance or the "energy" of each principal component. The larger the "energy" is, the more information the corresponding principal component contains. Here, the computation result of `S` is $[9.94340494, 0.82624201]$, denoting the "energies" of the two principal components. As we can see that, the energy of the first principal component accounts for $9.94340494/(9.94340494 + 0.82624201) = 92.3\%$. Therefore, we can remove the second principal component to achieve the dimensionality reduction. The raw data with the directions of the two principal components are shown in Fig. 1. Applying PCA is equivalent to applying a linear transformation to the centered data to map these data points to a new 2-d space spanned by the two principal components as shown in red and green line segments in Fig. 1. We can find out that, the data points are distributed widely along the direction of the first component, while closely clustered along the second one. So in the second dimension, the coordinates are very close to each other, and thus cannot provide much information for us to distinguish different data points, and consequently we can remove it without too much information loss. We can represent the data points in an 1-d linear space spanned by the first component $[-0.88938337, -0.45716213]$. The points are represented by 1-d coordinates while keeping the most of the information from the raw data.

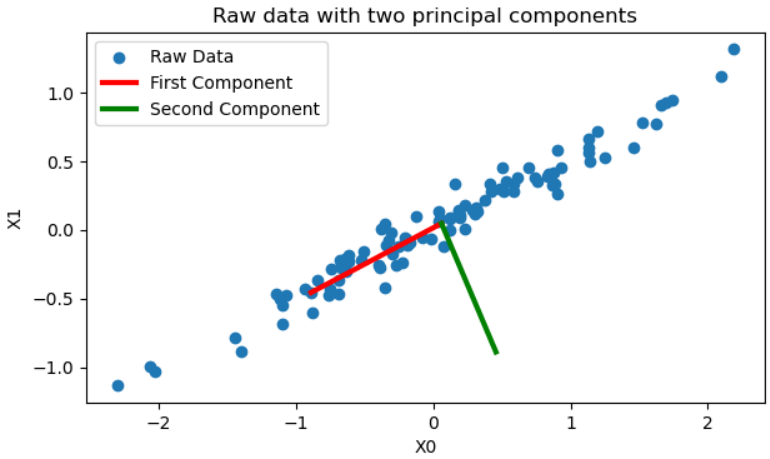


Figure 1: Visualization of the raw data with the two principal components.

1.2 Applying PCA to an image

In this part, we will apply PCA to an image of raccoon. We download the image and put it in the code directory. We convert the image to gray-scale, and resize it to 249×185 as required. We take the columns of the image as our data points, so we have 249 data points with dimension 185. In order to keep the code modular, we define a class to implement PCA in `pca.py`. This class achieves one argument which is the data matrix to be processed in the form of 2-d Numpy array of the shape (N, d), where N is the number of the data points, and d is the dimension of each point. The PCA class also includes the following methods:

- Method `svd`: In this method, we first center the data, and then apply SVD. This method returns the singular vectors and singular values.
- Method `reconstruct`: In this method, we reconstruct the data matrix using the given number of principal components by calculating $X_r = US_rV^T$, where S_r is a square matrix with the upper left diagonal elements to be the given number of singular values and the remaining to be 0. This method returns the reconstructed result.
- Method: `energy_loss`: This method calculates at least how many principal components do we need to make sure the energy loss is lower than a given number p which is between 0 and 1.

After that, we begin to process the image. We store the image as a 2-d Numpy array, and then create a PCA instance passing the image array as the argument. We call the `reconstruct` method to compute the reconstructed data, with specified number of components. We reconstruct the image with all, 120, 50, and 10 principal components respectively. Besides, to illustrate the information loss more intuitively, we also use the Canny edge detector to detect the edges in the reconstructed images. Since the images are in the form of 2-d array, we use Frobenius norm of the difference of the reconstructed images and the original image as a measurement for reconstruction error. The lower this error is, the closer the reconstructed image is to the original image. The results are shown in Fig. 2.

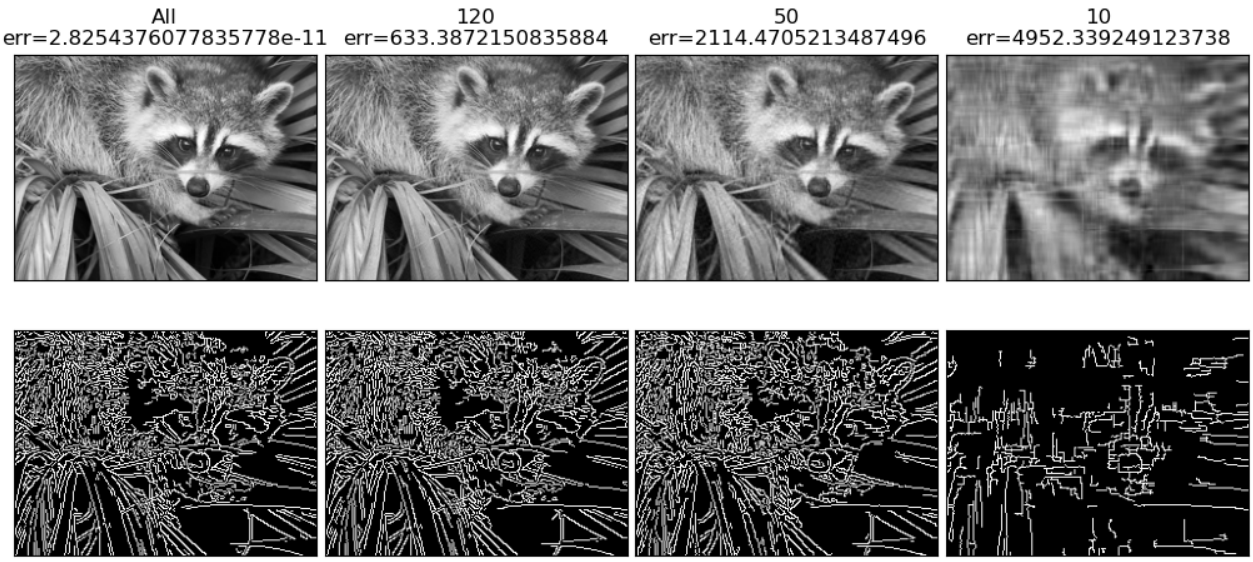


Figure 2: First row: Reconstructed images with all, 120, 50, and 10 principal components. Second row: Detected edges in corresponding reconstructed images.

From the Fig. 2, we can see that, the reconstructed image with all components are exactly the same as the original image, and the error is extremely small. For the image reconstructed with 120 components, it is hard to discern the differences compared to the original image with naked eye. We can observe that the image reconstructed with 50 components becomes a little blurry, but the most of the features still remain. For the image reconstructed with only 10 components, we can observe significant information loss, and the image is very blurry. As a result, at the number of 10 components is the information loss visible. The second row of the

Fig. 2 can also prove this conclusion. The first 3 edge images are almost the same, while for the last image, only part of the edges can be detected.

Finally, we call the method `energy_loss`, and set p to 0.01 to compute how many principal components do we need to maintain at least 99% of the energy. The result is 165, so we need at least 165 components to make sure that the energy loss is smaller than 1%.

1.3 Applying PCA to the trajectory data

In the last part, we will use PCA to analyze the trajectory data of 15 pedestrians over 1000 time steps in a 2-d space. Similarly, we load the data and store them as a 2-d Numpy array. We also define a function `visualization` to visualize the trajectories of pedestrians more conveniently. We call this function to draw the trajectories of the first two pedestrians shown in Fig. 3. We observe that, from a smaller view, the pedestrians change direction frequently, and this results in their trajectories being very tortuous. While from a larger view, the trajectory of each pedestrian forms a loop with a topological structure called **Trefoil knot**, the 2-d shape of which is shown in Fig. 4. Furthermore, we visualize trajectories of all pedestrians, finding that the trajectories all have such shape.

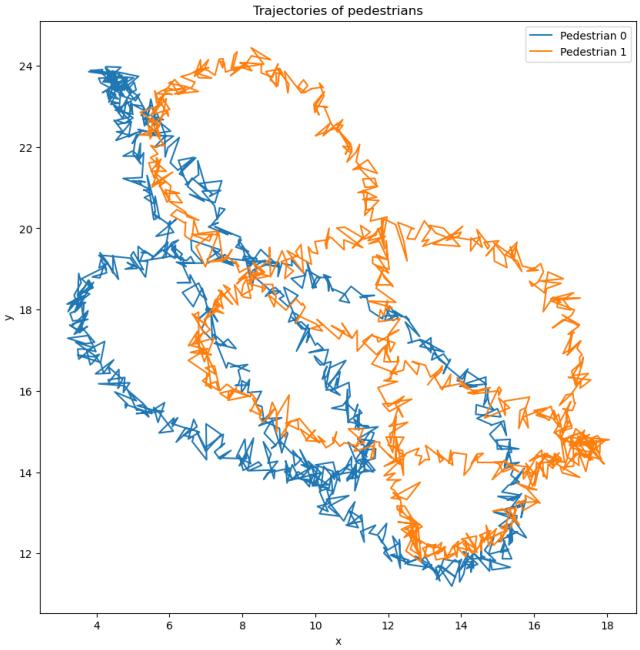


Figure 3: Trajectories of the first two pedestrians.

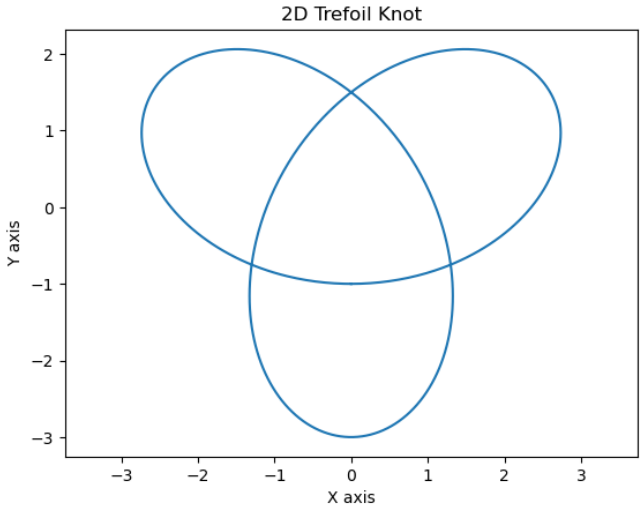


Figure 4: Shape of 2D trefoil knot.

We consider the whole data set as 1000 data points with 30 dimension, and project them to the first two components. We define a `PCA` instance as mentioned before and pass the whole data set. We calculate the singular values of the matrix and find that the first two singular values only account for 66.28% of the whole energy, so it's not enough to use only 2 components to capture most of the energy. Then we call the `energy_loss` method to compute the least number of components needed to capture 90% of the energy, and it gives us 11. Fig. 5 shows us the original trajectories of 15 pedestrians and reconstructed trajectories with 11 and 2 components, respectively. Comparing the left and the middle figure, we can find that, except that the trajectories become a little smoother, the shape and position of trajectories are precisely preserved, confirming our opinion that 11 components can capture most of the energy. Comparing the left and the right figure, we can see that, the trajectories are much smoother. Besides, the position changes, and the shapes are distorted. However, the topological structure of the trajectories, trefoil knot, is preserved. So we can conclude that, though the first two components cannot capture most of the energy, they contain the information of topological structure of the trajectories under this scenario. Here we also use the mean of the L1 norm of the difference between the original data and reconstructed data to estimate the reconstruction error. The error of the reconstructed data matrix with 11 and 2 components are 1.00 and 0.12, respectively.

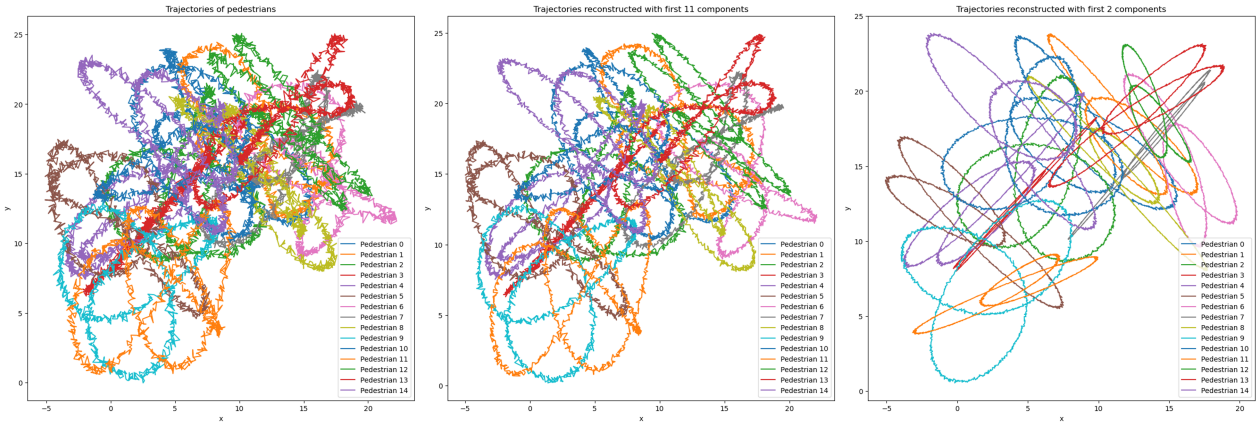


Figure 5: Comparison of different trajectories.

1.4 Three questions on the exercise sheet

1. I used about 2 days to finish the implementation and the test of the methods in this task;

2. I represent the data with the given number of principal components as required in the exercise sheet, and estimate the error with the Frobenius norm or L1 norm of the difference between the reconstructed data matrix and the original one. The specific values of the error are introduced in respective part in the report;

3. Through this task, I found that, under most circumstances, reconstructing the data with 90% of the energy can give us good enough approximations, thus we can reduce the high dimension of the data to a relatively low dimension, e.g. 185 to 101 in part 2, and 30 to 11 in part 3. This also confirms that high-dimensional data are always correlated. Additionally, from part 3, we know that, different principal components encode different types of information of the original data. Principal components with higher energy tends to encode more general information of the data (e.g. topological structure). As the energy gets lower, the corresponding components capture detailed information (e.g. precise position).

---

2. Diffusion Maps

2.1 Similarity of diffusion maps and Fourier analysis

For the given input signal with N=1000 points

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

the input signal is plotted in fig 6. It can be clearly seen that the signal is circular waveform.

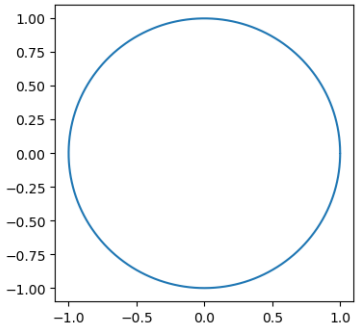


Figure 6: Visualization of input signal

Then, the eigenfunctions corresponding to the five largest eigenvalues of Laplace-Beltrami operator are computed with the diffusion map algorithm. These are shown in the fig 7

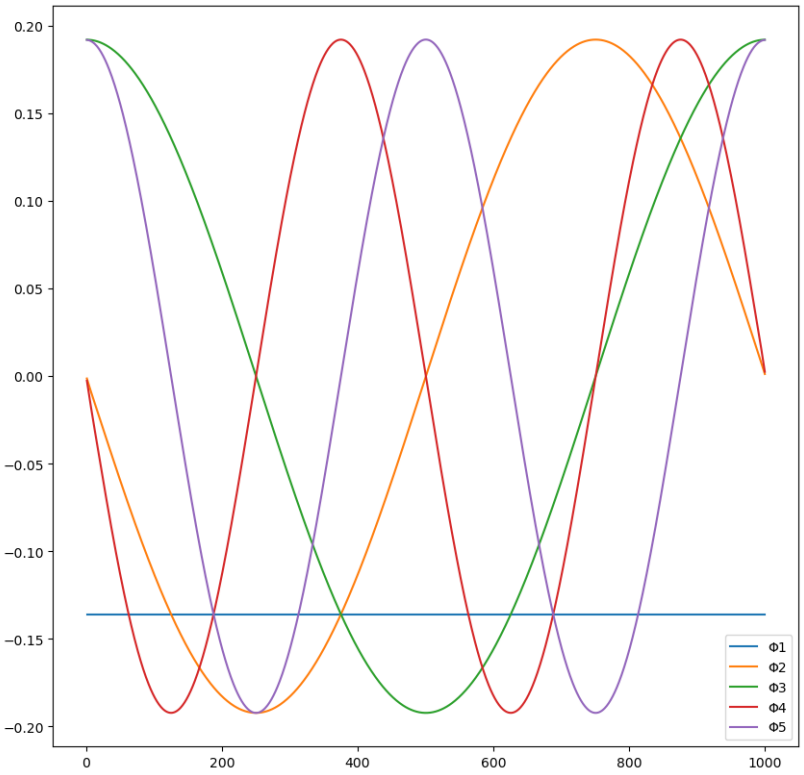


Figure 7: ϕ_l against t_k

As the normalized kernel matrix is row stochastic, first eigenvalue as expected is 1 with constant eigenfunction (trivial). Other four eigenfunctions as in fig 7 are sinusoidal waves sin(t), cos(t), sin(2t), cos(2t). As widely

known in the fourier analysis of signals, any periodic function can be expressed as a sum of sine and cosine functions.

$$f(t) = a_0 + \sum_{n=1}^{\infty}[a_n cos\frac{2\pi nt}{T} + b_n sin\frac{2\pi nt}{T}]$$

As we can see the obtained eigenfunctions sine and cosine are the components of this circular waveform signal for n=1,2.

2.2 Diffusion Maps of Swiss Roll Data

The sklearn Python library is used to generate the swiss-roll dataset with 5000 data points in three-dimensional space, with no additional noise. The visualization of the same has been illustrated in fig [8]

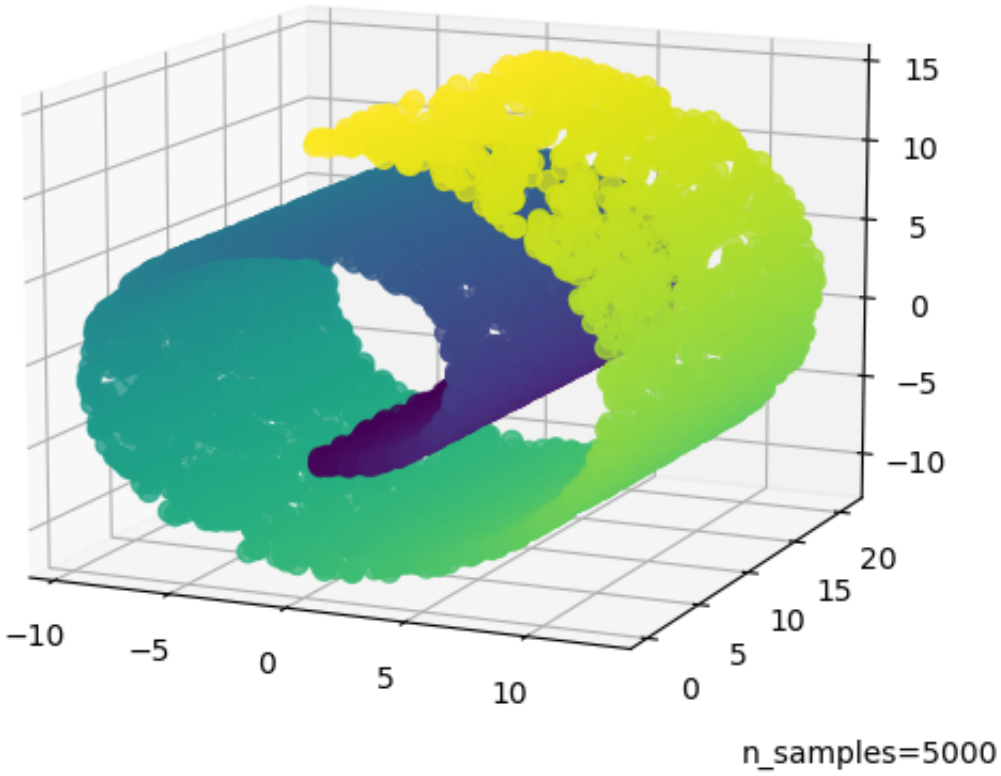


Figure 8: Visualization of Swiss Roll Dataset

Ten Eigenfunctions of the embedding space for the swiss roll dataset were computed using diffusion map algorithm. All these were plotted against first non-constant eigenfunction ϕ_1 shown in fig [11.

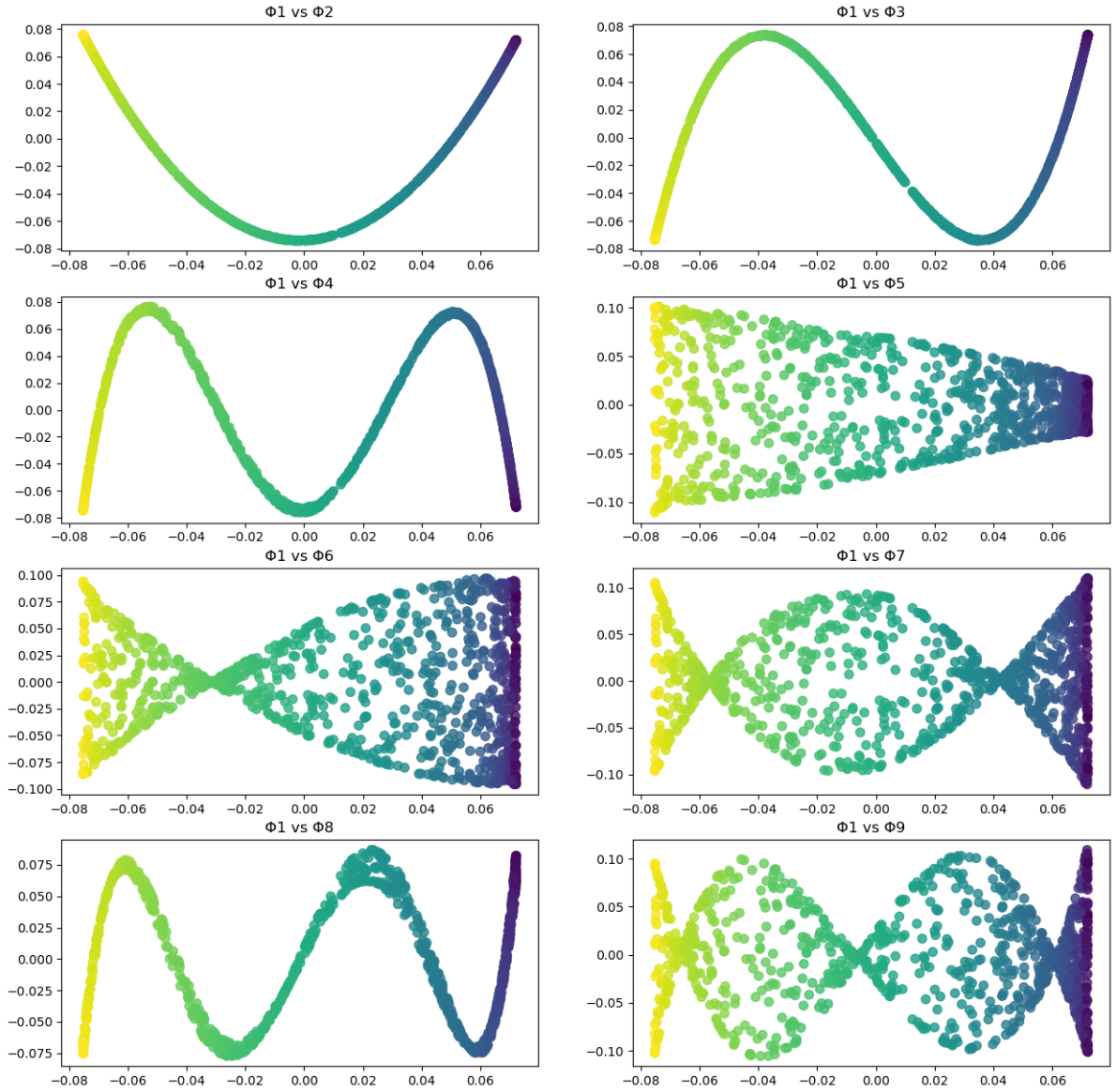


Figure 9: Eigenfunctions plot vs ϕ_1

The first eigenfunction ϕ_1 is constant (property of row-stochastic matrix) and can be ignored. The eigenfunctions ϕ_2, ϕ_3, and ϕ_4 show functional dependence with ϕ_1 and hence, doesn't add a new/independent direction. Whereas ϕ_5 seems to be relatively independent of ϕ_1 and so, pair of ϕ_1 and ϕ_5 give a good two-dimensional embedding space for this dataset. This is illustrated in fig [10]

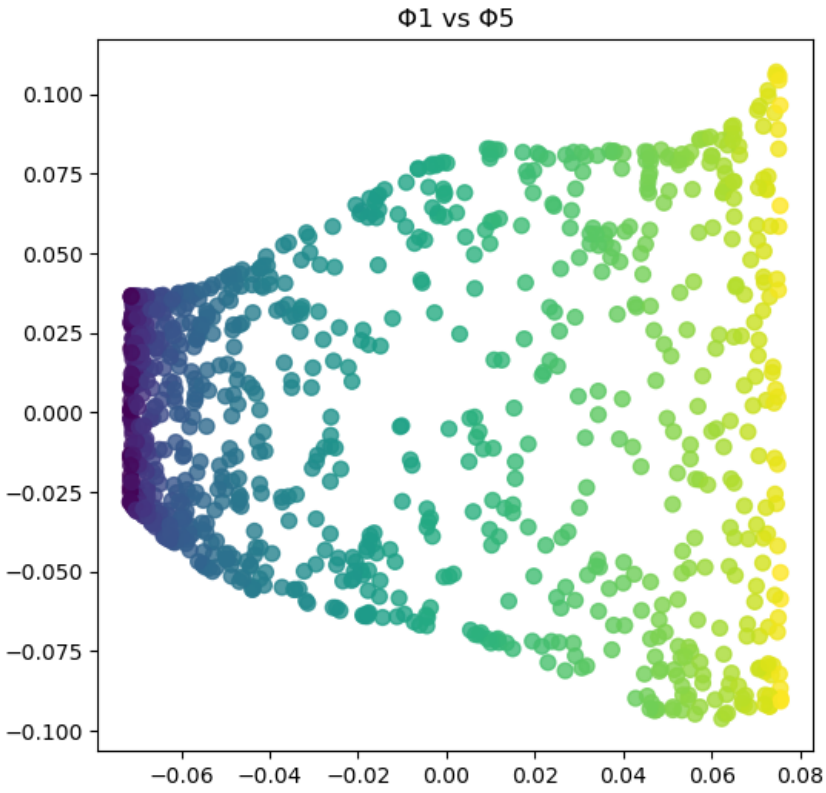


Figure 10: ϕ_1 vs ϕ_5

With just 1000 points of swiss roll dataset, the quality of the embedding space got worsened. This is quite apparent in fig. [11. This might be attributed to the fact that diffusion maps rely on local relationships between data points and so may struggle to estimate the global structure with limited data points. Hence, the effectiveness of these manifold learning techniques which aim to capture the intrinsic geometry of the data can be influenced by the number of data points.

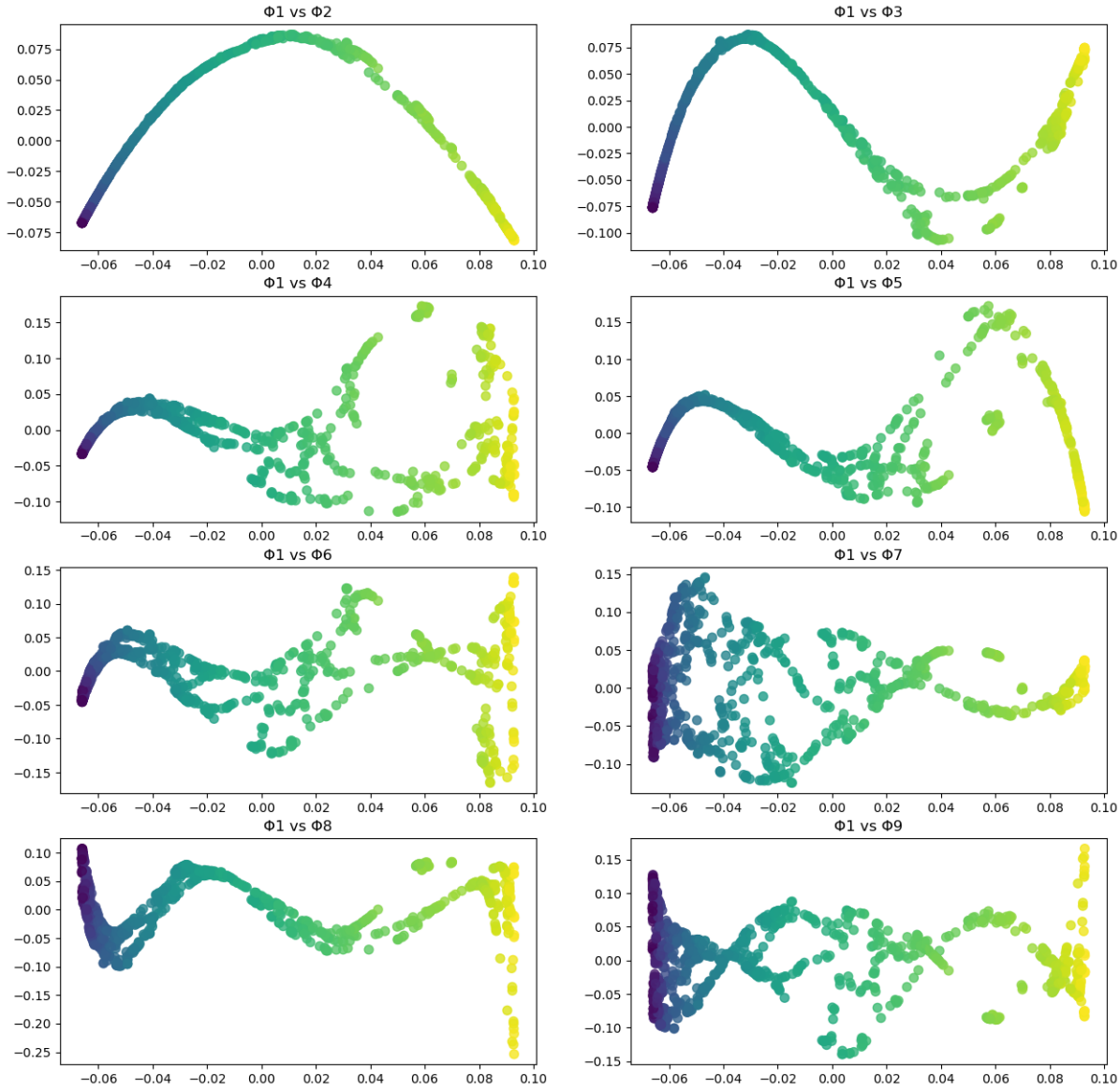


Figure 11: Eigenfunction plots for 1000 data points

2.3 PCA on Swiss Roll Dataset

PCA analysis was carried out on Swiss Roll dataset and three principal components was required to incur maximum loss of 10 percent. It is impossible to represent the Swiss Roll using two principal components as it is getting stretched comparatively in all the three dimensions i.e. variances in three dimensions are quite comparable with each other. These three components were evaluated and plotted as shown in the figure fig [12]

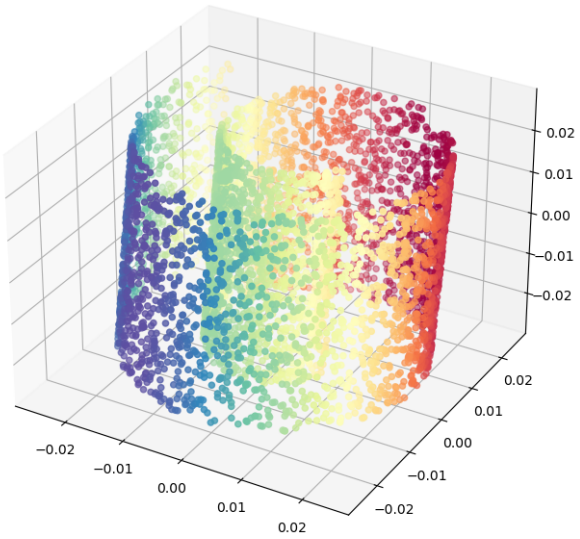


Figure 12: Principal components of Swiss Roll dataset(5000 points)

The same analysis was also carried out for 1000 points as shown in fig [14]

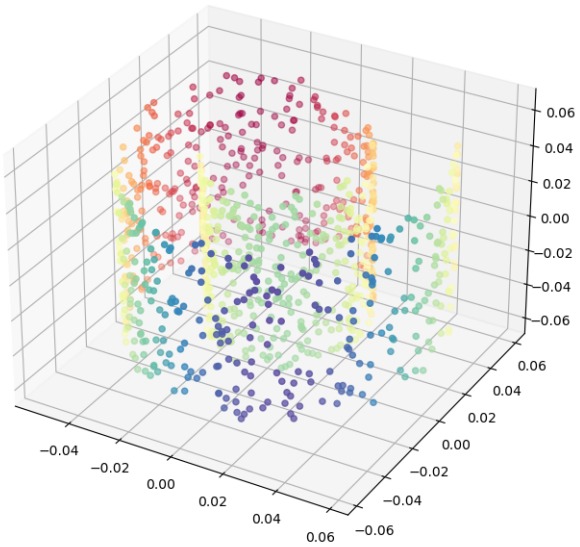


Figure 13: Principal components of Swiss Roll dataset(1000 points)

2.4 Diffusion Map on Trajectory Data

The trajectory dataset is a (1000 x 30) shaped matrix, where every row corresponds to 1 single timestep. And every row contains the x and y coordinates of all 15 pedestrians. For this dataset, we again deploy our custom dmaps_nn_dist() function and calculate the largest eigenvalues and the eigenvectors associated with them. We select 500 random trajectory data rows for visualisation purposes. The first constant eigenvector is dropped and the plots below show the mapping of $\Phi 1$ with the subsequent eigenvectors.

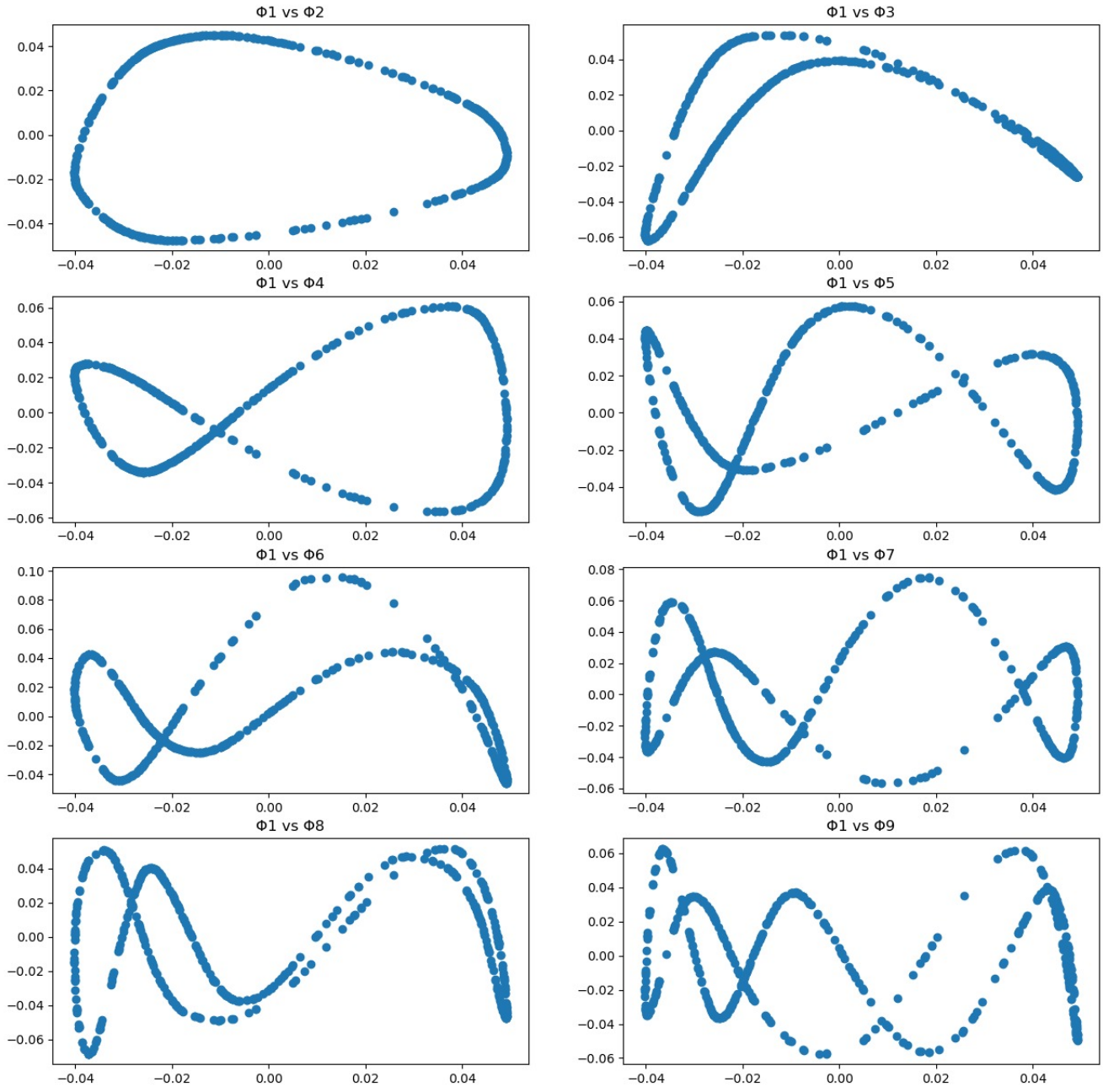


Figure 14: Embedding Space of Φ1 versus other eigenvectors

We can clearly see that the inclusion of the fourth eigenfunction introduced intersections in the curve. Hence **three eigenctions** are enough to accurately represent the dataset and its intrinsic geometries.

2.5 Bonus

The datafold software was downloaded and installed. Diffusion map is implementated on swiss roll dataset using this module. The eigenfunctions plot has been shown in fig [15]. It can be easily observed that the previous plots generated from our own developed diffusion map algorithm matches quite well with these plots. Furthermore, selection of best embedding space using localRegressionSelection method was also done in which ϕ_1 and ϕ_5 was found to be pair of parsimonious eigenvectors. We see that the selection-model was able to find the same eigenvector pairs for embedding that we identified as the best choice before.

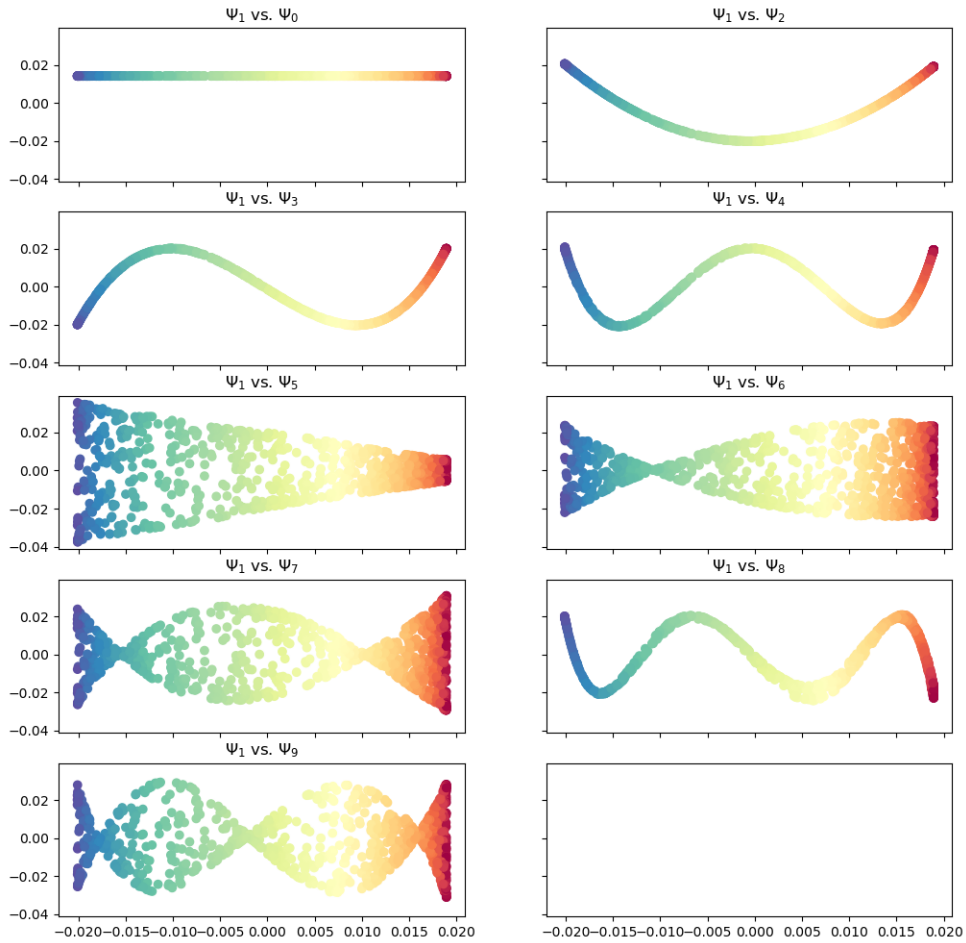


Figure 15: Eigenfunction plots using datafold

2.6 Mandatory Questions

(a) **an estimate on how long it took you to implement and test the method?**
It took roughly around 3 days for us which ranges from grasping of the theoretical contents, programming implementation of subtasks, struggle to get correct answers, testing the method, discussion with tutor and peers to finally putting the stuffs into the report.

(b) **how accurate you could represent the data and what measure of accuracy you used**
We used the independent eigenfunctions from Diffusion Map algorithm to determine the 2D embedding space representation for the Swiss Roll Dataset. The qualitative measure of accuracy was performed by plotting the data in the embedding space, visually inspecting the result and comparing it with the Datafold software.

(c) **what you learned about both the dataset and the method (which is probably different from what the machine learned).**
We learned that the Diffusion Maps is a method for nonlinear dimensionality reduction that tries to capture the intrinsic geometry of the data by computing eigenfunctions of Laplace Beltrami operator on manifold describing the data. It uses the concept of diffusion to define a distance metric between data points. The Swiss roll dataset is a good representation of 2D manifold in 3D space. Also, the number of points in dataset influences the data representation technique.

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

The Variational Autoencoder (VAE) is a type of generative model within the realm of unsupervised learning. It's a neural network-based probabilistic model that learns to represent high-dimensional data in a lower-dimensional latent space.
VAE solves the problem of non-regularized latent space in autoencoders and provides generative capabilities to the entire space. The encoder in AE outputs latent vectors. The encoder of VAE does not output vectors in the latent space, but rather outputs parameters of a predefined distribution in the latent space of each input. The VAE then imposes constraints on this underlying distribution, forcing it to become a normal distribution. This constraint ensures that the latent space is regularized.
This task is difficult for us. This is the first time to implement a variational autoencoder. We tried to implement the model based on tutorials and information found on the Internet. We were very frustrated with dealing with the ELBO loss, and the resulting curve did not look like Good. But fortunately the latent representation, reconstructed digits, generated digits and training curves performed well.

1.Preprocessing dataset

We selected the MNIST dataset, preprocessed it, including normalization, and divided it into a training set and a test set.

2.Model definition

According to the definition in the tutorial, using 2D latent space to build the model, we added the "generate-samples" function and the "reparameterize" function to solve the next problems.

3.Model training

As the question says, we use the Adam optimizer and a learning rate of 0.001 to train the model and the training batch size is 128.

4.Optimize models and evaluate results

After the 1st, 5th, 25th, and 50th iterations, the four potential representations are drawn (i.e., the test set is encoded and different classes are marked), 15 reconstructed digits and the corresponding original digits are drawn, and 15 digits are generated. Finally draw the loss curve.

5.Train the model using a 32-dimensional latent space

We use a 32-dimensional latent space to train the VAE and compare the generated 15 digits with the loss curve.

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

It took us about 10 hours to implement the method and about 1 hour to test it.

(b) How accurate you could represent the data and what measure of accuracy you used.

We use ELBO loss to measure the performance of the model. In the setting of 2- and 32-dimensional latent spaces, we get ELBO losses of 513 and 511. ELBO is the sum of the reconstruction loss and the KL divergence loss. The reconstruction loss measures the difference between the reconstructed numbers and the original numbers, while the KL divergence loss measures the difference between the latent variable distribution 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 have learnt that MNIST is a relatively small dataset of handwritten digits, and that such a dataset is ideal for learning and understanding the VAE model. This implementation of VAE is also a simpler model. They provide a platform to help us understand learning, data generation, compression and other basic concepts in unsupervised learning.

1.What activation functions should be used for the mean and standard deviation of the ap- proximate posterior and the likelihood—and why?

In a VAE, the mean and standard deviation parameters of the approximate posterior distribution (encoder output) are learned by the neural network. Similarly, the likelihood (decoder output) represents the reconstructed output or the probability distribution of the input given the latent variables.
For approximating the mean of the posterior, a linear activation function is generally used or no activation function is used. Because we want the encoder to be able to map the input data to any position in the latent space, maintaining its linearity allows the network to output any real number as the mean without limiting the value range.
To approximate the standard deviation of the posterior, the standard deviation should be positive, use an activation function that outputs positive values, such as the exponential function we used (torch.exp), you can also use the ReLU, Sigmoid, Softplus function, etc.
For the mean of the likelihood, we use the sigmoid activation function, since we are dealing with image data with pixel values ranging from 0 to 1, ensuring that it represents a valid probability. For continuous data (for example, grayscale images or normalized data in the range [0, 255]), you can use hyperbolic tangent (Tanh) or ReLU (with appropriate scaling). Tanh compresses values between -1 and 1, ReLU introduces sparsity by setting negative values to zero, and is generally suitable for appropriately scaled and normalized image data.
For the standard deviation of the likelihood, there is no need to use an explicit activation function.

2.What might be the reason if we obtain good reconstructed but bad generated digits?

Overfitting on the training set. VAE cannot generalize because it relies too much on the specificity of training data. The VAE's latent space might be overly constrained or regularized, making it difficult for the decoder to generate diverse and meaningful samples. A highly regularized or restrictive prior distribution in the latent space could limit the diversity of generated samples.
Another possible reason is that, during training, the model may not be able to fully explore and understand all possible regions in the latent space. The methods used to sample from the latent space may be inefficient or inadequate. If sampling is not performed properly, it can result in the resulting samples having limited diversity and poor quality.

3(a-c) Latent representations of the 1st epoch, 5th epoch, 25th epoch, 50th epoch, 15 reconstructed digits and corresponding original digits and 15 generated digits

The latent representation of the quartic is shown in the figure 16. It can be seen that the data distribution is spreading outward, and the three categories of numbers "0", "1" and "2" are the most obvious.

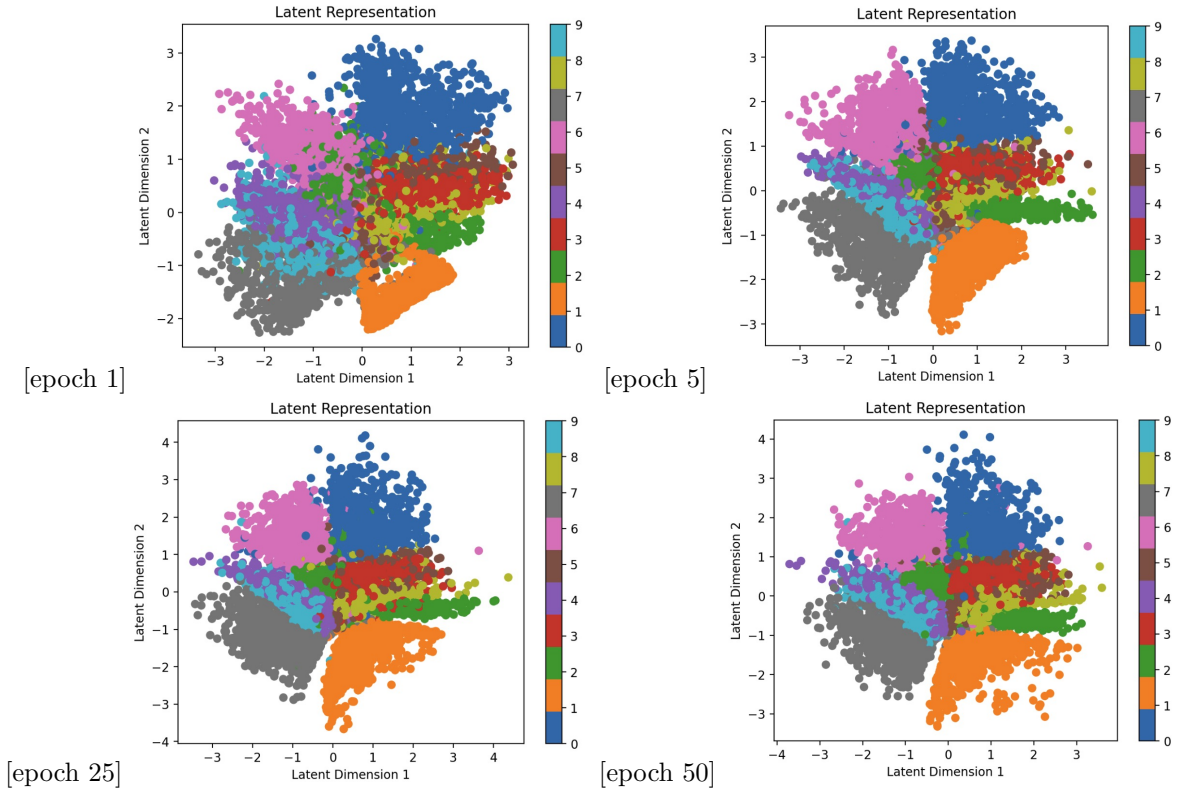
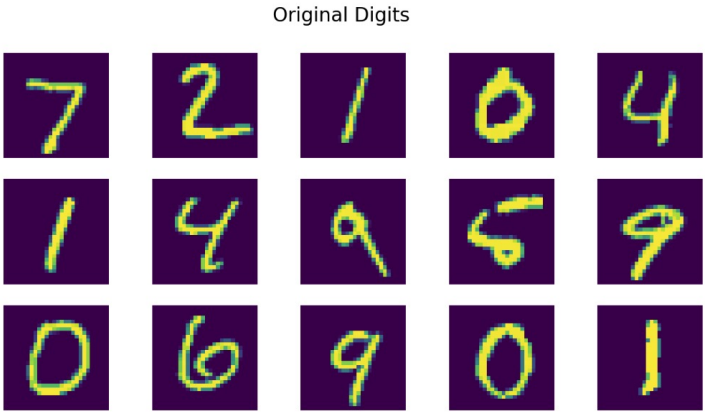
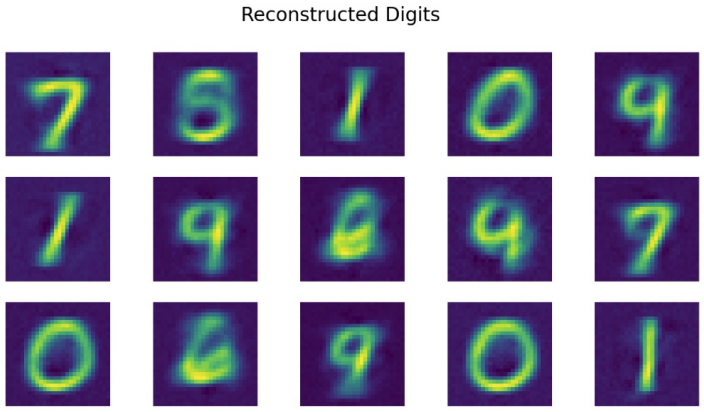


Figure 16: Latent Representation

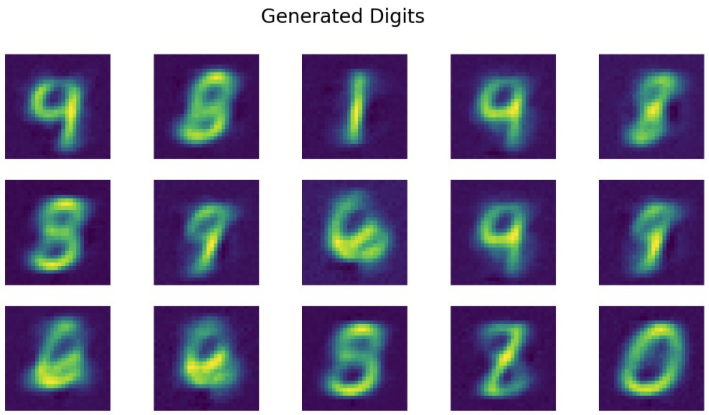
The original digits, reconstructed digits, and generated digits of the 1st epoch are shown in the figure 17. We can see from subfigure a that the original image has some numbers that are difficult to decipher. Most of the reconstructed images in subfigure b are satisfactory, except that the number "2" is reconstructed into "8". However, the reconstruction of the originally illegible image fails and is almost unrecognizable. The resulting image in subfigure c is more difficult to identify.



(a) original digits



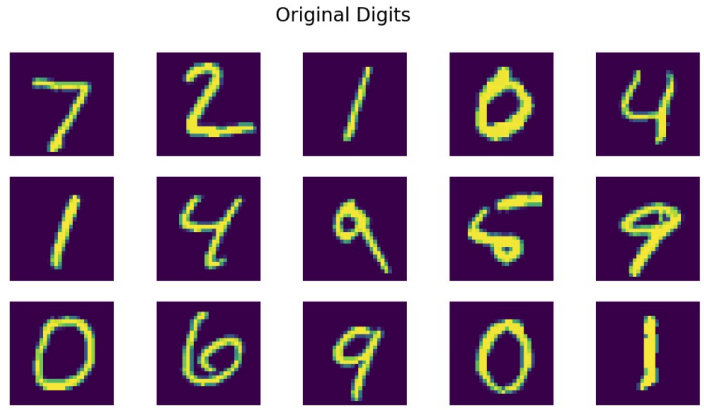
(b) reconstructed digits



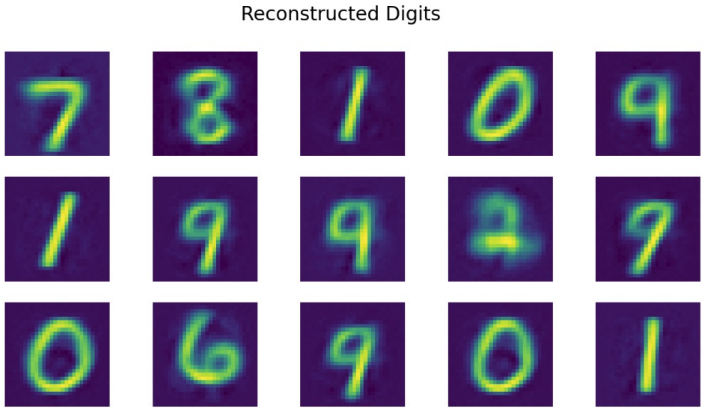
(c) generated digits

Figure 17: 15 Digits

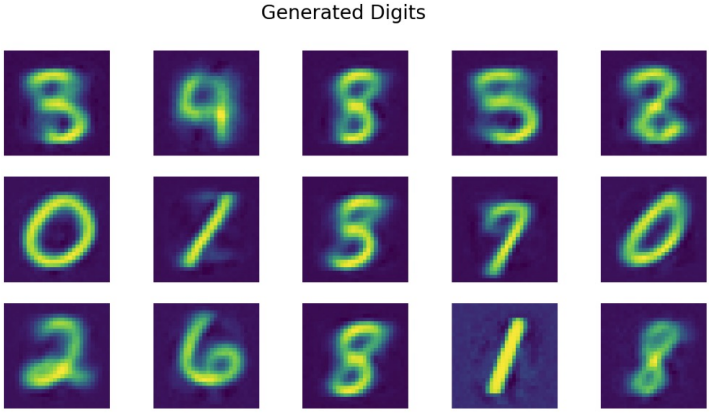
The original digits, reconstructed digits and generated digits of the 5th epoch are shown in the figure 18. As can be seen from subfigure b, we can see that the contrast is clearer for the first time. The "9" in the middle is correctly reconstructed. However, the numbers "2" and "4" are still mistakenly considered to be "8" and "9". The comparison of the generated images in subfigure c is clearer for the first time, but it is still ambiguous for "3" and "8".



(a) original digits



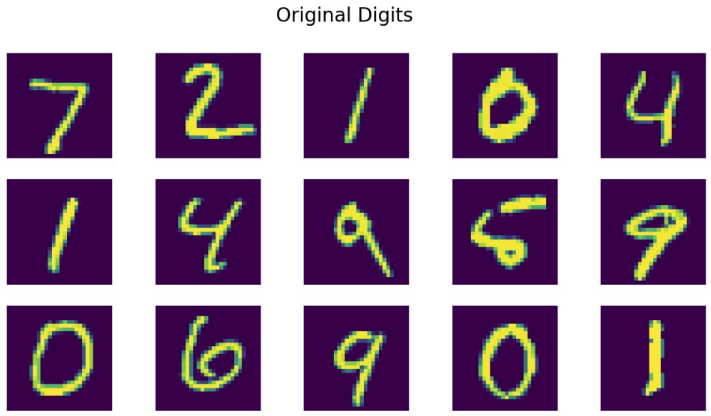
(b) reconstructed digits



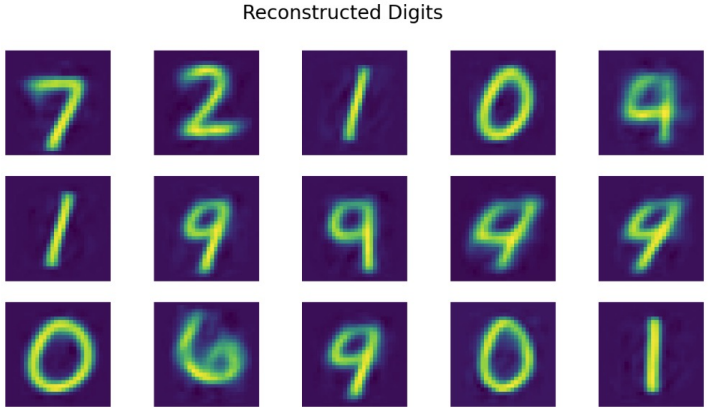
(c) generated digits

Figure 18: 15 Digits

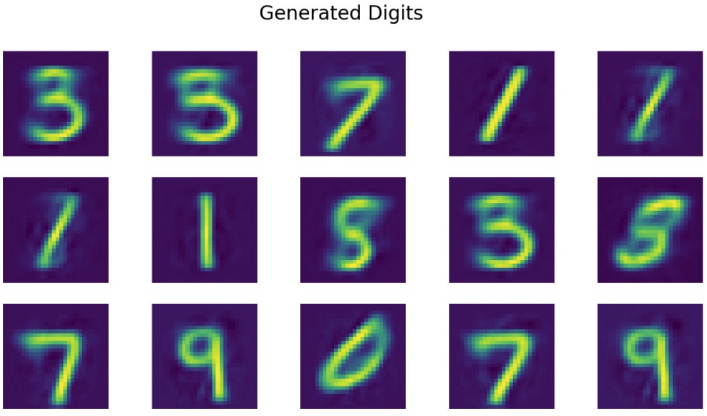
The original digits, reconstructed digits and generated digits of the 25th epoch are shown in the figure 19. In the 25th subfigure b, we can see that the number "2" has been successfully reconstructed, which is a great improvement. However, the reconstruction of "4" still failed. Subfigure c looks difficult to distinguish between the numbers "3", "5" and "8".



(a) original digits



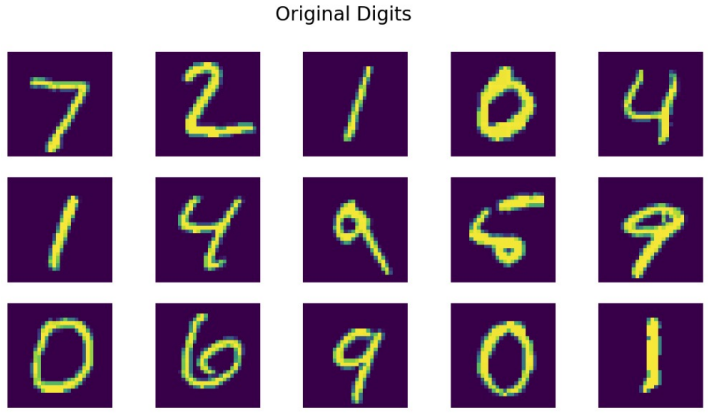
(b) reconstructed digits



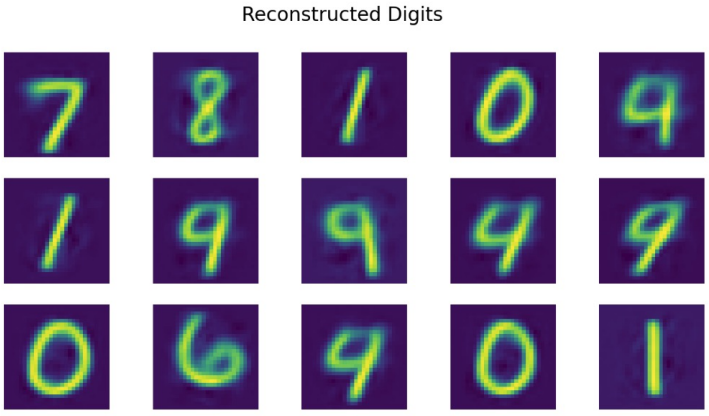
(c) generated digits

Figure 19: 15 Digits

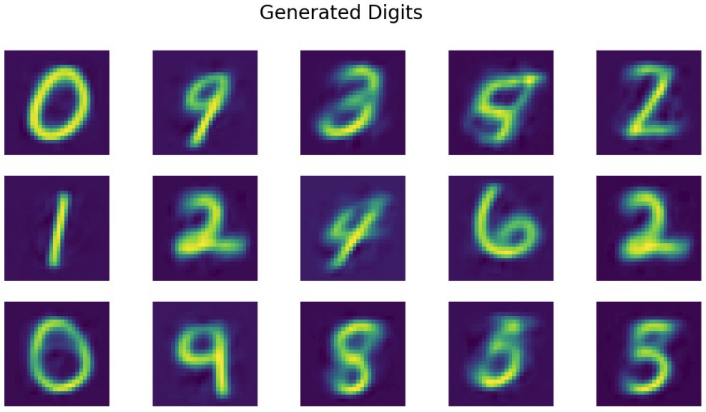
The original digits, reconstructed digits and generated digits of the 50th epoch are shown in the figure 20. The 50th subplot is worse than expected, neither the number "4" is clearly reconstructed, nor the number "2" is successfully reconstructed. Some numbers in subfigure c are still difficult to decipher.



(a) original digits



(b) reconstructed digits



(c) generated digits

Figure 20: 15 Digits

4. Plot the loss curve (test set), i.e., epoch vs. ELBO.

As shown in the figure 21, the loss curve for 50 epochs is shown. It can be seen that the training loss curve gradually converges. However, the test curve shows that there seems to be some overfitting state.

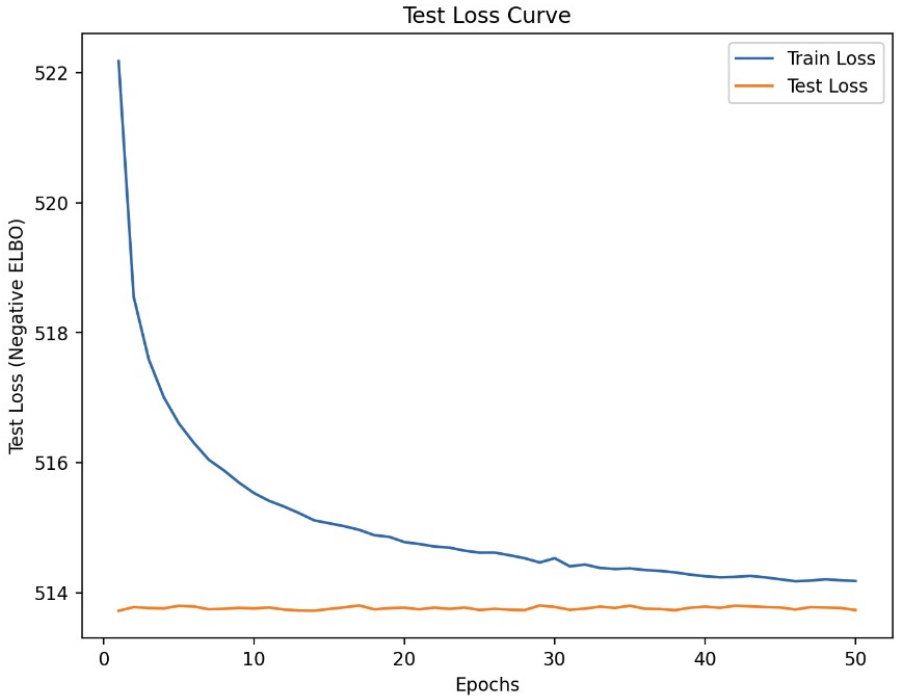


Figure 21: Loss curve in the 2-dimensional latent space

5(a) 32-dimensional latent space: Compare 15 generated digits with the results in 3.2

We can see from the figure 22 that digits are much clearer, and the numbers "3", "5" and "8" can clearly be identified. These numbers are difficult to identify in the 2-dimensional latent space.
A higher-dimensional latent space can provide more capacity to capture the complexity and variation of the data, which will make the resulting image clearer and more legible. A 2-dimensional latent space cannot fully express the complexity of the original data. However a latent space that is too high-dimensional may cause the model to remember the noise in the training data instead of learning the key features of the data.

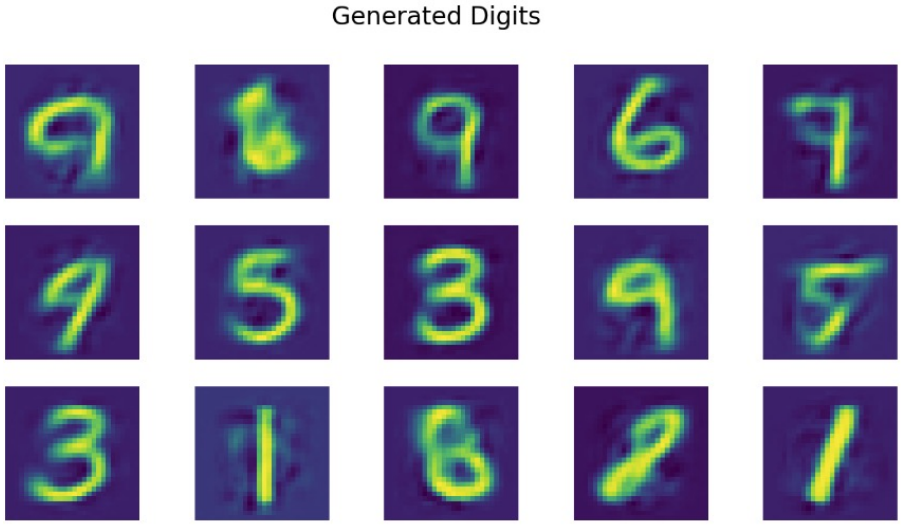


Figure 22: 15 numbers generated in the 32-dimensional latent space

5(b) 32-dimensional latent space: Compare the loss curve with the one in 4

It can be seen from the figure 23 that the training curve is much smoother than that of 2 dimensions, and gradually converges. Although the test curve is still not good, compared with 2 dimensions, the 32-dimensional latent space is better.

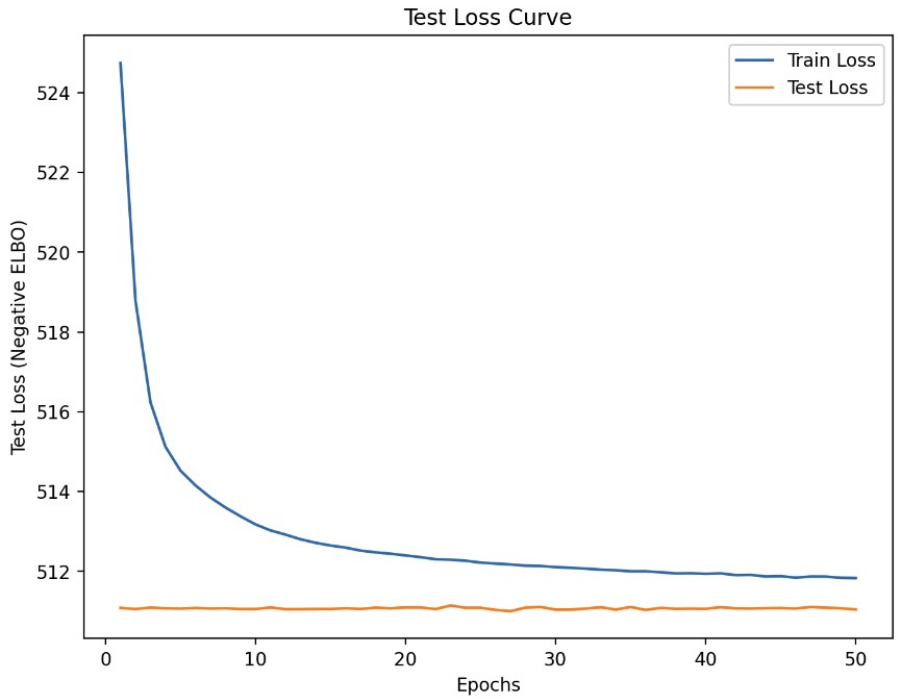


Figure 23: Loss curve in the 32-dimensional latent space

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

1.Scatter Plot of FireEvac Dataset

In this section, we will work on the fire evacuation plan for the MI building dataset. This is what looks like 24:

2.Train a VAE on the FireEvac Data to Learn p(x)

In this section, we changed our code accordingly.

1. We changed the dataset
2. Then, we rescaled the input data x to a range of [-1; 1]
3. We changed our dimensions. In the task 3, our input dimension was 784, now it is 2. Hidden dimension was 256, now it is 64. Latent dimension stayed the same as 2. Also, learning rate from 0.001 to 0.005, batch size 128 to 64.
4. Some changes in the model according to our dataset and needs

3.Scatter Plot of the Reconstructed Test Set

Then, we plotted the reconstructed test set by coding a new function. We used 200 epochs for our model. This is our plot 25:

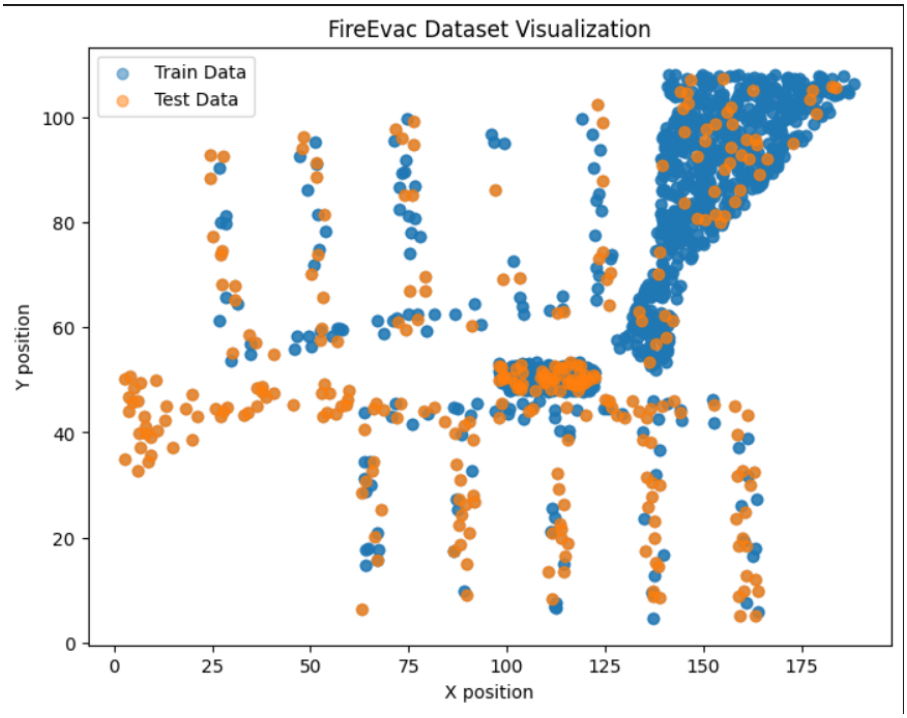


Figure 24: FireEvac Dataset Visualization

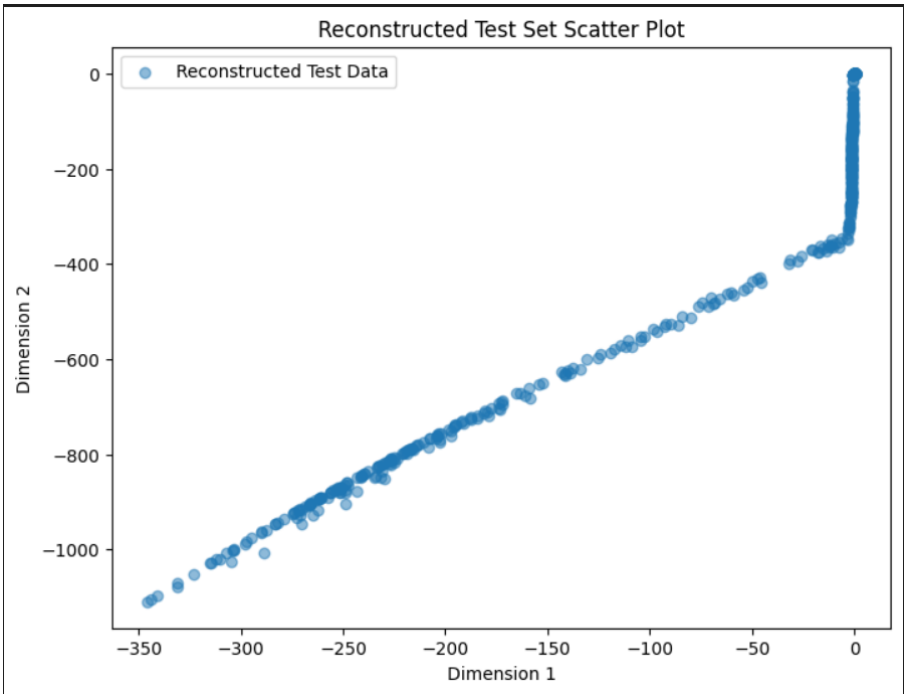


Figure 25: Plot of the Reconstructed Test Set

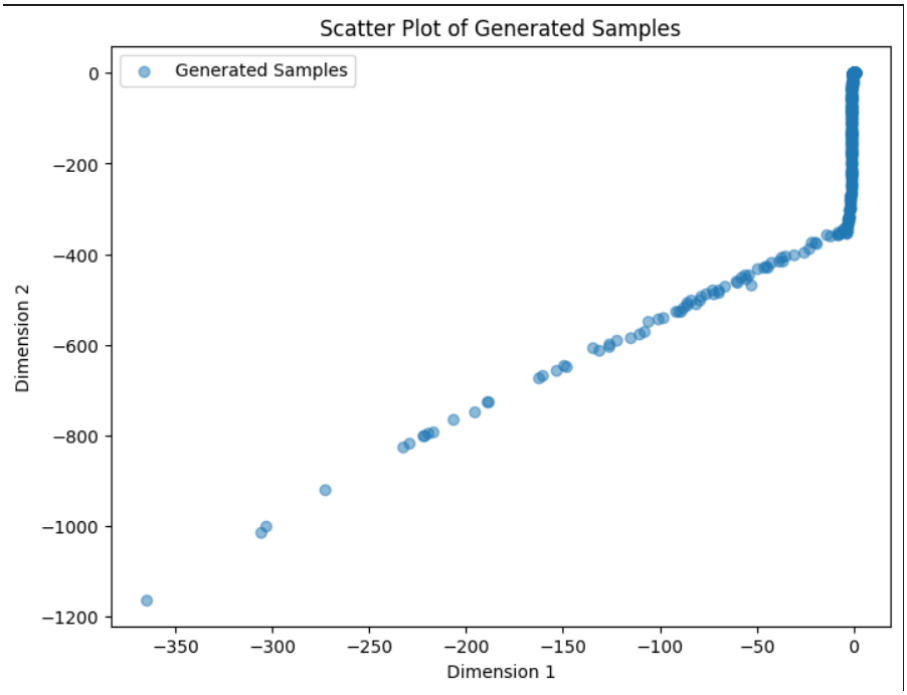


Figure 26: Plot of the 1000 Generated Samples

4.Scatter Plot of 1000 Generated Samples

Then, we plotted the new 1000 generated samples set by coding a new function. We used 200 epochs for our model. This is our plot 27:

5.Generate Data to Estimate the Critical Number of People for the MI Building

We tried to find the critical people. It should be between 500 and 1000 but we couldn't find in the code. We tried to find the people between (130/70 150/50) when we sample from our model. However, even 1000000 wasn't enough because our model was putting every sample to the end.

bonus

We did the Scenes.

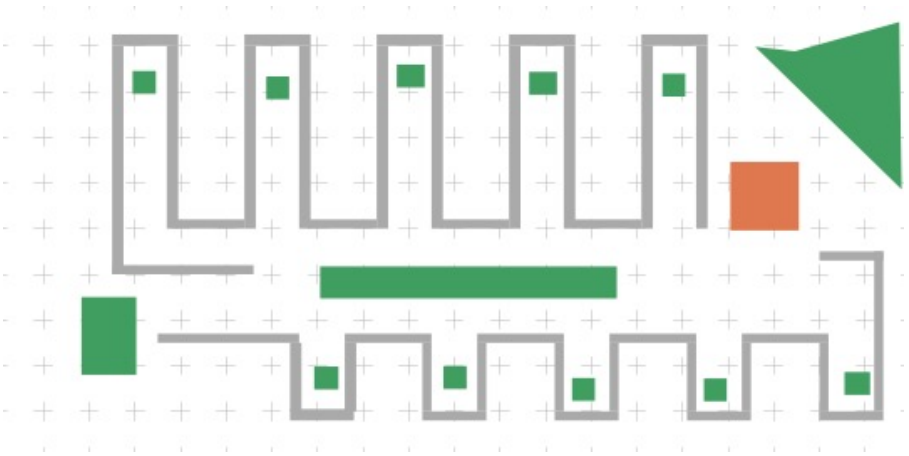


Figure 27: Plot of the 1000 Generated Samples