Deep Learning Assignment 3

Philipp Lintl 12152498
philipp.lintl@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

1. Standard Autoencoders encode input of all sorts by compressing it into a smaller feature vector. This latent feature vector is then decompressed or decoded back to the original input size to reconstruct the input. This procedure in general is regarded as self-supervised/unsupervised, as the encoding and decoding parameters are learned according to the reconstruction loss with regards to the original input. The compressed representation of the input data can then be used for all kinds of tasks or even as a mean of pretraining to initialize the weights and save training efforts. Variational Autoencoders on the other hand enhance the mere compressed representation to a generative model. Generative in the sense, that the probability distribution with respective parameters of the input data is learned to then generate new input samples similar to the original input data.
2. Standard Autoencoders are in fact not suitable to generate instances similar to the input data. This is due to the fact that in a Autoencoder, by compressing and decompressing the original input according to reconstruction loss, only a discontinuous latent representation of the input data is learned. This representation does not enable to model the probability distribution of the latent variable, which can be done by a VAE. More precisely, the standard Autoencoder learns a deterministic representation, meaning that for feeding the same input through the network repeatedly, will lead to the same output after passign the latent space and being decompressed by the decoder. VAE's on the other hand incorporate non-deterministicality by placing prior distributions on the latent parameters. That way, by inputing the same observation several times, the encoding depends on the randomness of the sampling procedure.

1.2 Decoder: The Generative Part of the VAE

Question 1.2
Ancestral sampling for probabilistic graphical models describes the procedure to sample from parent nodes before children nodes. The given VAE incorporated ordering of the nodes, as $\mathbf{z}_n$ is the root to $\mathbf{x}_n$. Thus, we have a a directed acyclic graph, which enables to sample via ancestral sampling. More precisely, we begin by sampling from the latent variable $\mathbf{z}_n$ according to it's Gaussian distribution:

$$\mathbf{z}_n \sim \mathcal{N}(0, \mathbf{I}_D)$$

This sample $\mathbf{z}_n$ is then passed through the decoder network f_θ to yield $f_\theta(\mathbf{z}_n$, which maps $\mathbf{z}_n$ to the parameters of the distribution p_X. As independence among the pixels given $\mathbf{z}_n$ is assumed, solely the m-th element of the latent variable sample $\mathbf{z}_n$ is required to sample $\mathbf{x}_n^{(m)}$, the m-th pixel of the n-th image. For all m pixels this means, we get the n-th sample by:

$$\mathbf{x}_n^{(m)} \sim Bern(f_\theta(\mathbf{z}_n)_m) \qquad \forall m = 1, \ldots, M$$

Submitted to the Winter 2019 UVA Deep Learning course

Question 1.3
Eventhough the standard normal Gaussian prior on the latent variable $\mathbf{z}_n$ is a rather simple and little expressive distribution, it is not considered a restrictive assumption. This is mainly due to the fact that a Neural Network is used to map the samples from the standard normal distribution to the space needed as input to generate $\mathbf{x}_n$ samples. In other words, given the decoding network $f_\theta()$ is chosen to be of high expressiveness/complexity, its capabilities as a function approximator suffice to transform the standard normal inputs to inputs suitable to all kinds of complex distributions that might be chosen to represent the data.

Question 1.4 (a)
As given in the hint, Monte-Carlo integration enables us to approximate the originally intractable integral of the posterior. At first, the latent variable is sampled as such:

$$\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_D), \qquad \forall i = 1, \ldots, N$$

According to MC, we then get the approximation as such:

$$\log p\left(\mathbf{x}_n\right) = \log \mathbb{E}_{p(z_n)}\left(p\left(\mathbf{x}_n|\mathbf{z}_n\right)\right) \tag{1}$$

$$\approx \log\left(\frac{1}{L}\sum_{i=1}^{L} p\left(\mathbf{x}_n|\mathbf{z}_n^{(i)}\right)\right), \tag{2}$$

whereas L denotes the number of samples for F.

Question 1.4 (b)
As denoted in (a), the expected value is estimated by taking L samplef of the latent variable according to the standard normal prior. These samples are then conditioned on, to yield the respective probability for $\mathbf{x}_n$. Averaging over all L samples is then supposed to yield an approriate approximation of the data log likelihood. Inefficiency in this approach arises, when we consider Figure 1 in the assignment sheet. According to it, the region with high posterior probability (visualized by the blue contours) can be much smaller than the region of the prior (visualized by red contours). Thus, for most samples $\mathbf{z}_n^{(i)}$ will yield low probabilities for $p(\mathbf{x}_n|\mathbf{z}_n^{(i)})$. This means, that in order to estimate $p(\mathbf{x}_n)$ properly, a lot of samples will be required as most of the samples do not factor in much in the sum of the estimation, making the procedure inefficient. The larger the dimensionality of $\mathbf{z}_n$, the larger the inefficiencies, as even more sampling will be required. The latent variable is modeled by a Bernoulli distribution and thus of dimensionality $\{0,1\}^M$. As $p\left(\mathbf{x}_n|\mathbf{z}_n^{(i)}\right)$ involves multiplication over all of these M dimensions, scaling will be exponential in the dimension of $\mathbf{z}_n$

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

Question 1.5
Given the two univariate Gaussians $q = \mathcal{N}(\mu_q, \sigma_q^2), p = \mathcal{N}(0, 1)$.
(a) Give two examples for (μ_q, σ_q^2) that will yield a very small and one very large KL Divergence between q and p.
Considering the KL divergence definition:

$$D_{\mathrm{KL}}(q\|p) = -\mathbb{E}_{q(x)}\left[\log\frac{p(X)}{q(X)}\right]$$

we clearly see that the KL divergence will be smallest possible, if the two distributions are the same. Thus, $(\mu_q, \sigma_q^2) = (0, 1)$ and accordingly $D_{\mathrm{KL}}(q\|p) = 0$. As the assignment sheet asks for a very small (possibly not zero) value though, one could also consider $(\mu_q, \sigma_q^2) = (0.5, 1)$. $Larger D_{\mathrm{KL}}(q\|p)$, will be achieved, the more the two distributions differ from each other. For instance by setting the mean of q very different from p: $(\mu_q, \sigma_q^2) = (5, 1)$. That way, the two

distributions will not have a large overlap and therefore a very large KL divergence.
(b) For two Gaussian distributions p_1, p_2 with parameters (μ_1, σ_1^2) and (μ_2, σ_2^2) is

$$D_{KL}(p_1 \| p_2) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \tag{3}$$

The closed form formula for $D_{KL}(q\|p)$, given that p is standard normal and q is Gaussian with mean μ_q and variance σ_q^2 thus arises as:

$$D_{KL}(q\|p) = \log \frac{1}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q)^2}{2} - \frac{1}{2} \tag{4}$$

$$= \frac{1}{2}\left(-\log \sigma_q^2 + \sigma_q^2 + \mu_q^2 - 1\right) \tag{5}$$

Question 1.6
One property of KL divergence is non-negativity. This means, that the term on the left $D_{\mathrm{KL}}\left(q\left(Z|\mathbf{x}_n\right)\|p\left(Z|\mathbf{x}_n\right)\right) \geq 0$. Thus, looking at equation (11) from the assignment sheet and taking into account, that from the log-probability is subtracted a term larger or equal to zero, we get:

$$\log p\left(\mathbf{x}_n\right) \geq \mathbb{E}_{q(z|\mathbf{x}_n)}\left[\log p\left(\mathbf{x}_n|Z\right)\right] - D_{\mathrm{KL}}\left(q\left(Z|\mathbf{x}_n\right)\|p(Z)\right) \tag{6}$$

Therefore, the right side of equation (11) is a lower bound on the log-probability $\log p\left(\mathbf{x}_n\right)$.

Question 1.7
Optimizing the log-probability directly would entail optimizing over an intractable integral, more precisely the true posterior distribution, which has already been mentioned above. Not having an analytical solution does not allow for optimization with respect to according parameters. Therefore, if we optimize the lower bound, that has just been shown, the log-probability is implicitly optimized as well. This is due to the fact, that the respective log-probability to the optimized lower bound will be at least, that minimal value.

Question 1.8
In order to move the lower bound on the right hand side up, the terms on the left hand side need to increase as a whole, as well. This happens either, if the log-probability/data likelihood $\log p(\mathbf{x}_n)$ increases. This is what we initially wanted, as a larger log-probability means that the network can generated data more similar to the original input from sampling z. Secondly, the left hand side would increase, if the KL divergence $D_{\mathrm{KL}}\left(q\left(Z|\mathbf{x}_n\right)\|p\left(Z|\mathbf{x}_n\right)\right) \geq 0$ decreases, as then a smaller term is subtracted from the log-probability. A smaller KL divergence between the original posterior p and the approximated posterior q means that they are more alike than before. Again, this is wanted, as the closer the approximated posterior to the original posterior, the better the latent space representation will be, which in turn is crucial to generate high quality output.

1.4 Specifying the encoder $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.9
Considering the hint to only use one sample, the total loss of $\mathcal{L}$ equals the sum of $\mathcal{L}^{recon} + \mathcal{L}^{reg}$, which means:

$$\mathcal{L} = -\mathbb{E}_{q_\phi(z|x)}\left(\log p_\theta(\mathbf{x}|Z)\right) + D_{\mathrm{KL}}\left(q_\phi(Z|\mathbf{x})\|p(Z)\right)$$

The first term is called reconstruction loss, as the expectation of the log-likelihood of x_n is with respect to the latent Z. Therefore, as it is the negative log-likelihood, small values correspond to a probability inside the log close to 1. Large values correspond to a small probability inside the log, as the - turns large negative values of the log close to 0 positive. So, lower values for that term correspond to a high probability $p_\theta(\mathbf{x}|Z)$, which in turn means that given the latent value Z, the sample x can be reconstructed quite well. Thus, it is desirable to minimize this loss, in order to increase the capabilities to reconstruct inputs.

The second term measures the similarity between the prior and the approximate posterior. To optimize and in this case then minimize the loss, the two distributions have to become similar. Optimizing this term then means that the learned latent distribution is forced to be close to the rather simple prior distribution. Enforcing this constraint on the parameters of the latent distribution leads to a latent mapping that does not differ too much from the previously determined prior structure. Therefore, the degree of overfitting the latent distribution too strongly on the data is limited and therefore this loss term acts as a regularization term.

Question 1.10

The complete loss we want to calculate is of form:

$$\mathcal{L} = \frac{1}{L} \sum_{i=1}^{L} \mathcal{L}_i^{recon} + \mathcal{L}_i^{reg}$$

We begin by looking at the reconstruction loss part. In order to get a reconstruction, at first an observation $\mathbf{x}_n$ needs to be passed through the encoder to obtain a sample $\mathbf{z}_n$ from the latent space to approximate the original input according to the decoder parameters. With passing the observation through the encoder, we can compute:

$$\mu_\phi\left(\mathbf{x}_n\right), \Sigma_\phi\left(\mathbf{x}_n\right)$$

According to these, we can sample $\mathbf{z}_n$ from the approximate posterior $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$ by:

$$\mathbf{z}_n \sim \mathcal{N}(\mu_\phi(\mathbf{x}_n), diag\left(\Sigma_\phi(\mathbf{x}_n)\right))$$

This sample then yields the parameters of the Bernoulli distribution by passing it through the decoder network to compute the mapping from the Neural Network $f_\theta(\mathbf{z}_n)$.

Now, we can look at the reconstruction loss part of the full loss as such:

$$\mathcal{L}_i^{\text{recon}} = -\mathbb{E}_{q_\phi(Z|\mathbf{x}_n)}\left[\log p_\theta\left(\mathbf{x}_n|Z\right)\right] \tag{7}$$
$$= -\log p_\theta\left(\mathbf{x}_n|\mathbf{z}_n\right) \tag{8}$$
$$= -\log \prod_{m=1}^{M} \text{Bern}\left(\mathbf{x}_n^{(m)}|f_\theta\left(\mathbf{z}_n\right)_{(m)}\right) \tag{9}$$
$$= -\log \prod_{m=1}^{M} f_\theta\left(\mathbf{z}_n\right)_{(m)}^{\mathbf{x}_n^{(m)}} + \left(1 - f_\theta\left(\mathbf{z}_n\right)_{(m)}\right)^{(1-\mathbf{x}_n^{(m)})} \tag{10}$$
$$= -\sum_{m=1}^{M} \mathbf{x}_n^{(m)} \log\left(f_\theta\left(\mathbf{z}_n\right)_{(m)}\right) + \left(1 - \mathbf{x}_n^{(m)}\right)\left(1 - \log\left(1 - f_\theta\left(\mathbf{z}_n\right)_{(m)}\right)\right) \tag{11}$$

The expectation dissolves, as I chose to only sample one $\mathbf{z}_n$. If it was for instance L, simply the average over those L samples can be applied to form a MC approximation of the expected value.

The regularization loss is obtained by taking advantage of the previously found closed form of the KL divergence of two distributions (equation (3)). However, as we previously only considered univariate distributions, we need to introduce the clsoed form KL divergence for multivariate distributions. For two multivariate Gaussians, the closed form KL divergence is formualted as such:

$$D_{\text{KL}}\left(\mathcal{N}_0\|\mathcal{N}_1\right) = \frac{1}{2}\left(\text{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + \left(\mu_1 - \mu_0\right)^\top \Sigma_1^{-1}\left(\mu_1 - \mu_0\right) - k + \ln\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\right)$$

K represents the dimensionality of the distribution.
Replacing $\mathcal{N}_1$ with the standard normal, leaves the desired KL divergence $D_{\text{KL}}\left(q_\phi\left(Z|x_n\right)\|p_\theta(Z)\right)$ as such:

$$\mathcal{L}_i^{\text{reg}} = D_{\text{KL}}\left(q_\phi\left(Z|x_n\right)\|p_\theta(Z)\right) \tag{12}$$
$$= \frac{1}{2}\left(\text{tr}\left(\Sigma_\phi(X)\right) + \left(\mu_\phi(X)\right)^\top\left(\mu_\phi(X)\right) - k - \log\det\left(\Sigma_\phi(X)\right)\right) \tag{13}$$

1.5 The reparametrization Trick

Question 1.11
(a)
$_\phi\mathcal{L}$ is needed, as it enables to update the parameters of the encoder network, which depends on ϕ by backpropagating the loss from the final decoder layer back into the encoder. The variational parameters ϕ appear in both the reconstruction and the regularization loss. The former, as the samples to approximate the posterior are drawn according to $q_\phi(z|\mathbf{x}_n)$. The latter more directly, as the approximate posteriors parameters appear directly in the respective loss term. Thus, by calculating $_\phi\mathcal{L}$, the parameters of the encoder can be updated, to learn better latent representations in the compression step.
(b)
When it comes to calculating $_\phi\mathcal{L}$, problems arise, as it incorporates a sampling operation that depends on the distribution $q_\phi(z|\mathbf{x}_n)$, which is parametrized by the encoder. The hinted tutorial mentions that optimization algorithms for Neural Networks, such as Stochastic Gradient Descent, can not cope with discontinuous operations. Discontinuous in this context means, that optimizing an objective that incorporates sampling according to a distribution that has parameters which are to be optimized, is a non differentiable operation.
(c)

The problem of (b) can be overcome by the so called reparametrization trick. This involves the introduction of another distribution that is not parametrized by the encoder to obtain the samples needed for the calculation of z. More precisely, a term ϵ is drawn from a standard normal distribution:

$$\epsilon \sim \mathcal{N}(0, \mathbf{I}_D)$$

That way, z can then be sampled deterministically, as the stochasticity is moved to the noise term ϵ. By multiplying this term to the actual μ_ϕ and Σ_ϕ in that way

$$\mathbf{z} = \mu_\phi + \Sigma_\phi^{\frac{1}{2}} \cdot \epsilon,$$

the required gradients are computable, as this function is deterministic and continous with respect to the parameters of the prior and the approximate posterior of the latent variable.

Question 1.12
The code mainly contains three building blocks. The encoder is built as a two layer MLP, whereas the initial layer is of dimension [784, 500], which is followed by a ReLU non-linear activation layer. 784, as we have 28x28 images, which are flattened to a 28*28 vector. The second layer consists of two separate linear layers of dimension [500, z_dim] to output the mean and log-variance of the approximate posterior. The introduction of the log variance guarantees numerical stability. The decoder then takes one of the latent samples from the encoder to output an image of the original size by passing the latent input through two linear MLP layers with one ReLu activation in between. The first Linear layer is of size [z_dim, 500]. The second is of size [500, 784]. The third building block, the VAE module, brings together the encoder and decoder. This is done by first obtaining mean and log variance from the encoder to then sample one latent z with the help of the reparametrization trick. This latent z is then passed through the decoder to yield both reconstruction and regularization loss, which is then backpropagated through the encoder and decoder for parameter updates.

Question 1.13
The estimated negative lower bound for epoch 1 lies around 170, which decreases quickly after 5 epochs to around 110 and does not change much until the end of the 40 iterations. Both validation and traininig elbo converge around 100, whereas the validation elbo stays slightly above the training elbo. (Figure 1)

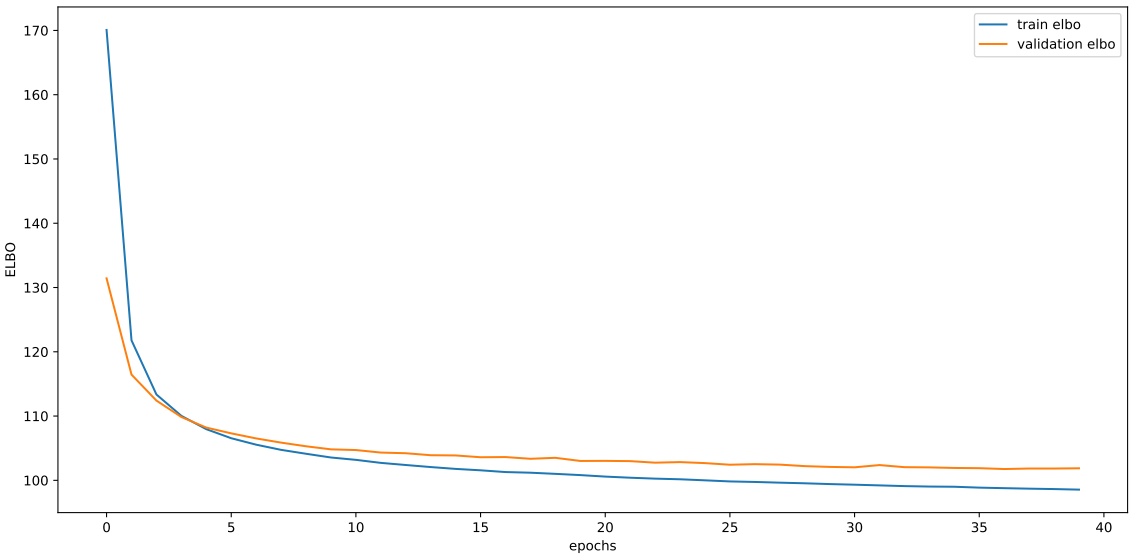

Figure 1: Negative lower bound of both training and validation data.

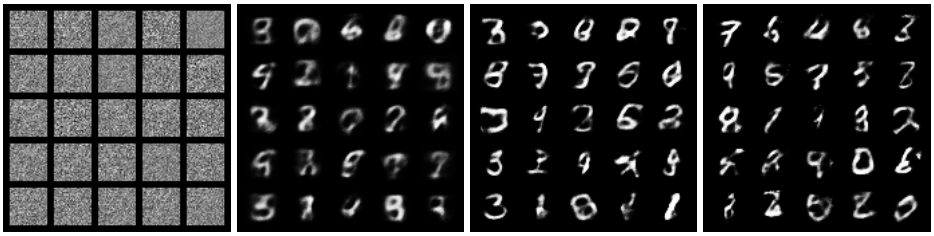

Figure 2: Plots of the means before any training, iteration 1, 25 and 40 respectively.

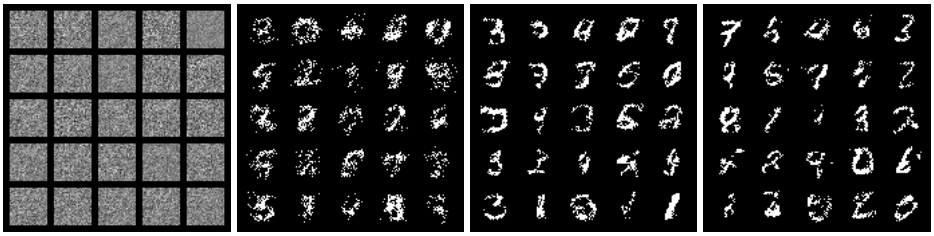

Figure 3: Plots of the samples before any training, iteration 1, 25 and 40 respectively.

Question 1.14

Figure 2 depicts the means of 25 sampled images from the generator before training, after the first, 25th and last epoch. It is clearly visible that the random noise is quickly gotten rid of and over the course of the epochs, the quality improves.

The same can be said when sampling is done the Bernoulli distribution. The images are sharper than the ones sampled with means.

Question 1.15

The data manifold for a 2 dimensional latent space with 20 points for each dimension is depicted in Figure 4.

2 GANs

Question 2.1
The generator takes a noisy sample as input. This noisy sample springs from a prior distribution or some latent space, which can be basically any distribution given there are assumptions on the latent space. This noise input is of the dimensionality of the latent space and is then propagated through the generator to generate an output, that is in the same space as the training data. For every different sample, the Generator results in different outputs. For the case of image classification, the output would be an image. In general, after training, the generator should generate images that reflect the real data distribution of the training data, such that the Discriminator can not, or hardly distinguish between real training data or outputs generated from the Generator.
The Discriminator on the other hand, takes inputs in the space of images. These images can either stem from the training data or the output of the Generator. The goal of the Discriminator then is to detect the generated ones, by assigning them a low probability. This probability corresponds to the output of the Discriminator being one value between zero and one. Where a high probability signals strong confidence of the Discriminator, that the input image is real and low for fake ones.

2.1 Training objective: A Minimax Game

Question 2.2
The first term $\mathbb{E}_{p_{\text{data}}(x)}[\log D(X)]$ describes the log-probability for an observation from the training data to be predicted by the Discriminator D. In other words, its the log-probability for observations

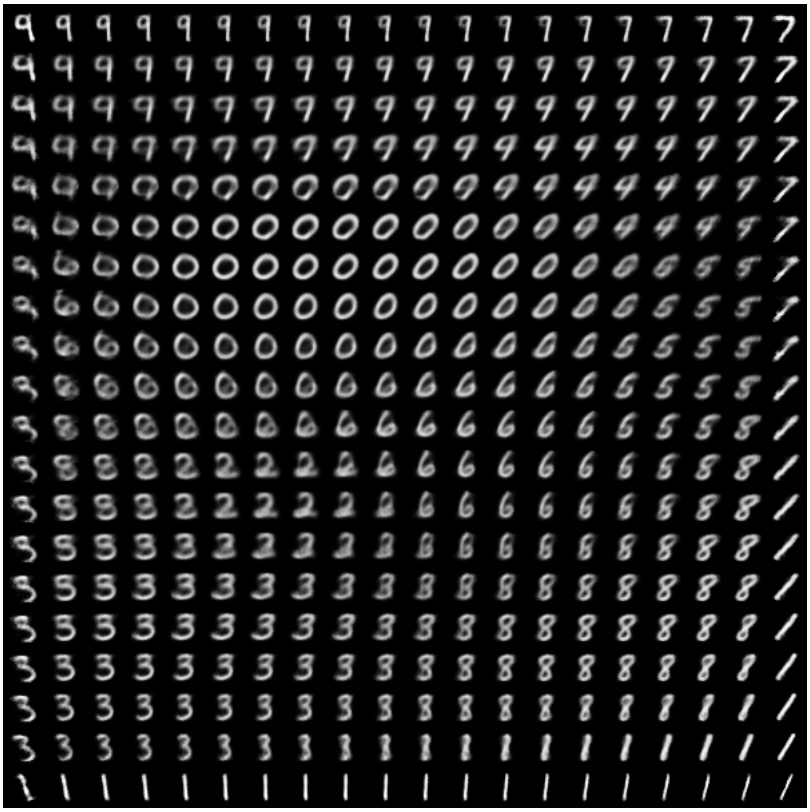

Figure 4: Data manifold for latent space of size 2.

sampled from the actual real data predicted by D. Thus, this term stands for the probability that D gives to an actually real image. By optimizing this term, the Discriminator independently of the Generators parameters learns to detect real images.
$\mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$ on the other hand describes the log-probability the Discriminator assigns to images outputted from the Generator. As $D(G(Z))$ gives the probability for the image to being real, taking 1 - that term reveals the probability of it being fake. The formulation of the optimization makes the Discriminator maximize this probability of assigning low probabilities to generated images for being real. At the same time, the Generator aims to minimize this term, so $D(G(Z))$ gets large, meaning the Discriminator thinks generated images are in fact real.

Question 2.3
After convergence has been reached, the Generator should be able to output images, that can not be distinguished from the real ones by the Discriminator. If that is the case, the Discriminator yields probability a half for both generated and real ones, as it is unable to distinguish real ones from fake ones. For the term $V(D, G)$ this means that both expected values will be $\log \frac{1}{2}$. In total, $V(D, G) = 2 \cdot \log(\frac{1}{2}) = \frac{1}{4}$.

Question 2.4
Early in the training process, the Generator produces images that are not similar to the data distribution, as it relies on random noise for the randomly initialized weights of the Generator Network. That way, the Discriminator can easily detect fake images with high confidence, which is reflected in very low probabilities $D(G(Z))$. These low or possibly close to zero values lead to the right term $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - \mathrm{D}(\mathrm{G}(\mathbf{z}))]]$ to be close to $\log 1$, which means that that part of the loss will be close to zero. As the first term of the total loss is independent of the Generator, all of the Generators learning depends on the second term. As this will be close to zero early on, the gradients will be very small too. Thus, learning the Generators parameters will be not possible or at least not lead to the changes needed to improve the outputs drastically, while at the same time the

Discriminator achieves accuracies close to 1, if the generated images do not increase in quality.

To overcome this issue, the optimization problem for the Generator can be changed from minimization to maximization and respectively changing the term $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - \mathrm{D}(\mathrm{G}(\mathbf{z}))]]$ to $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(\mathrm{D}(\mathrm{G}(\mathbf{z}))]]$. Thus, the new total loss arises as:

$$\max_G \max_D \mathbb{E}_{p_{\text{data}}(\mathrm{x})}[\log D(X)] + \mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})}[\log D(G(Z)]$$

This is equivalent to the original formulation, as the Generator now tries to maximize the probability that the Discriminator detects the generators output as real. This is the same as minimizing the probability for it being fake. This change will still lead to very low probabilities $D(G(Z))$ early on. However, the Generators loss now will be really high and according gradients will too, proper learning is possible.

2.2 Building a GAN

Question 2.5

Regarding the implementation, I pretty much followed the given structure in the code base. Meaning, that the generator consists of 5 linear layers, each of which is followed by a LeakyReLU activation and Batchnormalization. The discriminator consists of 3 linear layers, again with LeakyReLU activations, but no Batchnormalization. As the discriminator classifies binarily, a sigmoid layer is stacked on top. I implemented the Generator loss as the right term of the full GAN loss and take the mean to approximate the expectation. The Discriminator loss uses both terms of the full loss, as it is part of both terms. I did not incorporate measures to improve stability or overcome mode collapse due to time limitations. Other measures for regularization would have been to add dropout. Also, the input data could have been noised to increase the loss of the generator in the beginning, as the Discriminator would not be as sure, as it usually is in early training. Figure 5 depicts 25 samples for 4 stages during training. Once can clearly see that the quality of images increases. However, the final output is noisy and seems to have sampled many 3s which is a sign for mode collapse. This means that the generator will get very good for some classes but fails to do so for others and thus will primarily generate images of that part of the distribution. Another strategy could have been to use another loss function, such as the Wasserstein loss, which seems to be used mostly in recent research.

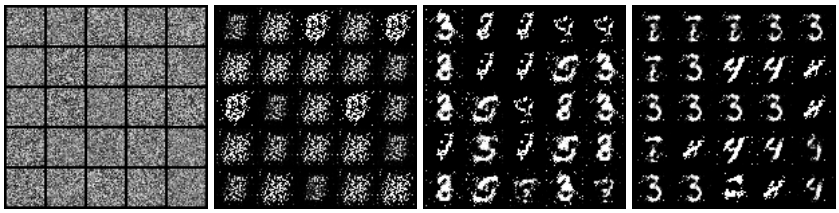

Figure 5: 25 sample images before training, step 2000, step 111000 and step 184000.

3 Normalized Flow

3.1 Normalized Flow

Question 3.1
For the case of an invertible mapping $f : \mathbb{R}^m \to \mathbb{R}^m$ and multivariate random variable $\mathbf{x} \in \mathbb{R}^m$, the distribution $p(\mathbf{x})$ with $\mathbf{z} = f(\mathbf{x})$ the difference to the univariate case lies in the partial derivatives. In fact, the partial derivatives change to the determinant of the Jacobian in the multivariate case. So,

equation (16) of the assignment sheet becomes

$$p(\mathbf{x}) = p(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{x}} \right|$$

The Jacobian on the right side is of dimenion $m \times m$ and contains all pairwise partial derivatives between $f(z)$ and x. Using this change, we also get a rewrite of equation (17) as follows:

$$\log p(\mathbf{x}) = \log p\left(\mathbf{z}\right) + \sum_{l=1}^{L} \log \left| \det \frac{\partial h_l}{\partial h_{l-1}} \right|$$

Question 3.2
The equation is computable, if the determinant of the Jacobian is computable. This is the case, if the Jacobian is square, invertible and differentiable. In order for it to be square, both $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{z}_l \in \mathbb{R}^m$ for all $l = 1, \ldots, L$ have to be in the same dimension. Invertibility ensures that the determinant is non zero and thus log(0) can not appear. Thus, all intermediate partial derivatives need to fulfill invertibility and differentiability.

Question 3.3
The derived equation incorporated the computation of a Jacobian. this operation is computationally expensive and of complexity $\mathcal{O}(n^3)$ (general determinant complexity of a matrix). Additionally, when the inverse f^{-1} is computed, all intermediate h_l also have to be inverted. So, two instances that can cause computational instabilities appear. Strategies to overcome these problems are to use an easily invertible function f, which then leads to a less heavy determinant calculation.

Question 3.4
Using a continuous flow-based model for discrete data will have an effect on the the probability mass of the continuous distribution in the way, that all the mass will lie on the discrete values and as it is continuous, these will then by extremely high. In order to come up with a smoother distribution to model the true data distribution, a technique called dequantizitation comes into play. This basically means adding uniform noise to each pixel to map the discrete values into a continuous interval and after the addition applying the continuous flow model. Alternatives include to parametrize the noise itself, as well to learn the best possible noise added to yield the best optimization possible.

3.2 The coupling-layers of Real NVP

3.3 Building a flow based model

Question 3.6
First, with regards to the input and outputs during and after training: During training, real observations (e.g. images) are the input and the respective log-probability (according to the equation above) of those are the output. During inference time, the input are latent vectors according to a prior distribution. These latent vectors are inversely passed through the flow network. The respective output is generated data according to the latent inputs. The steps involve dequantizizing (is that a word?) the discrete training images by adding noise (e.g. Gaussian or uniform) and then applying logit-normalization. The respective terms are then forwarded to several coupling layers with alternating masks. At each step, the output of a coupling layers is then again forwarded into the next one. At the final transformation, the log-probability and log-determinants of Jacobians are computed. Its negative form is then used as the loss to backpropagate to the model for parameter updates.

4 Conclusion

This project dealt with three different kinds of generative models, namely Variational Autoencoders, General Adversarial Networks and Normalizing Flows. They all differ in their approach. VAEs model an intractable posterior distribution implicitely by variational inference approximation. In it, the estimated lower bound of the data is optimized in order to resemble the input data distribution as closely as possible given the approximated posterior. The training procedure is rather easy compared to GANs, as it only inbolves backpropagating reconstruction loss. An advantage of VAEs is the learned latent representation of the input data, which for instance can be used for pretraining of network architectures. The Normaliziong flow models on the other hand model a data distribution explicitely. This is achieved by taking advantage of serially applying invertible transformations. The training procedure is more complex than the VAEs, but usually the quality of outputs should be higher (not seen in this project due to the lack of the NF implementation). Thirdly, GANs do not model a data distribution explicitely, either The structure is based on two Neural Networks, that try to trick each other and thus based on a joint minmax loss function learn its parameters. In general, GANs yield the highest quality outputs, if stability issues are overcome. As seen in the tasks above, many issues such as mode collapse, early stage training and in general instabilites can occur during training.