Report for exercise 4 from group H

Tasks addressed:	4
Authors:	Taiba Basit (03734212) Zeenat Farheen (03734213) Fabian Nhan (03687620)
Last compiled:	2021–06–09
Source code:	https://github.com/Combo1/MLCMS/tree/main/exercise4

The work on tasks was divided in the following way:

Taiba Basit (03734212)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
Zeenat Farheen (03734213)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
Fabian Nhan (03687620)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%

Report on task 1/4, Principal component analysis

For the first part we downloaded the dataset pca_dataset.txt from Moodle. Then we read the text data using np.loadtxt. Then we plotted the data using plt.scatter where we provided the first column as the first dimension and second column as the second diemnsion. Figure 1 shows the plot of the original data.

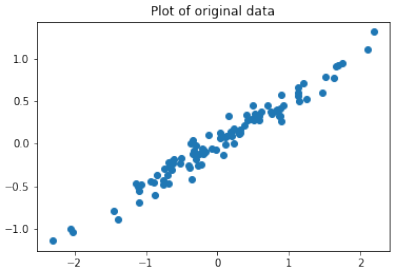


Figure 1: Plot of original data

Implementing Principal Component Analysis

We implemented the PCA algorithm in a separate python file and imported it whenever required. For the pca we took two arguments, the original data and the number of principal components the reduced data should have. The algorithm is discussed below:

Step 1: We calculated the mean along each column. We obtained a mean of 0.06 along first column and mean of 0.0455 along second column.

Step 2: To center the data we subtracted the data row wise from the mean. Figure 2 shows the centered data along with the original data. As it can be seen from the figure 2, the centered data is shifted from the original data and now has a mean of zero. The small shift represents the small value of mean as obtained in previous step.

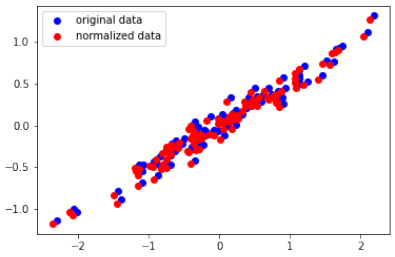


Figure 2: Plot of original data along with normalized data

Step 3: We then decomposed the centered data into singular vectors U, V and values S by using np.linalg.svd. We also set the full_matrices argument to false because the U and V matrices are not square matrices.The functions returns the following

- u of shape (100,2) with columns representing the eigenvectors.
- s , one dimensional array of singular values sorted in ascending order representing the eigenvalues
- vh of shape (2,100) representing the principal components

We then created a diagonal matrix s_diag by using np.diag(s).

Step 4: We then reduce the size of u , s and vh to the number of principal components as required by the user. We project the normalized data onto the reduced dimension coordinate system by multiplying the centred data with the resulting principal components.

Step 5 : To reconstruct the original data , we add the calculated mean to the projected data.

Step 6 : We also calculate the variance along the principal components by using np.var(projected_data, axis=0)

Figure 3 shows the two obtained principal components. The length of the principal component represents the variance along it. Thus it can be inferred from the figure that PC1 has a higher variance than PC2.

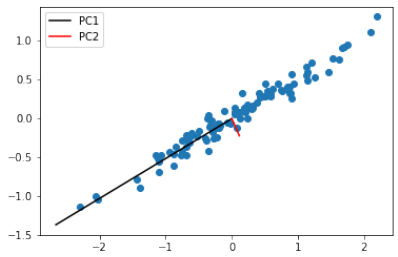


Figure 3: Plot of original data along with Principal Component

Figure 4 represents the centered data projected in the new coordinate system along the principal components, Here, PC1 represents the x-axis and PC2 represents the y-axis.

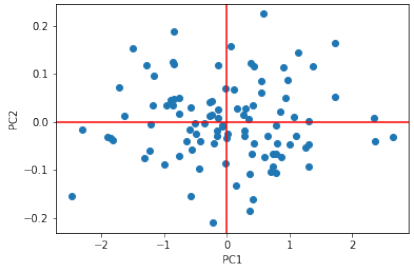


Figure 4: Plot of original data along the Principal Component

The energy of the i-th principal component is contained in the singular value on the diagonal of the matrix S. The energy along PC1 is obtained as 9.9434 and along PC2 is 0.8262. We see that PC1 has higher energy than PC2.

For the second part, we generated the image using scipy.misc.face() and set gray=True, load the grayscale image. The size of the original image is (768,1024). Figure 5 shows the grayscale image.

Figure 5: Grayscale image of racoon

After generating the image we rescaled it to size (249 × 185). For rescaling we used the resize function from scipy.transform and passed the original data and the desired size to it. Figure 6 shows the rescaled image.

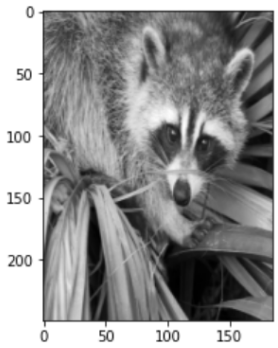


Figure 6: Rescaled(249 × 185) grayscale image of racoon

Then we called the pca function, which we implemented in the last part. For the first case we set the value of number of components to 185. Figure 7 is the reconstructed image. As we have kept all the principal components , the energy of the system is unchanged and we have 0% energy loss.

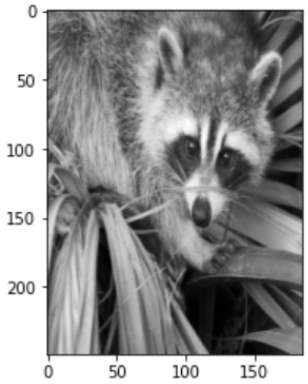


Figure 7: Grayscale image of raccoon with L=185

For the second case we set the value of number of components to 120. Figure 8 is the reconstructed image. After retaining only 120 principal components, we observe an energy loss of 0.2%. Here the energy lost is smaller than 1%. We get an energy loss $< 1\%$ up till retaining 77 principal components.

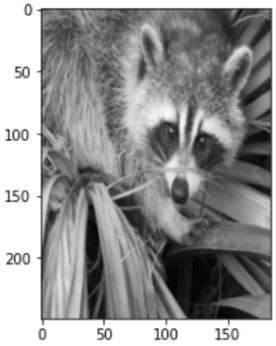


Figure 8: Grayscale image of raccoon with L=120

For the third case we set the value of number of components to 50. Figure 9 is the reconstructed image. After retaining only 50 principal components, we observe an energy loss of 2.48%. We observe that the reconstructed image is not as clear as the previous case. We see that the clarity of image is decreasing and the picture has started to look blurry.

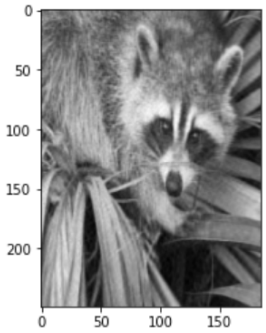


Figure 9: Grayscale image of raccoon with L=50

For the fourth case we set the value of number of components to 10. Figure 10 is the reconstructed image. After retaining only 10 principal components, we observe an energy loss of 17.61%. The image is not clearly visible and the quality has decreased drastically as compared to the previous case. This suggests that around 99% of the energy is contained in approximately the top 70 principal components, anything less than that will result in information loss of the reconstructed data.

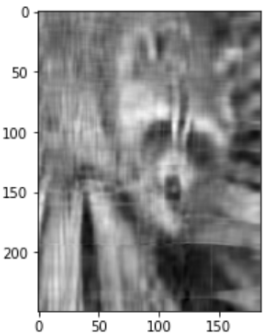


Figure 10: Grayscale image of racoon with L=10

For the third part we downloaded the data_DMAP_PCA_vadere.txt. The file contains the data of 15 pedestrians over 1000 timesteps. For plotting the trajectory of the first two pedestrian, we took data of first four columns. The first two columns contains the data of first pedestrian, where column 1 is the x-coordinate and column 2 is y-coordinate. Similarly, 3rd and 4th column represents the x and y coordinate of the 2nd pedestrian respectively. Figure 11 shows the trajectory.

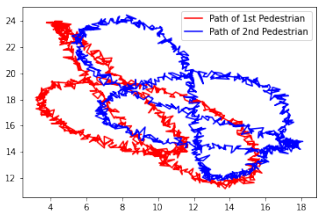


Figure 11: The path of first 2 pedestrians

After performing pca on this data and setting the number of principal components to 2 we observe the reconstruction of the trajectory of the first 2 pedestrians in Figure 12. The two components are not enough to capture the most energy of the system. With 2 components we get the energy around 84%. We observe that the trajectory of 2nd pedestrian loses detail. This can be explained because we have only retained the top 2 principal components which have not captured more than 90% of enegry in the data.

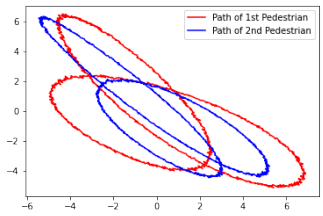


Figure 12: The path of first 2 pedestrians with 2 Principal Component

Next, we set the number of principal components to 3. Figure 13 shows the trajectory of pedestrians with 3 principal components.The total energy captured by 3 principal components is around 99%. Thus, the system with only 3 principal component is enough to capture the most of the energy of the original system. We see that the details in path of pedestrian 2 is very similar to the original data.

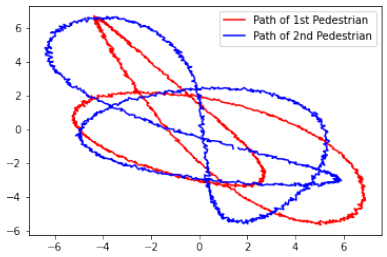


Figure 13: The path of first 2 pedestrians with 3 Principal Component

Report

(a) It took us approximately 2 day to implement and test the method.

(b) We could represent the data accurately using matplotlib and sklearn library. To measure the accuracy of the result obtained, we took the energy retained in the system after PCA as a metric. It gave us a good estimate of how much information was lost by truncating different number of principal components of the given data.

(c) We learned that it is difficult to assume that a given data could be easily represented in a lower dimension by linear dimensioanlity reduction. In the PCA analysis of data_DMAP_PCA_vadere.txt data, by simply looking at the trajectory one could not assume that a dimensionality reduction into 2 dimension would result in information loss. This is evident when performing PCA and comparing the results with 2 and 3 principal components.

Report on task 2/4, Diffusion Maps

For the first part we created the periodic dataset with N=1000 using the below equation:

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

The resulting data generated is shown in figure 14. We can see the components of sine and cosine wave in the resulting data.

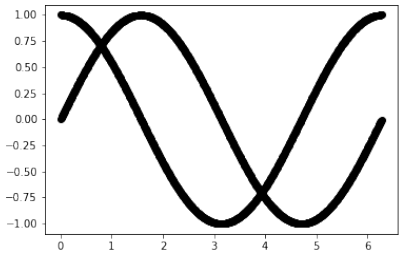


Figure 14: Plotting X against time t

We apply the diffusion map algorithm as descibed in exercise sheet and plot the top 5 eigenfunctions as shown in figure 15 to 19

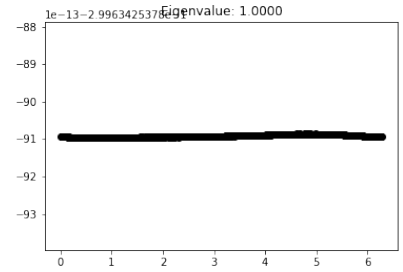


Figure 15: Plot of eigenfunction ϕ_0 against time

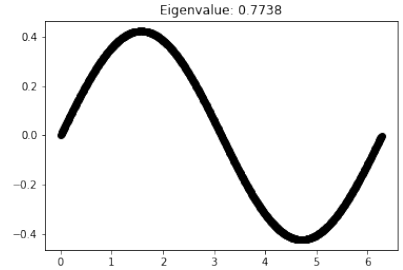


Figure 16: Plot of eigenfunction ϕ_1 against time

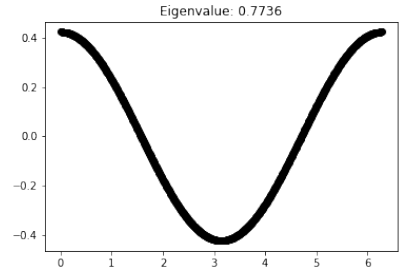


Figure 17: Plot of eigenfunction ϕ_2 against time

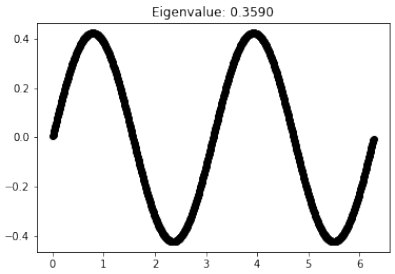


Figure 18: Plot of eigenfunction ϕ_3 against time

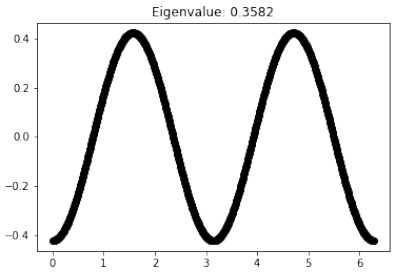


Figure 19: Plot of eigenfunction ϕ_4 against time

We notice that the eigenfunction ϕ_1 corresponds to the sine component function and ϕ_0 corresponds to the cosine component function of the input data X. From this we can infer that the diffusion map algorithm was able to extract the 1 dimensional manifold from the 2-dimensional embedding.

Bonus With the Fourier series, if we have a function that is periodic, then it can be written as a discrete sum of trigonometric or exponential functions with specific frequencies. The signal X is generated by two periodic functions. Carrying out Fourier analysis on X will result in the same two periodic function of sine and cosine. As diffusion map is a non-linear method of dimensionality reduction, the resulting eigenfunctions obtained from the diffusion map is a generalization of the Fourier basis as can be seen in from the result.

In the next part, we generate the swiss-roll manifold in a three-dimensional space by sampling 5000 points with zero noise as shown in Figure 20.

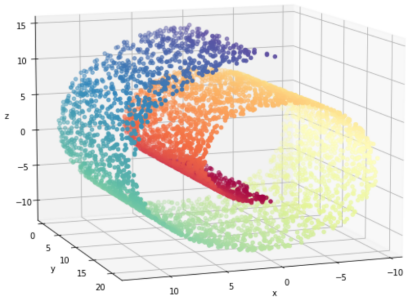


Figure 20: Swiss roll dataset with 5000 points

After this, we applied the diffusion map algorithm and calculated the top 10 eigenvalues and it's corresponding eigenfunctions. We then plot the first non-constant eigenfunction ϕ_1 against the other eigenfunctions in 2d plots with ϕ_1 at x axis. The results obtained are shown in figure 21

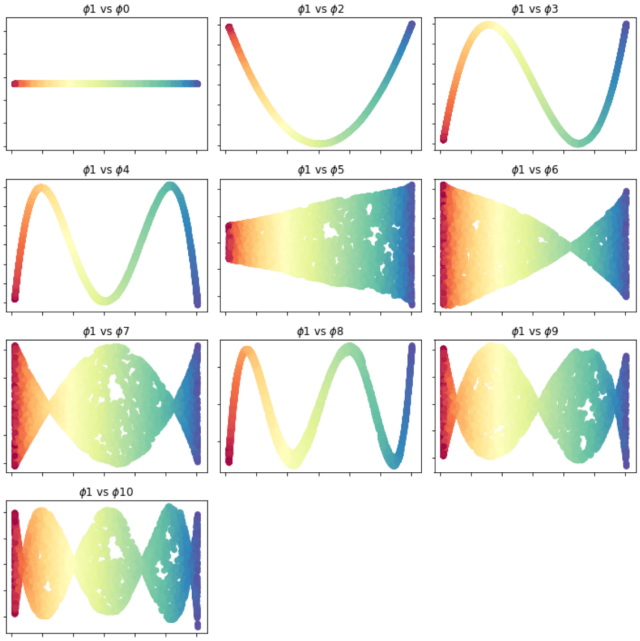


Figure 21: Plots of eigenfunction ϕ_1 against other eigenfuncions for 5000 data point swiss roll

We can see from the results, that for the values of l at 5,6,7,9 and 10, ϕ_l is no longer a function of ϕ_1. For pairings with a functional dependence, the embeddings are poor because the eigenfunctions does not go along a new and independent direction compared to eigenvector ϕ_1. From the obtained results, plot between ϕ_1 and ϕ_5 seems to achieve the purpose of unrolling the swiss roll into a 2 dimensional embedding space.

We use our algorithm to generate the 3 principal components of the swiss roll dataset. The three principal components obtained are

$$\begin{bmatrix} 0.47282996 & -0.05057136 & 0.87970129 \\ -0.87799162 & -0.11154175 & 0.46549883 \\ 0.07458252 & -0.99247215 & -0.09714151 \end{bmatrix}$$

After truncating the number of principal components to 2, we record an energy loss of 28.39%. This loss is significant and it tells us that more than 90% of energy is captured by three principal components. Plotting the reconstructed data on the 2 principal components as shown in Figure 22 , we see that PCA has failed to extract the underlying manifold of swiss roll. There is no way of interpreting that the blue points are the farthest away from the red points by looking at the reconstructed result after PCA. PCA does not differentiate on the basis of the geodesic distance which we are interested in.

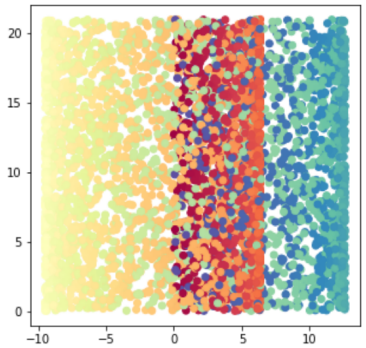


Figure 22: Reconstructed data with 2 principal components

Next, we reduce the number of sample points in swiss roll dataset to 1000. After applying the diffusion map algorithm, we obtain the the plots of first non-constant eigenfunction ϕ_1 plotted against other eigenfunctions as shown in figure 23

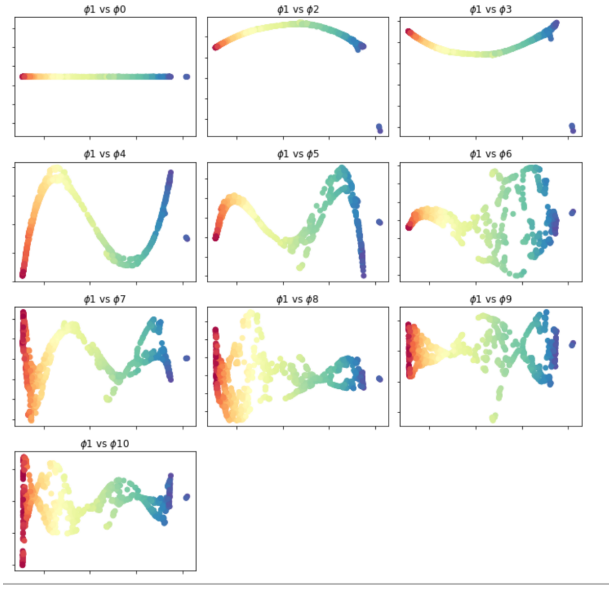


Figure 23: Plots of eigenfunction ϕ_1 against other eigenfuncions for 1000 data point swiss roll

Due to the reduced sample size, the plots obtained are not as good as those obtained with 5000 sample points. This is clear especially in the later eigenfunctions.

Performing PCA on this reduced dataset and keeping all 3 principal components yields similar result as previous case.The three principal components obtained are

$$\begin{bmatrix} 0.45375133 & 0.24638886 & 0.8563890 \\ 0.84882454 & 0.17308064 & -0.49953977 \\ -0.2713054 & 0.95359091 & -0.13060534 \end{bmatrix}$$

When we reconstruct the data on the first 2 principal components as shown in figure 24, we observe that there is a high overlap between the red and blue points that are far away in the original manifold. Thus, the reconstructed datatset in the reduced dimension fails to identify the underlying manifold based on the geodesic distance. Again, the energy loss by keeping only 2 principal componenets is approximately the same and cannot be used to represent the data.

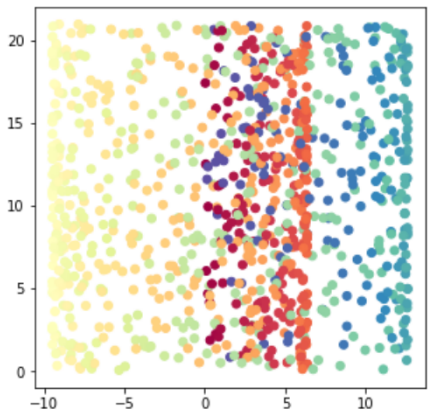


Figure 24: Reconstructed data with 2 principal components

For the third part, we loaded the data from file data_DMAP_PCA_vadere.txt into a numpy array. We plot the trajectory of the first two pedestrians in 3 dimensional space as shown in figure 25.

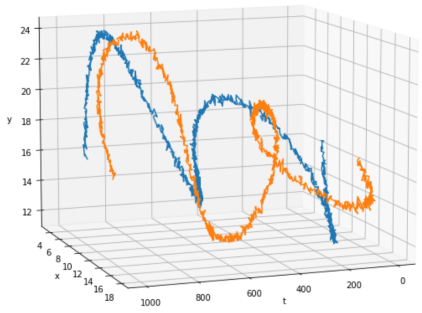


Figure 25: Trajectory of first 2 pedestrians over 1000 timesteps

After this, we apply the diffusion map algorithm and calculated 2 eigenfunctions. Plotting the 2 eigenfunctions as shown in figure 26, we observe that there is an intersection of the curves.

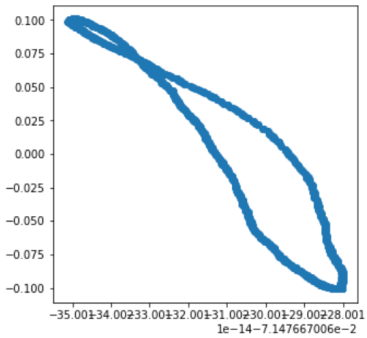


Figure 26: Plot of ϕ_0 and ϕ_1 with 2 eigenfunctions

Next, we increase the number of eigenfunctions to 3 and plot ϕ_0 against ϕ_1 in fig and ϕ_1 and ϕ_2 in figure 27 and 28. It can be seen that, now the curves do not intersect each other. The figure 29 shows the 3 dimensional plotting of all 3 eigenfunctions which do not intersect at any point.

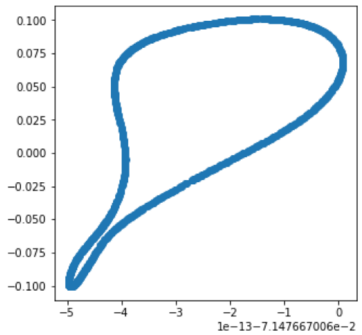


Figure 27: Plot of ϕ_0 and ϕ_1 with 3 eigenfunctions

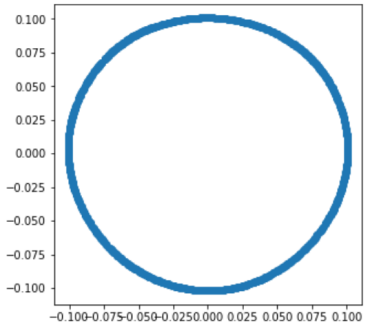


Figure 28: Plot of ϕ_1 and ϕ_2 with 3 eigenfunctions

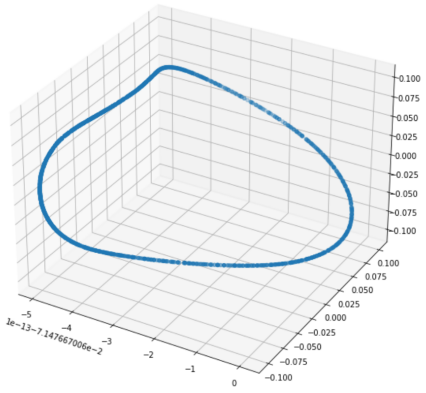


Figure 29: Plot of ϕ_0, ϕ_1 and ϕ_2 in 3-dimensional space

Bonus task For this part, we used the Datafold libraray to apply the diffusion map algorithm on the swiss dataset. We used the same dataset of with 5000 sample points. We first used the PCManifold function to estimate parameters of epsilon and cut-off.

Next we fit the DiffusionMaps model to the data with the optimized parameters and obtain the results as shown in Figure 30

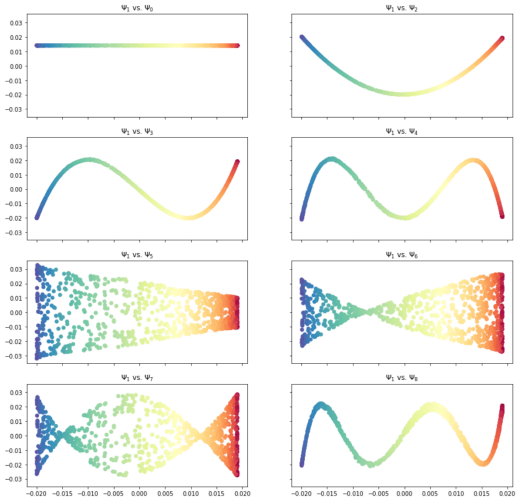


Figure 30: Plots of eigenfunction ϕ_1 against other eigenfuncions

Comparing our results with the results obtained from Datafold library functions, we can see that the func-

tional dependence between ϕ_1 and ϕ_2, ϕ_3 and ϕ_4 is similar. Also, the 2 dimensional embedding obtained in plot ϕ_1 and ϕ_5 is similar to the one obtained in our algorithm.

Report

(a) It took us 6 days to implement and test the method. We had to make decision between choosing different eigensolvers and comparing the result. Testing took the major time as we were trying to understand why the results obtained from our algorithm looked laterally inverted to the results obtained from standard library functions like Datafold.

(b) We could represent the result quite accurately using standard library functions from matplotlib. As diffusion maps do not have exactly the same energy interpretation of the eigenvalues as PCA, we mostly relied on comparing the two-dimensional embeddings of the eigenfunctions obtained.

(c) From the swiss dataset, while the linear strip of data points folded into a swiss roll could be easily recognized by just looking at it. It was interesting to see that the 2-dimensional embeddings of the eigenfunctions obtained from diffusion map could detect it as well. Moreover, we realized that after performing PCA we learned that if a data has linear manifold and is embedded in a higher dimension, PCA performs better in this case than when it has a non-linear manifold.This was evident from PCA's inability to detect the underlying manifold based on geodesic distance which diffusion map detected successfully. From the data_DMAP_PCA_vadere.txt, we learned that diffusion map could identify the underlying 2-dimensional circle embedding which is not visible to the naked eyed.

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

The notebook for this task can be found in My_Task3.ipynb. We used Tensorflow 1.15 to implement the Variational Autoencoder. The notebooks content is based on the sources given in the notebook.

1. z is our latent space and x our input images.

   likelihood: p(x—z): probability of our dataset given the latent space, prior: p(z): distribution of our latent variable which is assumed to be normal with mean 0 and std of 1, posterior: p(z—x): probability of our latent variable given the input.

   According to the tutorial to variational autoencoders, a softplus (modified sigmoid) function was used as activation function for the posterior, the important thing to keep in mind is that we should map towards [0,1], because we normalised our images between 0 and 1, thus our mean should be in this range, too. For the likelihood we can use a sigmoid as well for the same reason.

2. The loss function is composed of two terms: reconstruction loss and KL Divergence. To have good reconstruction as well as good generation, we need a good balance between REC and KL, as we have seen in the lecture, the better the reconstruction the worse our distribution approximates the prior. On the other hand the better the KL Divergence then reconstruction gets harder, because the distributions are overlapping. Good reconstruction and poor generation means that the KL divergence overpowers reconstruction loss.

3. The validation loss as well as the training loss reduces, but what was surprising is that there were no signs of overfitting. In the latent representation early on, the digits do not show much resemblance to specific numbers and are very vague, which gradually became more pronounced. During the training process we could see that the noise starting at epoch 1 gradually vanishes towards the later epochs of the reconstructed images. Even the generated images show noise and no concrete digits until later parts of training.

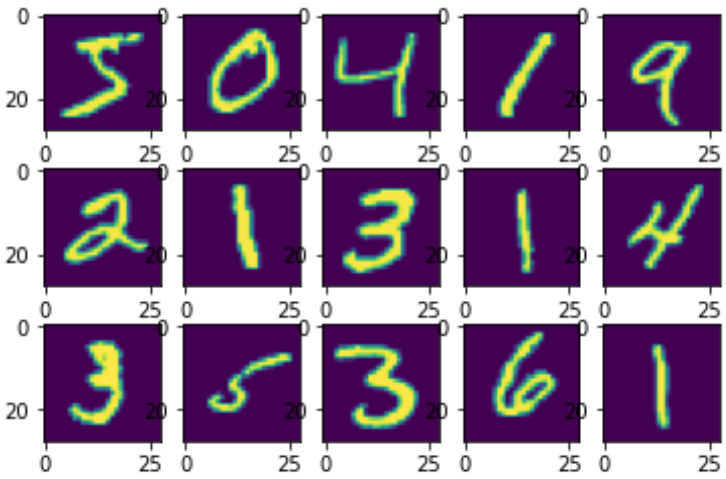


Figure 31: Original images, which are reconstructed later

Training of the Variational Autoencoder after first epoch: (a)

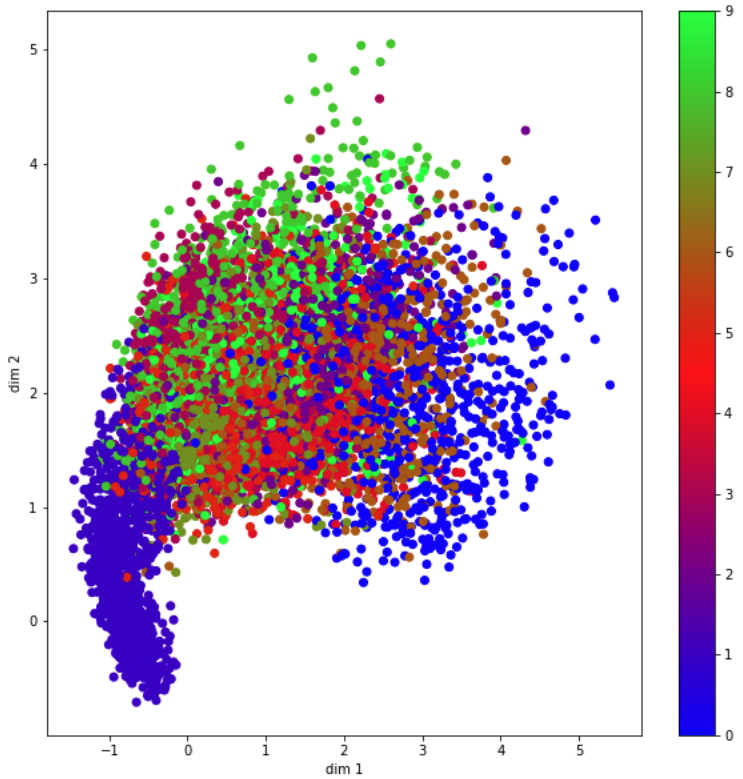


Figure 32: Latent space of vae with one epoch

(b)

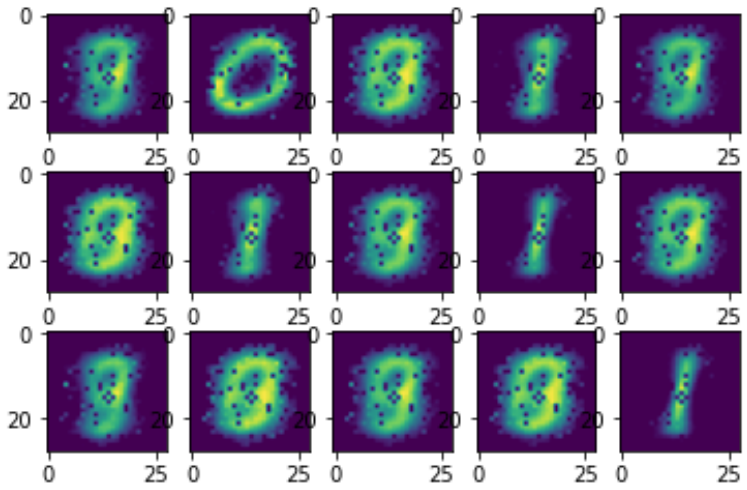


Figure 33: Reconstructed images of vae with one epoch

(c)

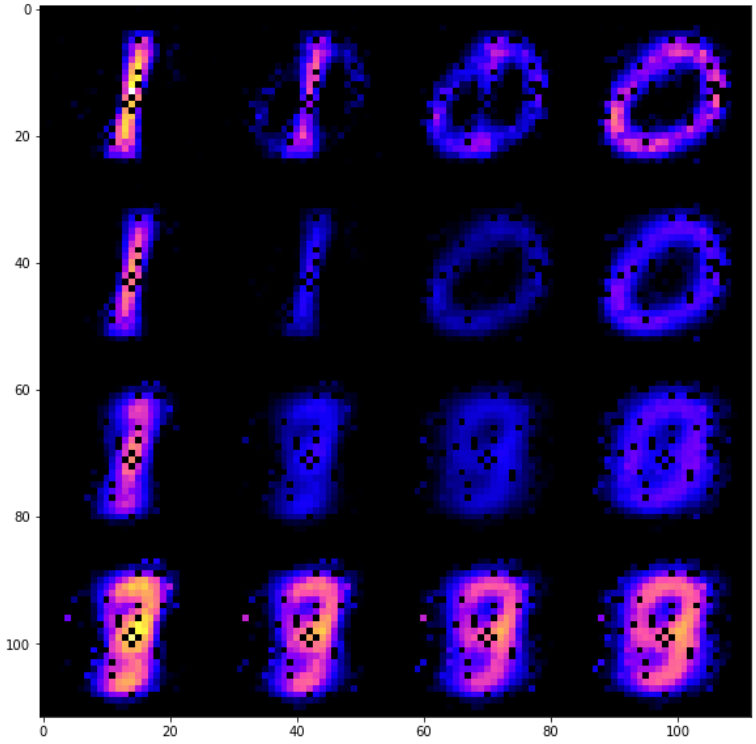


Figure 34: Generated digits from prior

Training of the Variational Autoencoder after fifth epoch: (a)

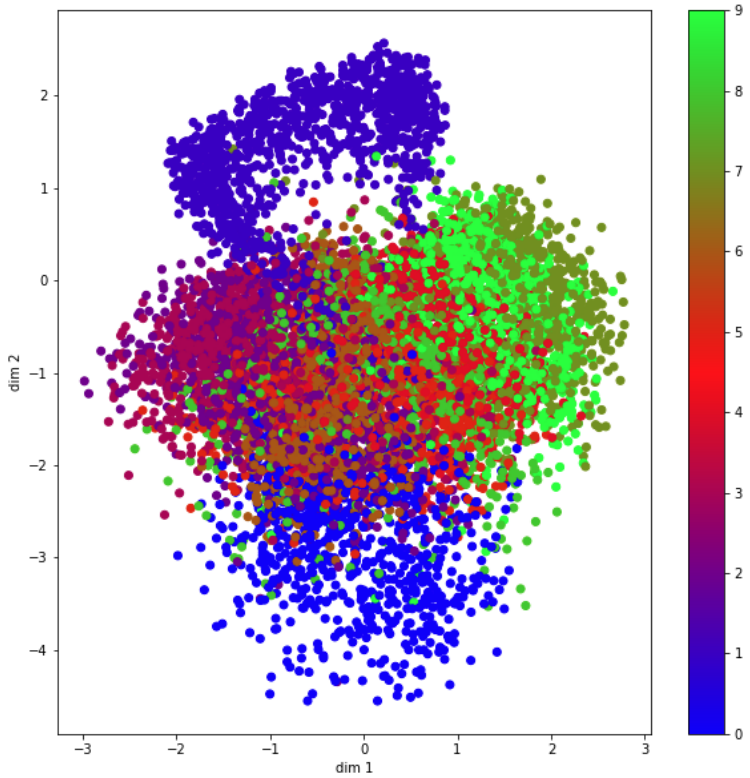


Figure 35: Latent space of vae with five epochs

(b)

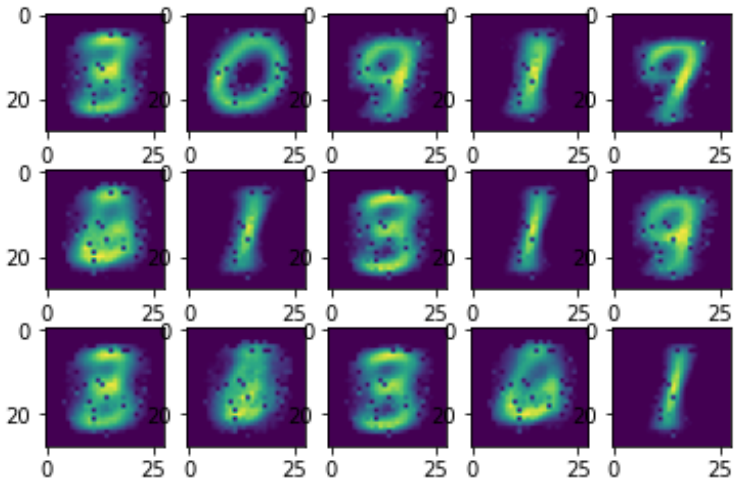


Figure 36: Reconstructed images of vae with five epochs

(c)

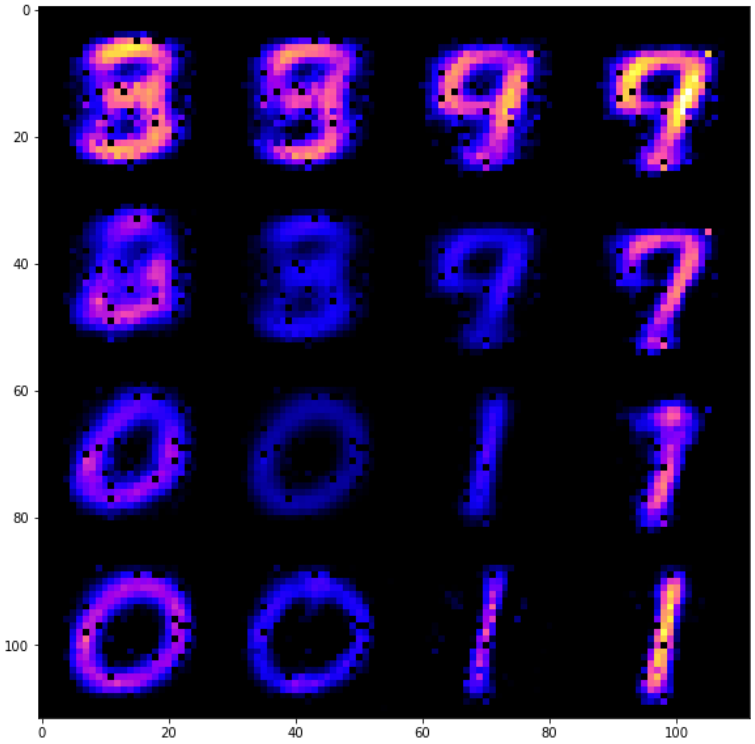


Figure 37: Generated digits from prior

Training of the Variational Autoencoder after 25th epoch: (a)

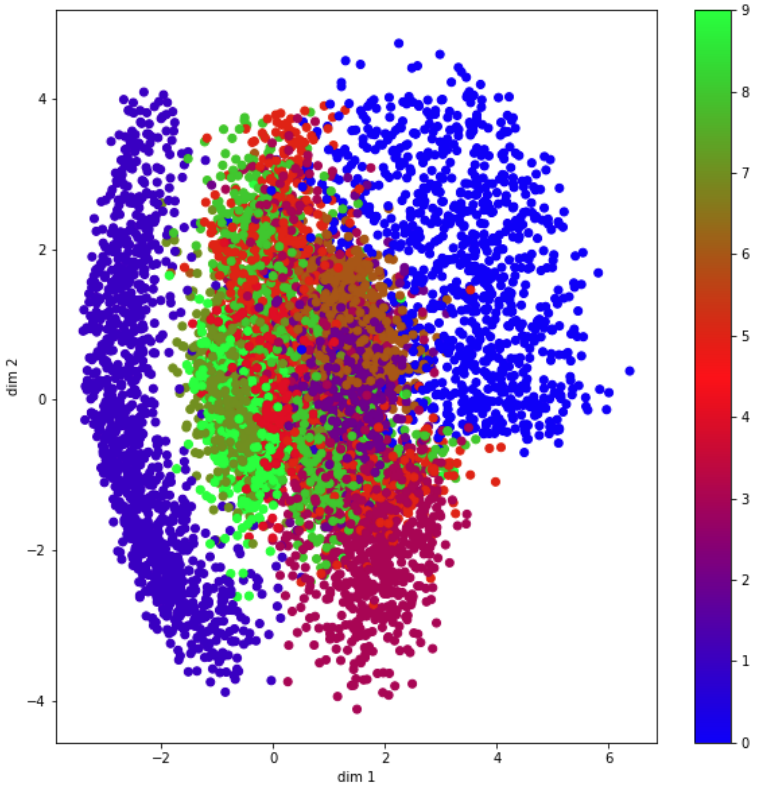


Figure 38: Latent space of vae with 25 epochs

(b)

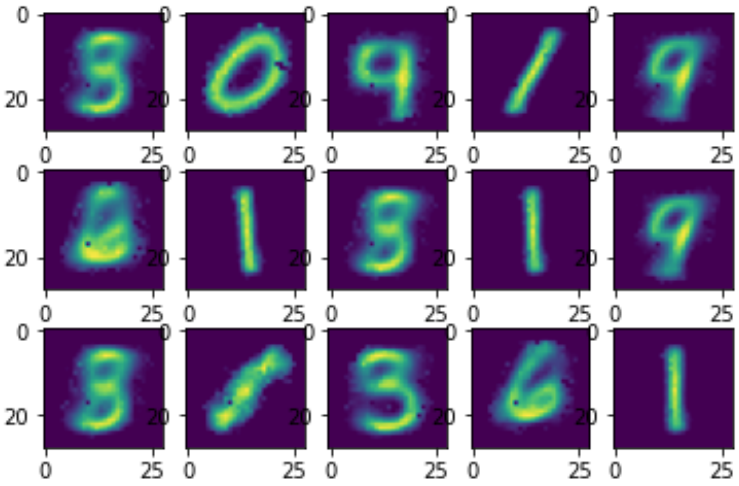


Figure 39: Reconstructed images of vae with 25 epochs

(c)

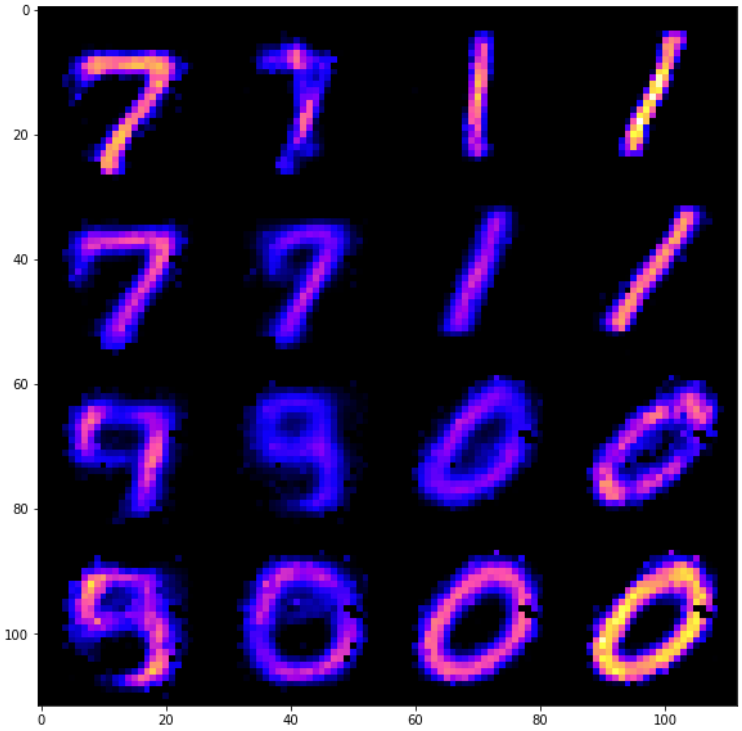


Figure 40: Generated digits from prior

Training of the Variational Autoencoder after 50th epoch: (a)

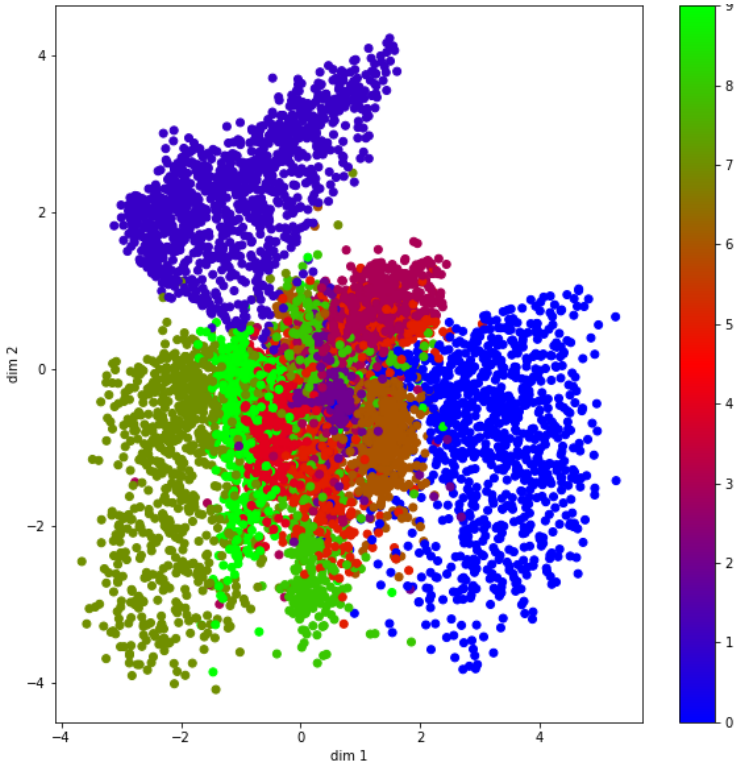


Figure 41: Latent space of vae with 50 epochs

(b)

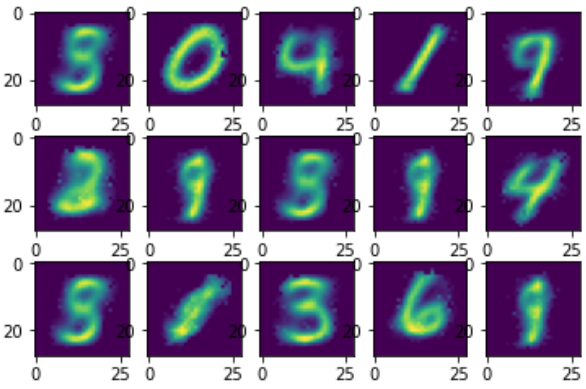


Figure 42: Reconstructed images of vae with 50 epochs

(c)

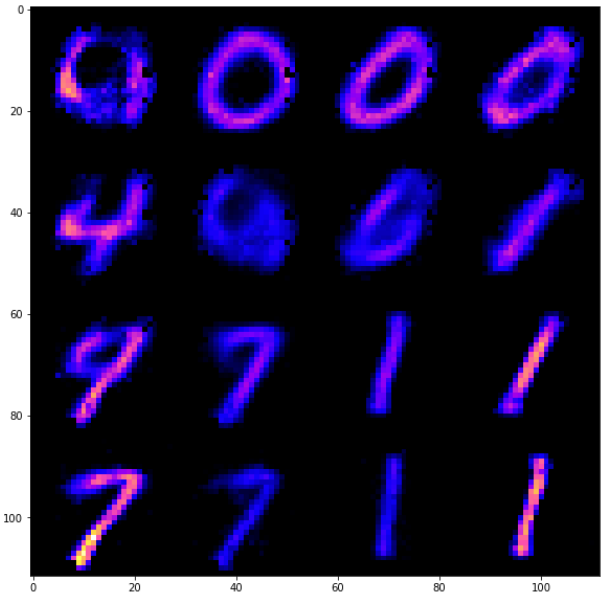


Figure 43: Generated digits from prior

Training of the Variational Autoencoder after convergence:
(a)

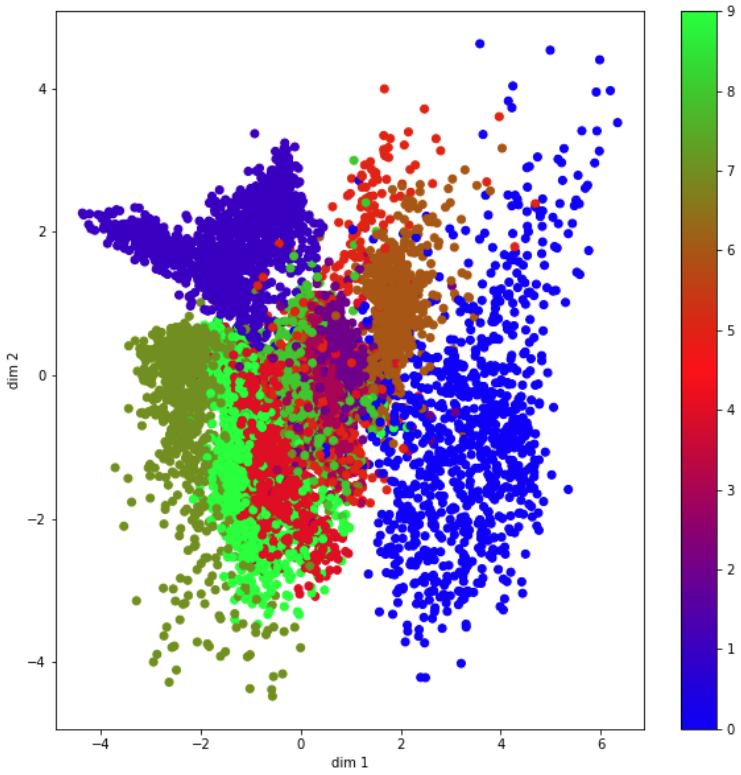


Figure 44: Latent space of vae with 100 epochs

(b)

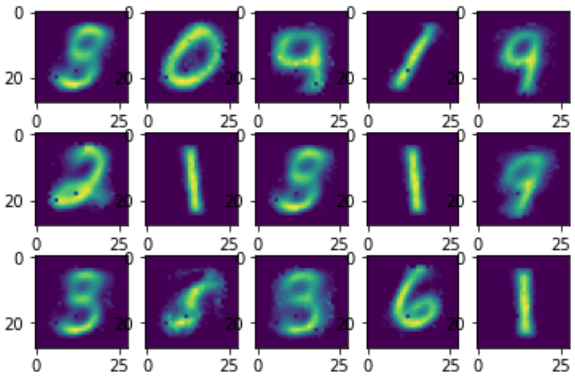


Figure 45: Reconstructed images of vae with 100 epochs

(c)

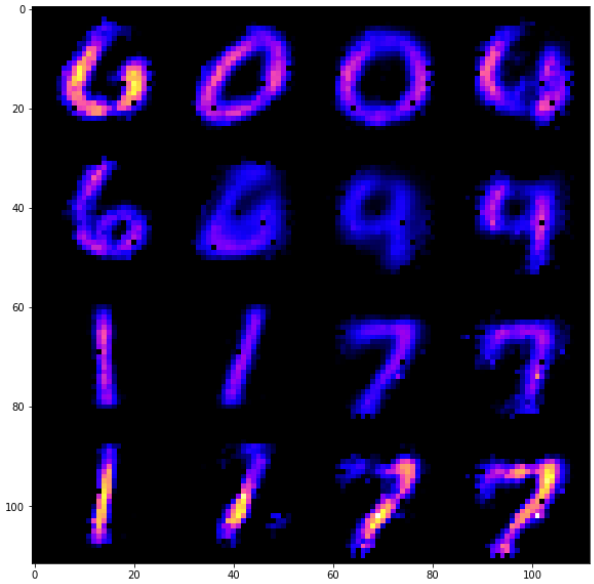


Figure 46: Generated digits from prior

4. The loss curve is quickly converging towards 0.250 around epoch 17 and only learning slightly after that. Training beyond that does not seem to influence the training/validation loss much.

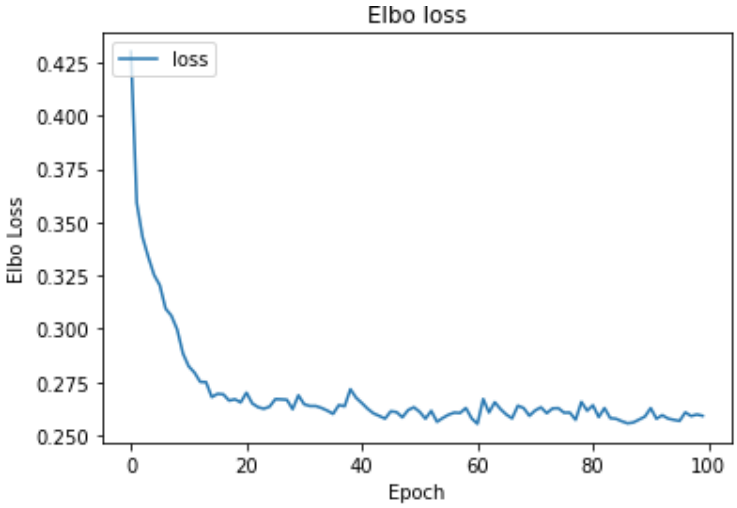


Figure 47: Loss Curve epoch vs Elbo loss

5. During the training process of the VAE with 32-dimensional latent space, we could see that the validation and training error was significantly reduced, which makes sense considering that we don't have to compress into a smaller representation. If we could have the original representation size as latent space size then our error would be zero.
(a) We randomly sampled from the normal normal distribution with mean 0 and std of 1. Interestingly most of our samples seem to be the number 9. And even though we use a 32-dimensional latent space there is a lot of noise in the images as in the 2-dimensional latent space VAE.

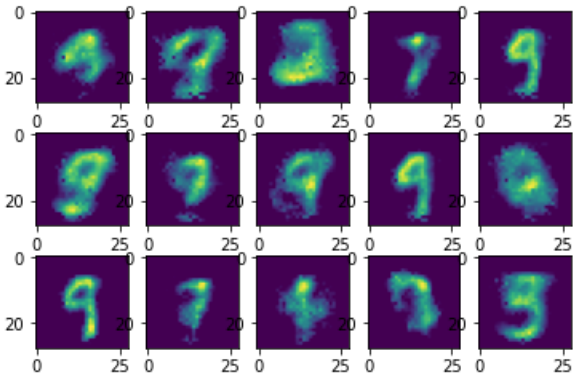
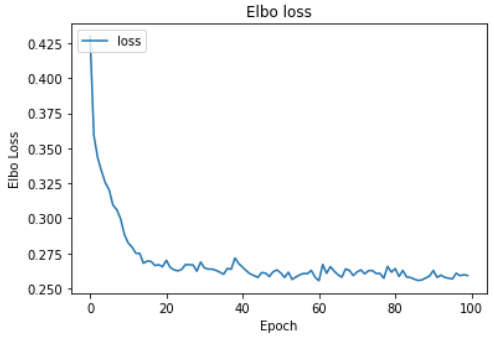


Figure 48: 15 generated images

(b) Our loss curve seems to learn for a much longer time compared to the 2-dimensional VAE. And the end result is a much lower loss. The 2-dimensional latent space loss curve is converging pretty fast, while the 32 VAE loss is still fluctuating.

```
plt.plot(h.history['loss'])
plt.title('Elbo loss')
plt.ylabel('Elbo Loss')
plt.xlabel('Epoch')
plt.legend(['loss'], loc='upper left')
plt.show()
```




```
]: plt.plot(h2.history['loss'])
plt.title('Elbo loss')
plt.ylabel('Elbo Loss')
plt.xlabel('Epoch')
plt.legend(['loss'], loc='upper left')
plt.show()
```

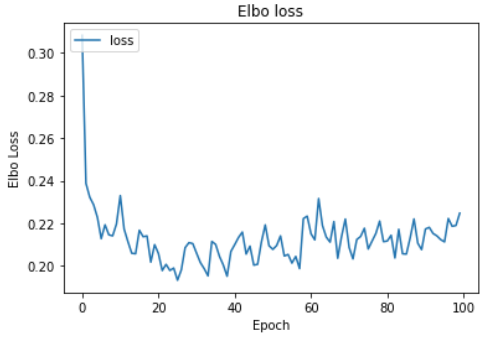


Figure 49: 2-Dimensional latent space VAE and 32-Dimensional latent space VAE loss

Report

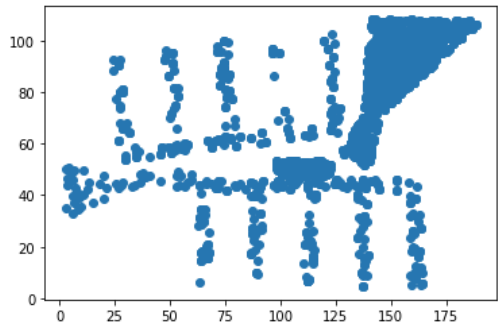
(a) It took 10 hours to implement the notebook, but a lot of time was used on understanding the tutorial, which was referred to in the exercise sheet, watching additional resources and reading the original VAE paper "Auto-Encoding Variational Bayes" which took up 10 hours. To test the methods we only needed 5 hours for this task.
(b) Dependent on the latent representation dimension and the amount of epochs we achieved a hugely varying degree of accuracy. The vae with only a single epoch with the two dimensional latent representation had a validation loss (KL Divergence + Reconstruction loss) of 0.417. In the end of the training with 100 epochs we converge around 0.2450. For the 32-dimensional latent space we were able to get much better results with a validation loss below 0.2 this reconstruction was much better.
(c) We have already worked with the MNIST dataset, but just generally the dataset consists of numbers from 0-9 and is a grey scale image not a RGB image. About the method we learned what a variational autoencoder is and the underlying Bayes theorem and the prior, likelihood and posterior. The given tutorial was a bit confusing, so I searched for an updated version, because the documentation for the tensorflow 1.15 is a bit outdated. I learned about the purposes of different activation functions and when to use them.

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

1. We started by downloading the dataset and loading it with numpy into a ndarray. Then we used pyplot to scatter plot the data with it's x and y position.

```
x_train=np.load('FireEvac_train_set.npy')
plt.scatter(x_train[:,0],x_train[:,1])
```

```
<matplotlib.collections.PathCollection at 0x1ce841f4748>
```




```
x_test=np.load('FireEvac_test_set.npy')
plt.scatter(x_test[:,0],x_test[:,1])
```

```
<matplotlib.collections.PathCollection at 0x1ce84a25dc8>
```

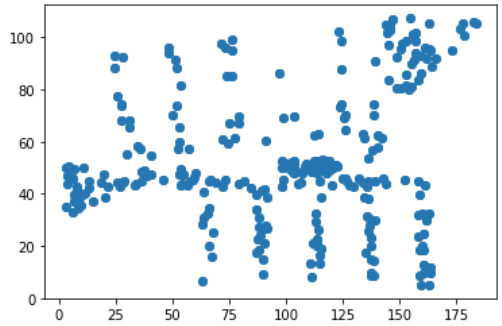


Figure 50: Training/Test Data Scatter plot

2. We began by rescaling the input data into the range of [-1,1] with the sklearn.preprocessing.MinMaxScaler. To adjust our VAE model to the new task, we tuned the hyperparameter hidden units per layer to 64 and epochs to 100. We achieved quite a low loss of 0.32.

3. The reconstructed test set was quite far away from the actual input data. It seems like a lot of information is lost during the transformation into z_mu and z_sigma, or the implementation was incorrect.

```
x2 = encoder.predict(x_test)
x2 = decoder.predict(x2)
x2 = x2.reshape(600,2)
# x2 = predict_encoder(x_test)
# x2 = predict_decoder(x2)

plt.plot(x_test[:,0], x_test[:,1], 'o')
plt.show()
plt.plot(x2[:,0], x2[:,1], 'o')
plt.show()
```

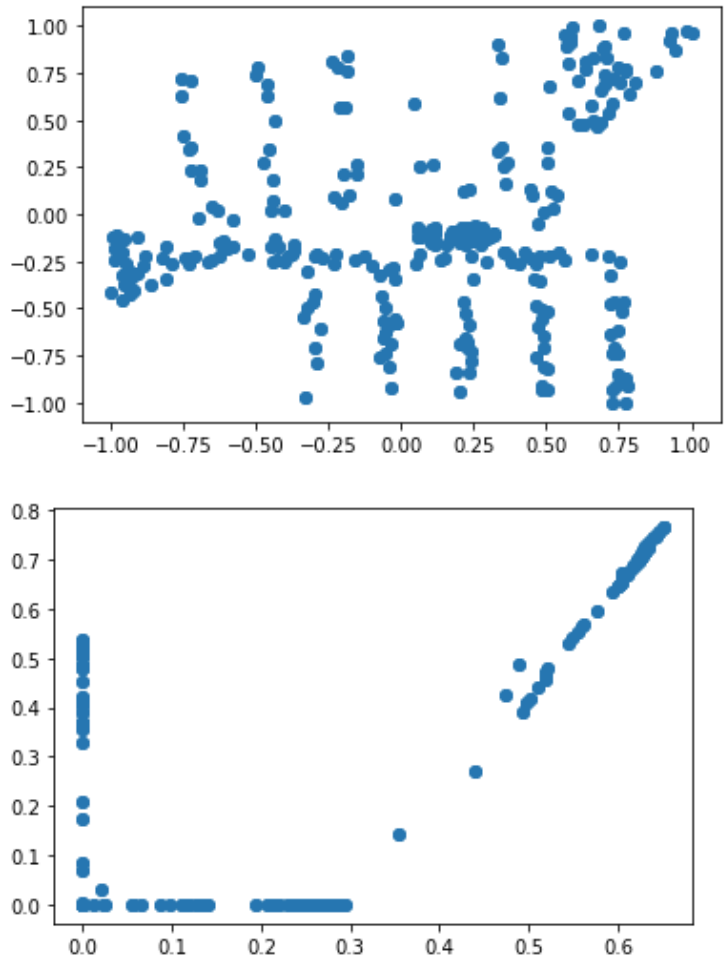


Figure 51: Reconstructed data points

4. We were trying to sample data from z as hown in fig 53. However, we were not able to implement it and then we decided to generate the samples using numpy.random.normal with a shape of (1000,2).

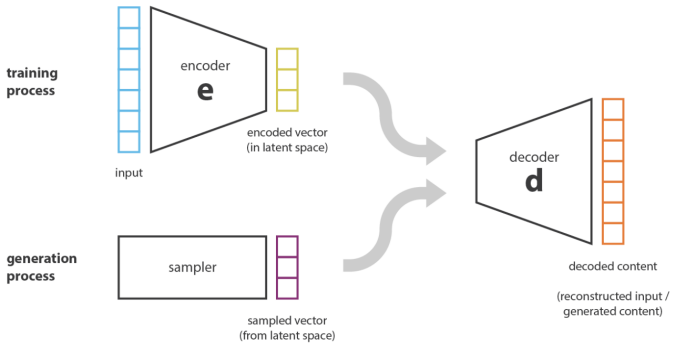


Figure 52: generate new data by decoding points that are randomly sampled from the latent space [1]

```
x3 = decoder.predict([np.random.normal(size = (1000,2))])

plt.plot(x3[:,0], x3[:,1], 'o');
plt.show()
```

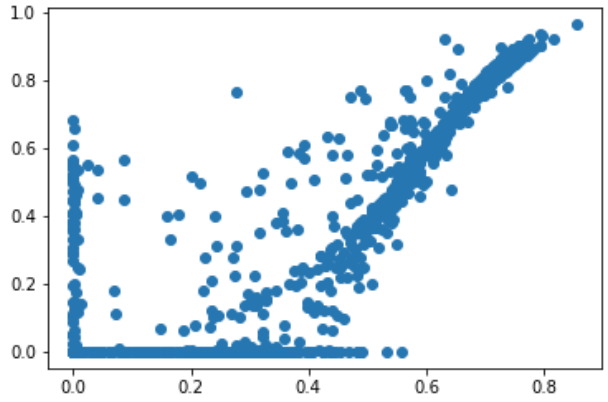


Figure 53: Generated data points

5. To calculate the critical number, we decided to vary the sample size between 1000 and 4000 with steps of 200. From each sample size, we would calculate the number of persons present between the region 130/70 and 150/50. If this number exceeded the 100, then we would consider the sample size as critical number.

Report

(a) It took 5 hours to implement the notebook. Testing took 5 hours as well, since we had a bug in the loss function.
(b) We weren't able to accurately reconstruct the data. For measure of accuracy we used the elbo loss, which went down to 0.32.
(c) About the VAE it was interesting to see that not all of the data is maintained, even though we have a higher amount of hidden units than the input dimension.

[1] Understanding Variational Autoencoders (VAEs) (https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73)