

Machine Learning for Signal Processing

[5LSL0]

Rik Vullings
Ruud van Sloun
Nishith Chennakeshava
Hans van Gorp

Answers: (Variational) Autoencoders

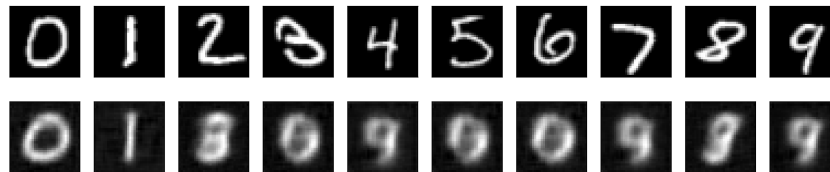
April 2023

This document outlines all the answers to the questions that are textual and/or graphical. As a companion to this answer sheet, we also provide the code answers separately. If an answer is (partially) provided as code we use a 📄 icon.

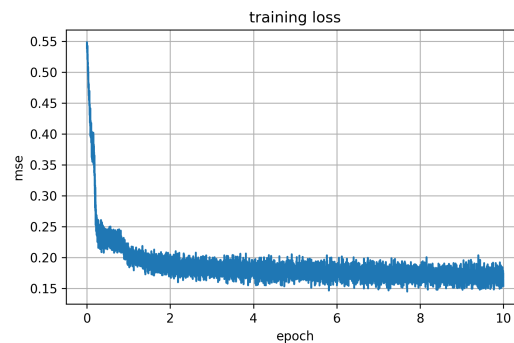
There was quite some implementation freedom in these exercises, so do not worry if your implementation is slightly different from the ones provided by us. In general terms, results should be comparable to ours, so latent spaces should show some clustering for deterministic autoencoder and better clustering for VAE. Likewise, the classification of digits should work a bit with the autoencoder, but much better with the classifier.

Exercise 1

- [2 pt] Code 📄
- [1 pt] The autoencoder seems to work well for zero and one, unfortunately other digits seem to be jumbled up into weird zeros and nines.

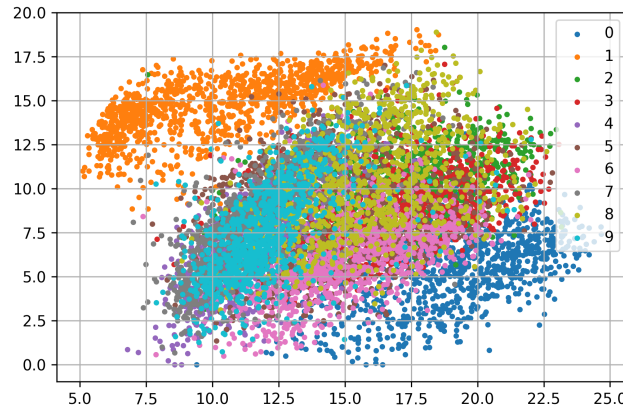


- [1 pt] Loss plot:



Exercise 2

(a) [2 pt] Code 



- (b) [0.5 pt] In particular the ones and zeros seem to yield relatively separate clusters, while many other have strong overlap. We can see that the zeros have some overlap with the sixes, which can be explained by them both having circles in their shapes.
- (c) [0.5 pt] Clipping occurs because of the ReLU activation that clips negative weights to zero. In our case, we can see some clipping on the y-axis.

Exercise 3

(a) [2 pt] Code 

Nearest-neighbour classification accuracies are:


- for digit 0, accuracy is 70.2
 - for digit 1, accuracy is 93.4
 - for digit 2, accuracy is 27.4
 - for digit 3, accuracy is 22.1
 - for digit 4, accuracy is 22.9
 - for digit 5, accuracy is 14.9
 - for digit 6, accuracy is 28.4
 - for digit 7, accuracy is 28.2
 - for digit 8, accuracy is 22.4
 - for digit 9, accuracy is 24.1
- (b) [1 pt] 10% for each of the digits. Especially the zero and one are detected much more accurate. The reason is that these digits produce distinct clusters in the latent space, reducing the probability that another digit is the nearest neighbour in the latent space.
- (c) [0.5 pt] Categorical crossentropy

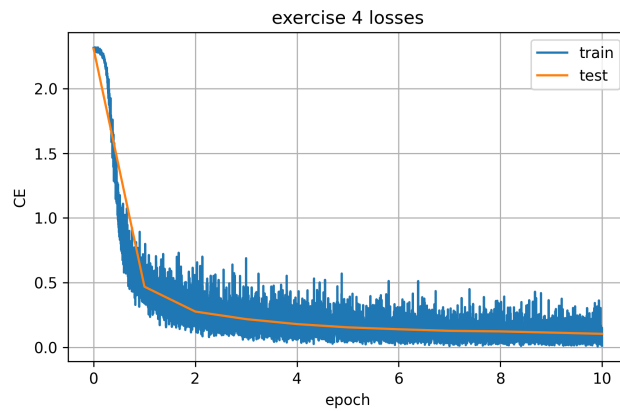
Exercise 4

(a) [2 pt] Code 

Neural network classification accuracies are:

- for digit 0, accuracy is 98.7
- for digit 1, accuracy is 99.1
- for digit 2, accuracy is 96.5
- for digit 3, accuracy is 97.4
- for digit 4, accuracy is 96.3
- for digit 5, accuracy is 97.4
- for digit 6, accuracy is 97.1
- for digit 7, accuracy is 96.6
- for digit 8, accuracy is 96.6
- for digit 9, accuracy is 95.5

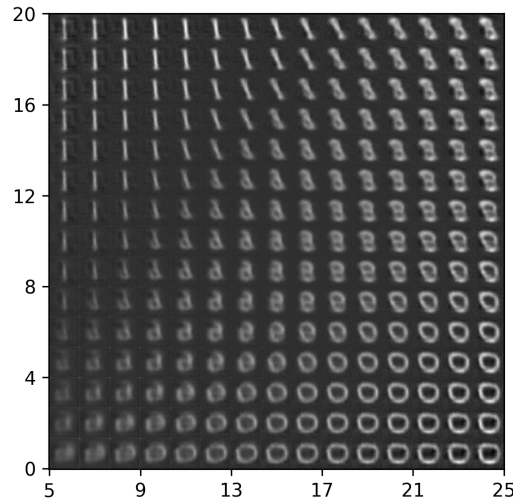
(b) [1 pt]  The capacity seems to be just right. There is no under or overfitting



(c) [1 pt] In case of underfitting, increase capacity of the network. In case of overfitting, apply regularization

Exercise 5

(a) [2 pt] Code 

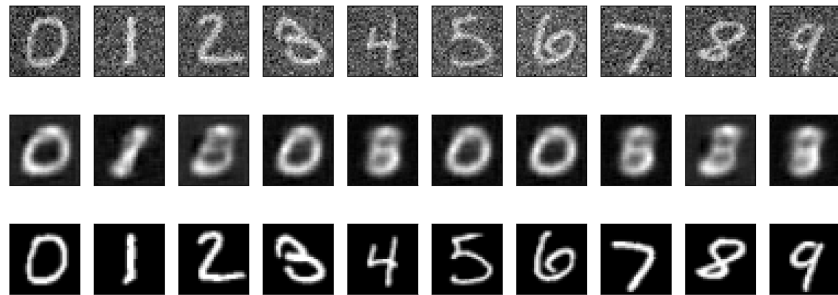


- (b) [1 pt] Most distinguishable are the zeros in the bottom right and the ones in the top left corner. In the top right and bottom left corner, the images do not resemble any real digits. From bottom right to top left, the digits "transition" from 1 to zero via a blur of various other digits.

These images are fully in line with the scatter plot of Exercise 2, where separate clusters for one and zero are seen in the same place as for the images. The scatter plot shows no data in the top right and bottom left corners of the latent space. Therefore, the decoder has not been trained on this part of latent space and hence produces results that do not relate to digits.

Exercise 6

(a) [2 pt] Code 

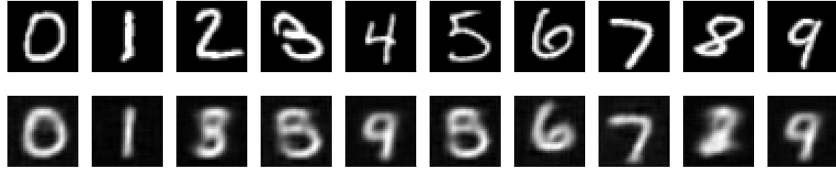


We can observe that the noise does indeed not map through the bottleneck; the autoencoder has successfully learned to ignore it. However, the reconstructed images do not look like the original clean images at all. Probably due to the limited capacity of the bottleneck. Interestingly enough, the one and zero do stay a one and zero, although they have been transformed in style and rotation. We can conclude that this specific autoencoder does not work very well for denoising purposes.

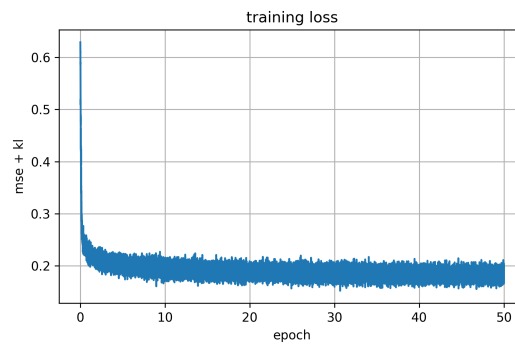
Exercise 7

(a) [4 pt] in total, split into three parts:

- [2 pt] Code 📄
- [1 pt] image:

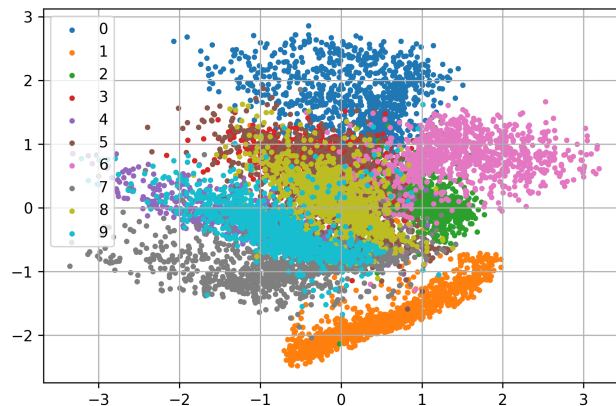


- [1 pt] image:



Note: if you did not include a Lagrange multiplier (in the code: $\beta = 0.01$) to the loss function, the results will look way worse. Deduct one point if you forgot this hyper-parameter

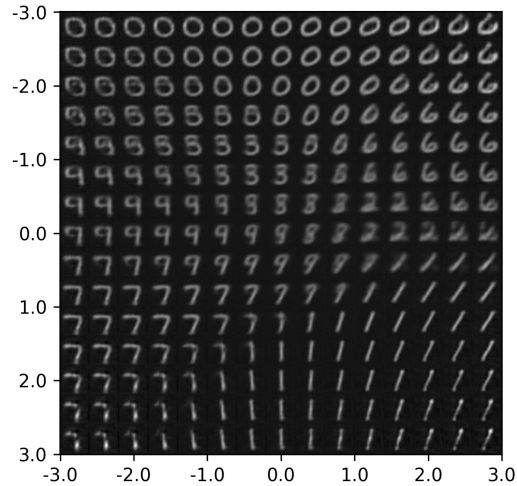
(b) [1 pt] Code 📄



(c) [1 pt] The classes are much better separated, (but not completely). Moreover, the latent space is of much smaller magnitude, and clustered around the origin. The latent space of a VAE is supposed to follow a normal distribution. However, here we are plotting only the means, and not necessarily the means+standard deviation.

Depending on your choice of the balancing weight between the mse-loss and the kl-loss the means will be closer to zero (favoring the kl-loss), or further apart (favoring the mse-loss).

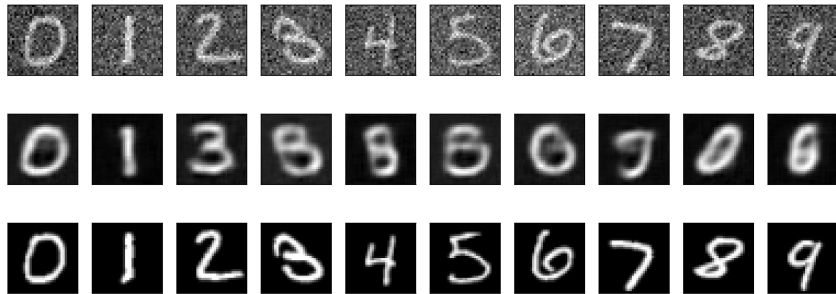
(d) [1 pt] Code 



(e) [1 pt] The are much more digits recognizable in the latent space plot of the VAE. This alligns with the scatter plot of the VAE where much more class seperation occurs. The reason for the better performance in this regard of the VAE over the AE is due to the extra regularization added to the latent space by the KL-loss. Moreover, we no longer have a ReLU activation acting on the latent space, meaning that the latent codes are no longer cut off at zero.

Exercise 8

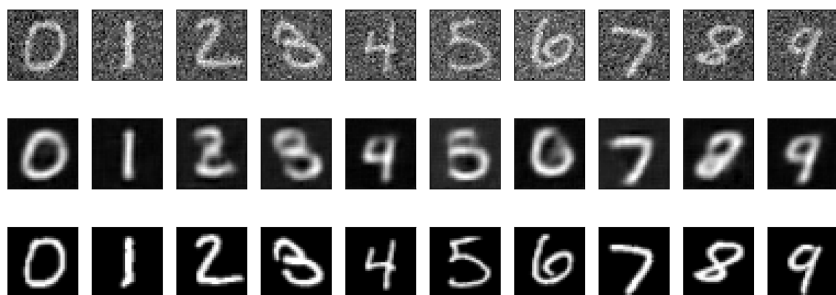
(a) [1 pt] Code 



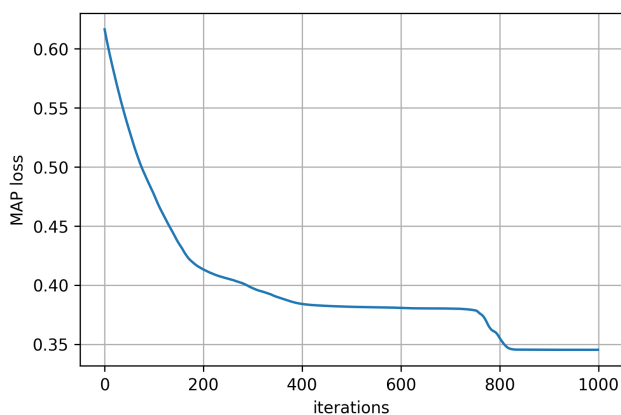
(b) [4 pt] total, split into three parts:

[2 pt] Code 

[1 pt] for image:



[1 pt] for image:



(c) [1 pt] MAP optimization clearly works better than the bottleneck method. This makes sense, as the bottleneck method has no penalty from diverging from the measurements. For example we can see that the two maps to a three using the bottleneck method, while the two stays a two using MAP estimation. Moreover, the bottleneck method also has no regularization on the latent space, thereby allowing unlikely outcomes, e.g. the strange four and nine for the bottleneck method.