Deep Learning Assignment 3

Oscar Ligthart
10557520
University of Amsterdam
oscarligthart@gmail.com

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

All methods use latent variables to apply dimensionality reduction of the data. With the difference that VAE is able to apply non-linear dimensionality reduction, while pPCA and FA do not.

1.2 Decoder: the Generative Part of the VAE

Question 1.2

In order to get a sample x_n, first you sample from distribution $p(z_n)$. Thereafter, you insert the sampled value into the neural net f_θ. Finally, the output of the neural net will be inserted into the Bernouilli function, returning a sample value of x_n

Question 1.3

The reason that the made assumption is not so restrictive is that a sufficiently complicated function (the neural net) on a Gaussian distribution ($p(Z)$) can approximate nearly all other distributions. During training the network should learn the parameter θ, such that the data distribution will be represented in the chosen distribution $p(Z)$. Therefore, all values in $p(Z)$ will represent some part of the data.

Question 1.4a

Monte-Carlo integration comes down to taking a set amount of samples to approximate a certain probability distribution.

$$\log(\mathbf{x}_n) \approx \frac{1}{K} \sum_{k=1}^{K} \log p(\mathbf{x}_n | \mathbf{z}_n^{(k)}) \tag{1}$$

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Question 1.4b

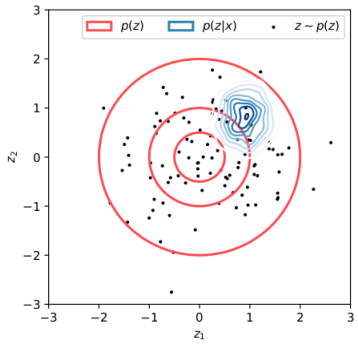

Figure 2. Plot of 2-dimensional latent space and contours of prior and posterior distributions.

The figure above shows an example where the posterior is known. This is the region (for a certain x) where the probability would be high. If the posterior is unknown, samples have to be taken from the whole latent space $p(Z)$. Many of these samples will have a probability that is very close to 0 and will thus be uninformative regarding a certain data class. In order to get a good approximation of the posterior, you would have to take a large amount of samples from the latent space. Hence, this approach is very inefficient. Furthermore, if the dimensionality of the latent space z increases, so does the amount of samples that need to be taken from the latent space to correctly approximate the posterior.

1.3 The Encoder: $q_\phi(z_n|x_n)$

Question 1.5a

An example of a very small KL-divergence would be to have two identical distributions, due to the fact that $\log \frac{q(x)}{p(x)}$ would result into 0. This leads to a KL-divergence value of 0. In order to get this result, (μ_q, σ_q^2) would have to be $(0, 1)$.

In order to get a high KL-divergence, the mean would for instance have to be a much larger value. While the standard deviation would have to be very small. This means that the distributions are very far away from each other and thus have minimal overlap.

Question 1.5b

$$D_{KL}\big(\mathcal{N}(\mu_q, \sigma_q^2)||\mathcal{N}(0,1)\big) = \log \frac{\sigma_q}{\sigma_p} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2} \tag{2}$$

$$= \log \sigma_q + \frac{\sigma_q^2 + (\mu_q)^2}{2} - \frac{1}{2} \tag{3}$$

$$= \log \sigma_q + \frac{\sigma_q^2 + (\mu_q)^2 - 1}{2} \tag{4}$$

Question 1.6

$$\log p(\mathbf{x}_n) - D_{KL}\big(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)\big) = \mathbb{E}_{q(\mathbf{x}_n|Z)}[p(\mathbf{x}_n|Z)] - D_{KL}\big(q(Z|\mathbf{x}_n)||p(Z)\big) \tag{5}$$

The value of the KL-divergence $D_{KL}\big(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)\big)$ can either be 0 or a positive value. Since this value is subtracted from $\log p(\mathbf{x}_n)$, the right-hand-side of the equation can either be the same value as $\log p(\mathbf{x}_n)$ or a smaller value. Therefore, it is called the lower bound of $\log p(\mathbf{x}_n)$.

Question 1.7

For optimizing $\log p(\mathbf{x}_n)$ directly, you will need $D_{KL}\big(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)$. However, obtaining an analytical form of $p(Z|\mathbf{x}_n)\big)$ is intractable. The KL-divergence between the distributions $q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)\big)$ will therefore be too tough to obtain, meaning the loss function will be very inefficient or even unobtainable.

Question 1.8

The following two things can happen when the lower bound is pushed up:

1. The KL-divergence between q and p decreases, meaning the distributions will be closer to each other.

2. The log-likelihood $\log p(x_n)$ of the data increases.

1.4 Specifying the encoder $q_\phi(z_n|x_n)$

Question 1.9

The name reconstruction is appropriate for the loss due to the fact that it ensures that the decoder reconstructs a data instance x given a code z.

The name regularization is appropriate due to the fact that this loss term ensures that the latent space variables are efficient descriptors of the input. The algorithm is penalized for straying from a Gaussian distribution.

Question 1.10

For the reconstruction loss:

$$L_n^{recon} = -\mathbb{E}_{q_\phi(z|\mathbf{x}_n)}[\log p_\theta(\mathbf{x}_n|Z)] \tag{6}$$

$$L_n^{recon} \approx -\frac{1}{K}\sum_{k=1}^{K} \log p_\theta(\mathbf{x}_n|\mathbf{z}_n^{(k)}) \tag{7}$$

This probability will be highest if the generated image is the same as the input, therefore we use binary cross entropy to approximate it.

$$L_n^{recon} \approx -\frac{1}{K}\sum_{k=1}^{K}\sum_{m=1}^{M} \Big((\mathbf{x}_n)_m \log(\hat{\mathbf{x}}_n)_m + (1-(\mathbf{x}_n)_m)\log(1-(\hat{\mathbf{x}}_n)_m)\Big) \tag{8}$$

For the regularization loss:

$$L_n^{reg} = \sum_{j=1}^{J} D_{KL}\big(\mathcal{N}(\mu_j, \sigma_j^2)||\mathcal{N}(0,1)\big) \tag{9}$$

$$L_n^{reg} = \sum_{j=1}^{J} \log\sigma_j + \frac{\sigma_j^2 + (\mu_j)^2 - 1}{2} \tag{10}$$

1.5 The Reparametrization Trick

Question 1.11a

To backpropagate through the model and update the weights of the ϕ and θ parameters.

Question 1.11b

Backpropagation is not possible through stochastic units in a network (sampling). This is due to the fact that sampling is a non-continuous operation and therefore has no gradients.

Question 1.11c

The reparametrization trick moves the stochastic unit to an input layer. Sampling from $\mathcal{N}(\mu(X), \Sigma(X))$ is possible by first sampling $\epsilon \sim \mathcal{N}(0, \mathcal{I})$, which represents the input layer. Then, $Z = \mu(X) + \Sigma^{\frac{1}{2}}(X) \times \epsilon$ is computed.

1.6 Putting things together: Building a VAE

Question 1.12

For both the encoder and decoder I chose to just implement a single hidden layer holding 500 nodes. The activation function of this layer was a ReLU. For generating the mean and log variance in the encoder, no activation function was used. For a pixel wise mean value, a sigmoid was used to push values between 0 and 1. The reconstruction loss consisted of binary cross entropy loss and the KL-divergence was calculated using Appendix B of the paper by Kingsma and Welling (2014).

A big difference in implementation I chose for is letting the neural net output a mean as well as a log variance, instead of a mean and standard deviation. The reason for this choice is the fact that a log variance can also consist of negative values, whereas the standard deviation only consists of positive values. This gives the neural net the freedom to also output negative values, meaning output will be around 0. We can then just take the exponent of this value to retrieve the standard deviation. Moreover, it becomes numerically stable, since we don't have to take the log of positive bad values anymore.

Question 1.13

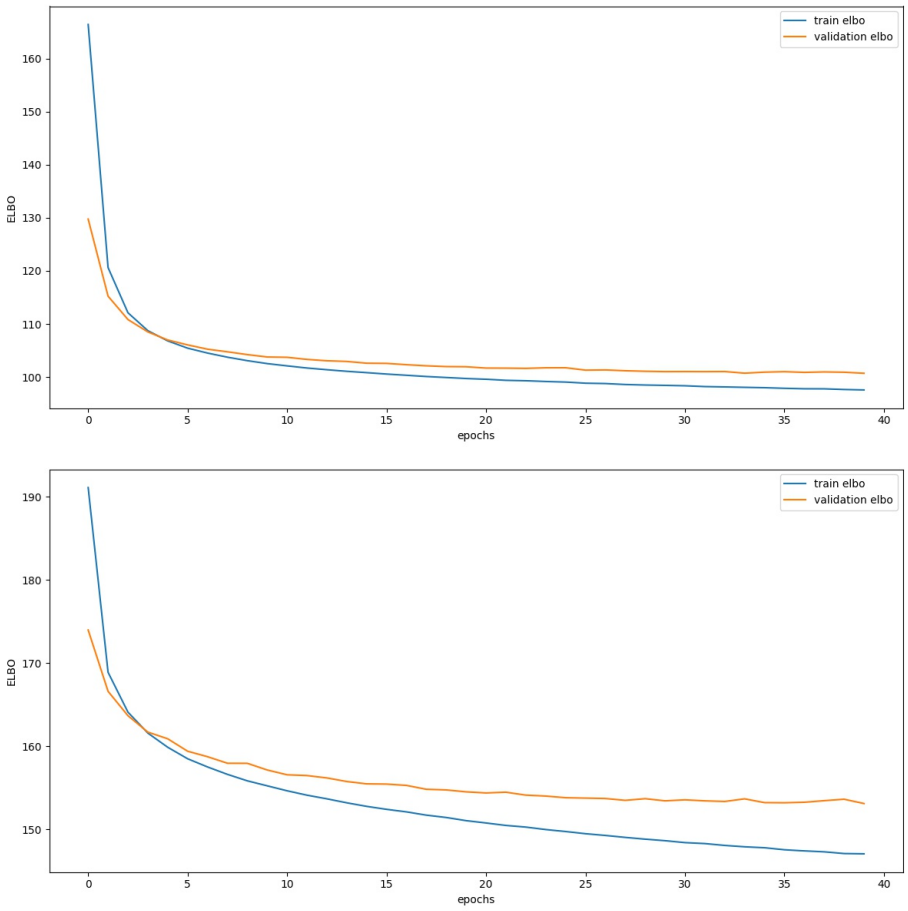

Figure 1: ELBO curves for latent space dimensionality 20 (upper) and 2 (lower)

Question 1.14

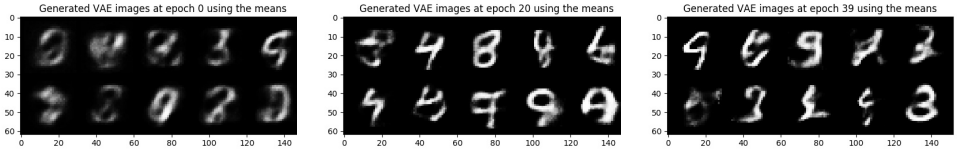

Figure 2: Samples during training with latent dimension 2.

Figure 2 shows 10 samples before, halfway through and after training. As can be seen by the figures, the model definitely seems to be learning and the results are improved over time. However, the difference between the model halfway through and after training does not seem to be that significant. What's more, it seems that the results are slightly worse after training. This might be due to the fact that the model has overfitted towards the data (as can be seen in the loss curve in figure 1. Another reason could be that the samples taken halfway of the training are better represented in the latent space compared to the samples taken after training. The results definitely show numbers, but sometimes the model creates samples that seem to be quite noisy or a mixture of other numbers. The large amount of dimensions might be responsible for that phenomenon, due to the fact that the space is too complex or vast to represent every data class in a readable manner. Samples that lie in between representations of the data will create mixtures of classes, resulting in unreadable characters.

Due to the fact that 20 dimensions might be too large for a problem of this size, I decreased the latent dimension to 2 and ran the test again. Results can be found in figure 3.

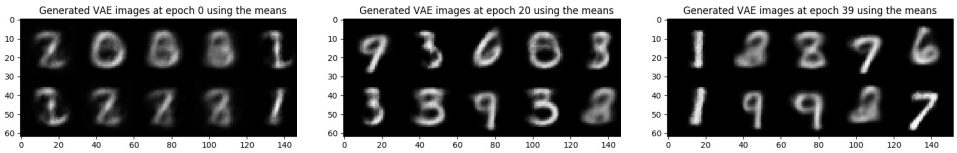

Figure 3: Samples during training with latent dimension 2.

The improvement over time also becomes apparent in these images. The first epoch does show some signs of numbers forming (like a 2, 8, 1 or 0) but they are not really readable. After training for a few epochs, numbers look very nice and can be easily classified and distinguished from each other. Due to the fact that the latent space now consists of only 2 dimensions, the variations on the numbers decrease and the numbers are more easily distinguishable from each other compared to the results of latent dimensionality 20.

Question 1.15

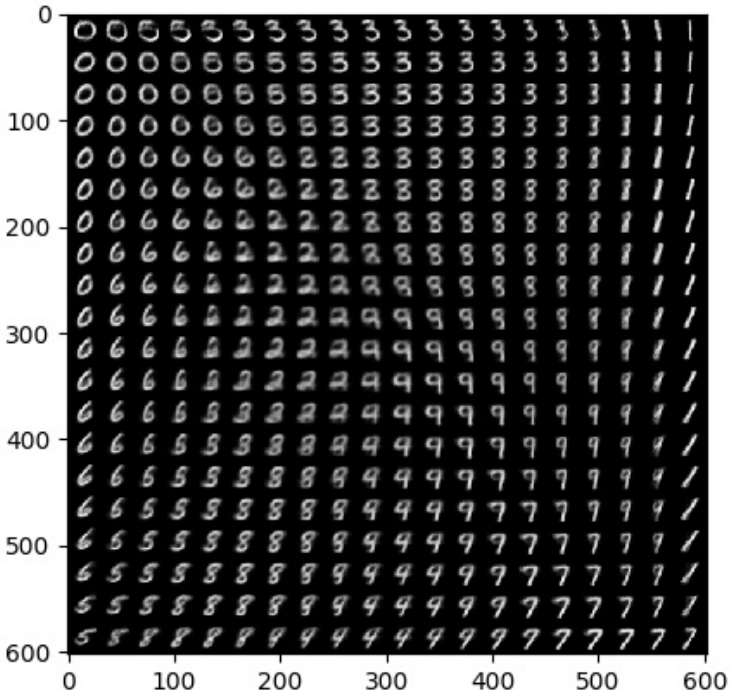

Figure 4: Manifold of the latent space

The manifold of the data in latent space is shown in figure 4. Linearly spaced coordinates on the unit square were transformed into values of latent variable z through the percent point function. Note that every data class is present within this manifold and the transition to neighbouring instances looks very smooth.

2 Generative Adverserial Networks

Question 2.1

Generator:

Input is some point sampled from a Gaussian distribution. Output will be a generated image of the size of the training images.

Discriminator:

Input can either be a fake image generated by the generator or a real image from the training set. Output will be a probability of the image being either fake or real. It will consist of a number between 0 and 1 where 0 can represent fake and 1 represent real.

2.1 Training objective: A Minimax Game

Question 2.2

For the first term you want to maximize on the data such that the discriminator learns proper representations of the images from the training set.

For the second term you want to minimize such that the discriminator will classify the generated image as being real. The output of the image should

Question 2.3

The value for $V(D, G)$ will be $\log(0.5) + \log(0.5)$ after training has converged. Intuitively, you want the generator to generate fake samples that look so realistic they will not be distinguishable from real images. This will lead to the discriminator having to guess, since the fake images look so realistic. Since it can only guess between two options, fake or real, the guessing probability will be 0.5 in both terms of the loss equation.

Question 2.4

Early on during training it is very easy for the discriminator to distinguish fake images from real ones since the fake once are very noisy. This will cause the term to become $log(1)$, which will result into a value of 0, meaning that the gradients could vanish. These vanishing gradients only occur for the generator, meaning that the discriminator will only get better and thus always predict the generated image to be fake (leading to more vanishing gradients for the generator). To prevent this, instead of minimizing $\log(1 - D(G(z)))$, maximize $\log(D(G(z)))$. This way the same optimum will be reached while the generator has much stronger gradients early on.

2.2 Building a GAN

Question 2.5

I created 2 separate implementations:

For the network architecture I used the given starting model architecture. As a loss function I first applied the binary cross entropy loss. To optimize the generator the generated image would be inserted into the discriminator and the BCE loss was calculated. To optimize the discriminator a batch of real images or a batch of fake images would be used at random. Furthermore, soft labelling was used to update the discriminator with. The reason for doing so was to make the task a little harder for the discriminator, so the stability of the model would be improved.

As a further improvement I implemented a MiniMax algorithm according to the lecture. I altered the loss function for the discriminator to $-\frac{1}{2}\mathbb{E}_{x\sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_z \log D(G(z))$ and treated it as a ZeroSum game, where the generator loss would just be the negative value of the discriminator loss. This way the improvement of one would be considered to be a direct decline for the other. Normally this is not used on tasks because of simplicity, but since this problem is not that difficult I implemented it anyway.

Question 2.6

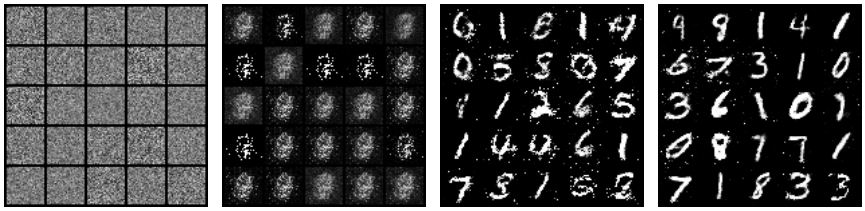

Figure 5: Results for first implementation. From left to right: Untrained, during first epoch, halfway through training and after training

As can be seen in figure 5, the model is able to learn a correct representation of the data and reconstruct it in a sufficient manner. The numbers can definitely be recognized, but the pictures still hold noisy white pixels within them.

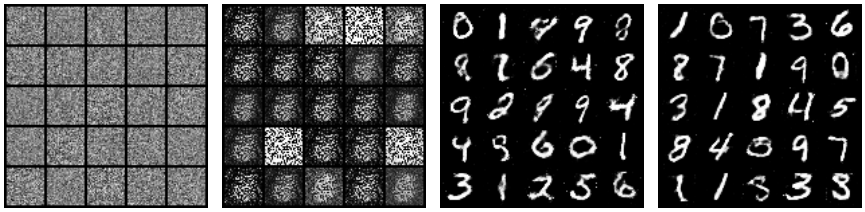

Figure 6: Results for second implementation. From left to right: Untrained, during first epoch, halfway through training and after training

Compared to the results of figure 5, the results of the second implementation (shown in figure 6) seem to be improved. The numbers can be separated from each other in a clear manner and the images do not hold many noisy pixels. Furthermore, the model seems to learn faster as the numbers look much better halfway during training in this model compared to halfway during training in the previous model.

Question 2.7

Figure 7: Interpolation between the classes 5 and 2.

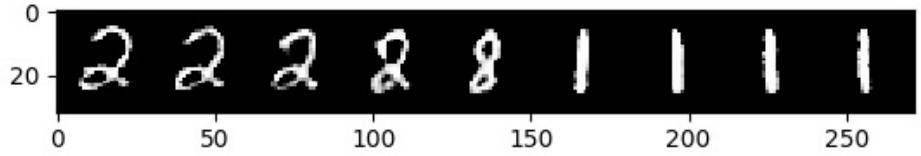

Figure 8: Interpolation between the classes 2 and 1.

Figure 9: Interpolation between the classes 7 and 5.

Figure 7 shows the interpolation between two images showing the number of class 5 and 2. Note that, in latent space, the number 3 resides between these classes.

3 Conclusion

Both the VAE model and the GAN are correctly able to map data into a latent space in an unsupervised matter. Subsequently, they can correctly generate data by sampling from this latent space. However, the manner in which they do so differs. The VAE compresses the data in a latent space that has much fewer dimensions compared to the input data and is trained to reproduce this data. Therefore it can be seen as a compression algorithm. The GAN works as a sort of inside out VAE, where the input

consists of a low dimensional vector (can be seen as latent space) and it produces an entire image out of that small "code". The discriminator then learns the generator to make more convincing data, while the generator does not get to see this data. The GAN seems to perform a little bit better when it comes to image quality, but it takes a longer time to train due to the fact that learning depends on both the generator and discriminator. A reason for these differences could be that the VAE is a probabilistic graphical model with the explicit goal of latent modelling (and reconstructing data), while the GAN is set up to optimize on generative tasks (generate new realistic samples based on seen data).