Assignment 3. Deep Generative Models

David Biertimpel
david.biertimpel@student.uva.nl
uva-id:12324418

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

Since not provided in the lecture I take a (sligthly adjusted) definition for the Autoencoder from Lilian Weng's blog article 'From Autoencoder to Beta-VAE'.

(1) The standard autoencoder (AE) and the VAE don't have that much in common except that both have an encoder and decoder part. This is evident when looking at their objective functions. The objective of the AE only contains the reconstruction loss between the ground truth image and its reconstruction:

$$L_{AE}(\theta, \phi) = ||X - f_\theta(g_\phi(X))||^2$$

where g is the encoder and f the decoder function. The AE effectively performs dimensionality reduction by learning a latent representation Z of its inputs X. The quality and purpose of this latent representation is directly steered by the reconstruction loss L_{AE}. This leads to the fact that the encoder part of the AE does not model a smooth and coherent distribution of the latent space Z but rather freely optimizes (without constraints) the representation of Z such that the decoder can perform best on its inputs.

In contrast to the AE, the objective function of the VAE does not only consider the reconstruction quality but also the (aimed) prior distribution p_λ of the latent space Z.

$$L_{VAE}(\theta, \phi) = \mathop{\mathbb{E}}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p_\lambda(z))$$

While the first term functions as a reconstruction loss, the second term ensures that the distribution of the latent space resembles the chosen prior distribution. In general we see that the VAE explicitly models the distribution q of the latent space Z instead of representing a plain mapping. This refleczs its purpose of being a generative model.

(2) As already hinted in the last paragraph the AE can not be considered a generative model as it does not learn an explicit joint distribution of the data X and latent space Z from which we could sample. Although, we could construct (by some rules arbitrary rules) latent vectors $\boldsymbol{z}_i$ which we could then pass through the decoder to obtain a generated $\boldsymbol{x}_i$, these most likely don't follow a coherent distribution. So similar vectors in the latent space will not necessarily correspond to similar reconstructions.

1.2 Decoder: The Generative Part of the VAE

Question 1.2

Similar to the graphical model in Figure 1 on the assignment we can represent the given example as a graphical model. Sampling from such a representation can be done with ancestral sampling, where we sample from the random variables in topological order. That means that we start by

sampling from variables with no parents and only sample from a variable when all of its parents have been sampled. In our particular example this means that we first sample a $\boldsymbol{z}_n \sim \mathcal{N}(0, \boldsymbol{I}_D)$ and use this $\boldsymbol{z}_n$ then to sample a $\boldsymbol{x}_n^m \sim Bern(f_\theta(\boldsymbol{z}_n)_m)$ or instead of just a pixel a whole image $\boldsymbol{x}_n \sim p(\boldsymbol{x}_n|\boldsymbol{z}_n) = \prod_{m=1}^{M} Bern(\boldsymbol{x}_n^m|f_\theta(\boldsymbol{z}_n)_m)$.

Question 1.3

We use our decoder function $f_\theta(\cdot)$ to map from our latent space Z to X. This decoder function may be highly non-linear and thus can map a simple distribution like a standard Gaussian to a complex distribution that is more expressive. Therefore, the role of the sampling of $\boldsymbol{z}_n$ is to introduce randomness to the otherwise deterministic decoding. Consequently, the focus is more on a distribution that is easy to sample as is the case with a standard Gaussian.

Question 1.4

(a) Considering the hint and looking at slides 22 – 27 in lecture 9 we conclude:

$$\begin{aligned}\log p(\boldsymbol{x}_n) &= \log \mathop{\mathbb{E}}_{p(\boldsymbol{z}_n)}[p(\boldsymbol{x}_n|\boldsymbol{z}_n)] \\ &= \log \int_{\boldsymbol{z}_n} p(\boldsymbol{x}_n|\boldsymbol{z}_n)p(\boldsymbol{z}_n)d\boldsymbol{z}_n \\ &\approx \log \frac{1}{S}\sum_{i=1}^{S} p(\boldsymbol{x}_n|\boldsymbol{z}_i) \quad , \boldsymbol{z}_i \sim p(\boldsymbol{z})\end{aligned}$$

where S is the size of the sample.

(b) This question relates to the curse of dimensionality as the 'volume' of the sampling space increases exponentially w.r.t the dimensions we add. Therefore, to get meaningful estimates for Z the number of required samples to cover the space grows exponentially to its dimensionality. Given the high dimensionality of latent spaces in many scenarios, the amount of samples we would need to draw during each forward pass of the model, is not feasible. Additionally, when looking at Figure 2 we see $p(\boldsymbol{z}|\boldsymbol{x})$ only taking up a small portion of the space. Therefore, when just sampling from $p(\boldsymbol{z})$ a lot of samples won't fall in the region where $p(\boldsymbol{z}|\boldsymbol{x})$ is high. This leads us to sample a lot of $\boldsymbol{z}_i$ which would most likely not generate our data and therefore result in $p(\boldsymbol{x}_n|\boldsymbol{z}_i)$ being close to zero. Consequently, these small $p(\boldsymbol{x}_n|\boldsymbol{z}_i)$ would not contribute much to our data likelihood and we would need to sample even more to make up for this. This makes it desirable to sample more informed w.r.t. to a given x_n. Since $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ is intractable in our case, we wish to approximate it which leads us to the encoder $q_\phi(\boldsymbol{z}_n|x_n)$ of the VAE.

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

Question 1.5

(a) Since the KL-divergence is *very large* if the two distributions are very different and *very small* (in fact 0) if the two distributions are exactly the same, I decided for the following values:

- *Very large* KL-divergence: $\mu_q = 10^8$, $\sigma_q^2 = 10^{-8}$, $D_{DL} = 5 \times 10^{23}$
- *Very small* KL-divergence: $\mu_q = 0$, $\sigma_q^2 = 1$, $D_{DL} = 0$

(b) We find the closed form forumula for the KL-divergence of two Gaussians (for the multivariate case) in Carl Doersch's tutorial on VAEs [1]:

$$D_{KL}(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)||\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)) = \frac{1}{2}\left(\text{tr}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0\right) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \log\left(\frac{\det\boldsymbol{\Sigma}_1}{\det\boldsymbol{\Sigma}_0}\right)\right)$$

This simplifies for the univariate case to:

$$D_{KL}(\mathcal{N}(\mu_0, \sigma_0^2)||\mathcal{N}(\mu_1, \sigma_1^2)) = \frac{1}{2}\left(\frac{\sigma_0^2}{\sigma_1^2} + \frac{1}{\sigma_1^2}(\mu_1 - \mu_0)^2 - 1 + \log\frac{\sigma_1}{\sigma_0}\right)$$

Question 1.6

We see that the log-likelihood $\log p(\boldsymbol{x})$ is constant as it does not depend on any parameters. Further, we know that $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|x_n)) \geq 0$. From that it directly follows that:

$$\log p(\boldsymbol{x}) = \text{ELBO}_{\theta,\phi}(\boldsymbol{x}) + D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$$
$$\log p(\boldsymbol{x}) + D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n)) = \text{ELBO}_{\theta,\phi}(\boldsymbol{x})$$
$$\implies \log p(\boldsymbol{x}) \geq \text{ELBO}_{\theta,\phi}(\boldsymbol{x})$$

Since $\log p(\boldsymbol{x}) \geq \text{ELBO}_{\theta,\phi}(\boldsymbol{x})$ is always holds, we know when we maximize $\text{ELBO}_{\theta,\phi}(\boldsymbol{x})$ that we simultaneously maximize the data log-likelihood (or in our case minimize $-\text{ELBO}_{\theta,\phi}(\boldsymbol{x})$).

Question 1.7

In theory we could minimize $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ right away instead of taking the detour with calculating the ELBO. However, $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ contains $p(\boldsymbol{z}|\boldsymbol{x}_n)$ which in turn contains $p(\boldsymbol{x}_n) = \int_{\boldsymbol{z}_n} p(\boldsymbol{x}_n|\boldsymbol{z}_n)p(\boldsymbol{z}_n)d\boldsymbol{z}_n$ whose integral is intractable. Therefore, we cannot calculate $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ and need to work with the ELBO.

Question 1.8

The left-hand side of equation 11 on the assignment sheet consists of two parts:

$$\log p(\boldsymbol{x}_n) - D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n)) = \text{ELBO}_{\theta,\phi}(\boldsymbol{x})$$

When we maximize the ELBO the $\log p(\boldsymbol{x}_n)$ may get maximized and $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ may get minimized. Maximizing $\log p(\boldsymbol{x}_n)$ lets the ELBO approach the log-likelihood which leads to a better modelling of the data density and consequently to a better reconstruction. When $D_{KL}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ gets minimized our approximate posterior $q(Z|\boldsymbol{x}_n)$ becomes more and more similar to the desired true posterior $p(Z|\boldsymbol{x}_n)$, which results in the VAE learning a better latent representation Z.

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

Question 1.9

The loss term $\mathcal{L}_n^{\text{recon}}$ resembles the reconstruction loss as it sets the ground truth with the VAE reconstruction into relation. We know that the decoder network $p_\theta(\boldsymbol{x}_n|Z)$ is deterministic. When we pass of a single latent space sample $\boldsymbol{z}_n$ to the encoder, $\boldsymbol{z}_n$ will be deterministically mapped to the reconstruction $\hat{\boldsymbol{x}}$. From equation (4) on the assignment sheet we know that $p_\theta(\boldsymbol{x}_n|Z)$ is Bernoulli distributed. This then leads to:

$$\log p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n) = \sum_{m=1}^{M} \log Bern(\boldsymbol{x}_n^{(m)}|\hat{\boldsymbol{x}}_n^{(m)})$$

where we already see that results in the binary cross-entropy loss, which will be determined by the reconstruction performance.

With the loss term $\mathcal{L}_n^{\text{reg}} = D_{KL}(q_\phi(Z|\boldsymbol{x}_n)||p_\theta(Z))$ we introduce a prior $p_\theta(Z)$ on the latent distribution to the model, so that the approximate posterior $q_\phi(Z|\boldsymbol{x}_n)$ should resemble the prior. This can be understood as regularization since we put a constraint on the encoder, such that $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\Sigma}_\phi$ approach the parameters of $p_\theta(Z)$. Without this term the encoder would freely optimize w.r.t. the reconstruction feedback without any consideration to the latent distribution. This will most likely not result in a coherent and smooth latent space which the prior $p_\theta(Z)$ as a regularizer would enforce.

Question 1.10

In the following we bring together the terms:

$$\mathcal{L}_n^{\text{recon}} = -\mathop{\mathbb{E}}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_n)}[\log p_\theta(\boldsymbol{x}_n|Z)]$$
$$\mathcal{L}_n^{\text{reg}} = D_{KL}(q_\phi(Z|\boldsymbol{x}_n)||p_\theta(Z))$$
$$q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}_\phi(\boldsymbol{x}_n), \text{diag}(\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n)))$$

and use the results from exercise 1.5 (b) to obtain our final objective that we can minimize. The final form should look like this:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(\mathcal{L}_n^{\text{recon}} + \mathcal{L}_n^{\text{reg}})$$

First, we can continue the calculation started in exercise 1.9:

$$\mathcal{L}_n^{\text{recon}} = -\mathop{\mathbb{E}}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_n)}[\log p_\theta(\boldsymbol{x}_n|Z)]$$

(Expectation omitted due to single sample)

$$\begin{aligned}
\implies \log p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n) &= \sum_{m=1}^{M} \log Bern(\boldsymbol{x}_n^{(m)}|\hat{\boldsymbol{x}}_n^{(m)}) \\
&= \sum_{m=1}^{M} \log\left(\hat{\boldsymbol{x}}_n^{(m)\boldsymbol{x}_n^{(m)}}(1-\hat{\boldsymbol{x}}_n^{(m)})^{1-\boldsymbol{x}_n^{(m)}}\right) \\
&= \sum_{m=1}^{M} \boldsymbol{x}_n^{(m)} \log \hat{\boldsymbol{x}}_n^{(m)}(1-\boldsymbol{x}_n^{(m)}) \log(1-\hat{\boldsymbol{x}}_n^{(m)})
\end{aligned}$$

Now, from exercise 1.5 (b) we know:

$$\begin{aligned}
\mathcal{L}_n^{\text{reg}} &= D_{KL}(q_\phi(Z|\boldsymbol{x}_n)||p_\theta(Z)) \\
&= D_{KL}(\mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}_\phi(\boldsymbol{x}_n), \text{diag}(\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n)))||\mathcal{N}(\mathbf{0}, \boldsymbol{I}) \\
&= \frac{1}{2}\Big(\text{tr}\big(\boldsymbol{I}^{-1}\text{diag}(\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n))\big) + (\boldsymbol{\mu}_\phi(\boldsymbol{x}_n)-\mathbf{0})^T\boldsymbol{I}^{-1}(\boldsymbol{\mu}_\phi(\boldsymbol{x}_n)-\mathbf{0}) - D - \log\big(\det\big(\text{diag}(\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n))\big)\big)\Big) \\
&= \frac{1}{2}\Big(\sum_{i=1}^{D}\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n)_i + \boldsymbol{\mu}_\phi(\boldsymbol{x}_n)^T\boldsymbol{\mu}_\phi(\boldsymbol{x}_n) - D - \sum_{i=1}^{D}\log\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n)_i\Big)
\end{aligned}$$

1.5 The Reparametrization Trick

Question 1.11

(a) We need $\nabla_\phi\mathcal{L}$ to obtain the gradient of the Loss w.r.t. the parameters ϕ of the encoder network, such that we can update the encoder via backpropagation. With this we relate the reconstruction performance of the decoder to the latent space mapping of the encoder.

(b) The encoder and the decoder network are *connected* by using the encoder's output $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\Sigma}_\phi$ to draw a latent space sample $\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{x}_n), \text{diag}\big(\boldsymbol{\Sigma}_\phi(\boldsymbol{x}_n)\big))$ which is then passed to the decoder. Since the sampling is a random process it interrupts the differentiable chain of functions and thus does not allow backpropagation. For that reason we cannot pass the gradients of the decoder to the encoder and consequently cannot update the parameters of the encoder.

(c) The reparameterization trick *moves* the random sampling outside the *chain of functions* through which we backpropagate. This is achieved by sampling a random variable $\boldsymbol{\epsilon}$ from a standard Gaussian and subsequently transforming the $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\Sigma}_\phi$ with:

$$\boldsymbol{z}_n = \boldsymbol{\mu}_\phi + \boldsymbol{\epsilon}\cdot\boldsymbol{\Sigma}_\phi, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

This way the random sampling is not blocking the backpropagation anymore as we now can easily calculate $\frac{\partial \boldsymbol{z}_n}{\partial \phi}$ to arrive at the gradient w.r.t. the parameters of the encoder.

1.6 Putting things together: Building a VAE

Question 1.12

See code in Python file `a3_vae_template.py`.

To a large extent I conformed to the given code structure and the descriptions in Kingma & Welling [2], as small hyperparameter changes did not yield better results. The encoder network is a regular MLP that consists of a linear layer with subsequent ReLU activation transforming the flattened input image into an embedding space of size 500. From there two separate linear layers are used to map the embedding to the mean and log variance of the latent distribution respectively (After Kingma & Welling [2] the encoder should model the log variance). The latent space has 20 dimensions. The decoder network is also a regular MLP consisting of two linear layers where the first has a ReLU and the second a Sigmoid activation (hidden dimension of size 500). We use the Sigmoid activation to model the Bernoulli distribution. The average negative ELBO is implemented as described in question 1.10. The model is trained for 40 epochs with Adam as an optimizer with a learning rate of 0.001 and a batch size of 128.

Question 1.13

The estimated training and validation lower-bound of the VAE is shown in Figure 1. We see that both curves steadily decrease and finally reach a validation lower-bound of 111.634 after 40 epochs.

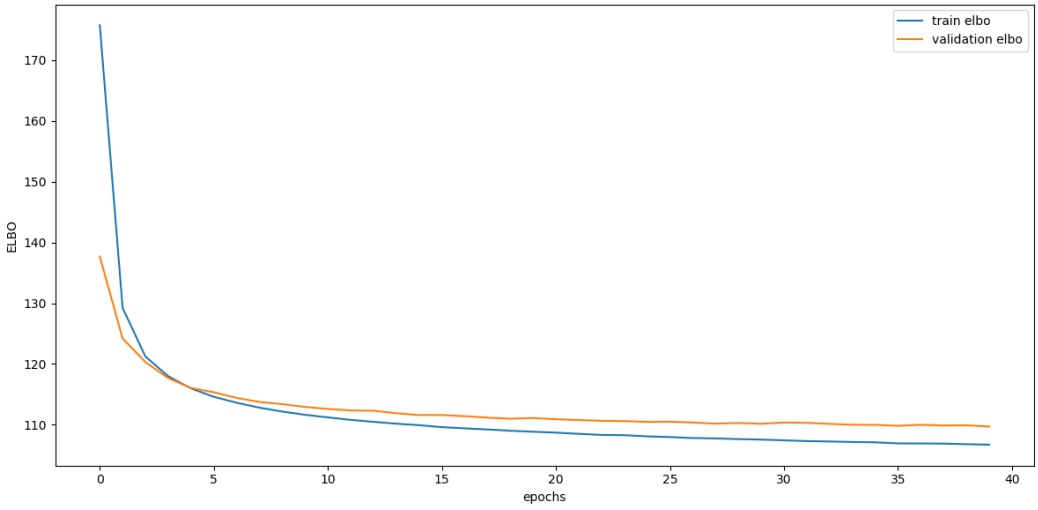


Figure 1: Train and Validation ELBO across epochs with a latent space of size 20.

Question 1.14

In Figure 2 we can observe generated digits at three points of the training. While we see random noise in epoch 0 before any training happened, we already see remotely recognizable digits at epoch 20. Finally, in epoch 40, we see slight improvement compared to iteration 20, but still see noisy or even unrecognizable digits generated. For a person it is not possible to fluently read all the generated digits.

Question 1.15

After training the VAE with a two dimensional latent space, we visualize the learned two dimensional manifold by using the Scipy function `stats.norm.ppf` to get a meaningful coverage of Z and the Pytorch function `make_grid` for the final plot. The corresponding plot can be seen in Figure 3. Although the different digits appear a bit blurry, we can clearly see that the VAE learned a meaningful latent representation of the digits, as similar digits are close together and morph into each other seamlessly. A good example are the transition chains in the bottom rows, where a 7 is morphing into a 9 which then morphs into a 6 which finally morphs into a 0.

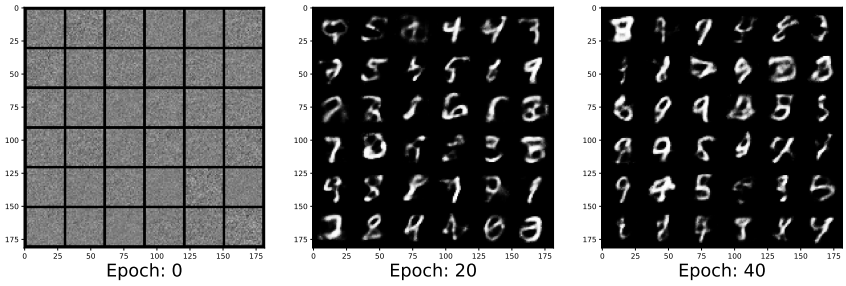


Figure 2: Samples from the VAE from epoch 0, 20 and 40 representing the stages before, during and after training.

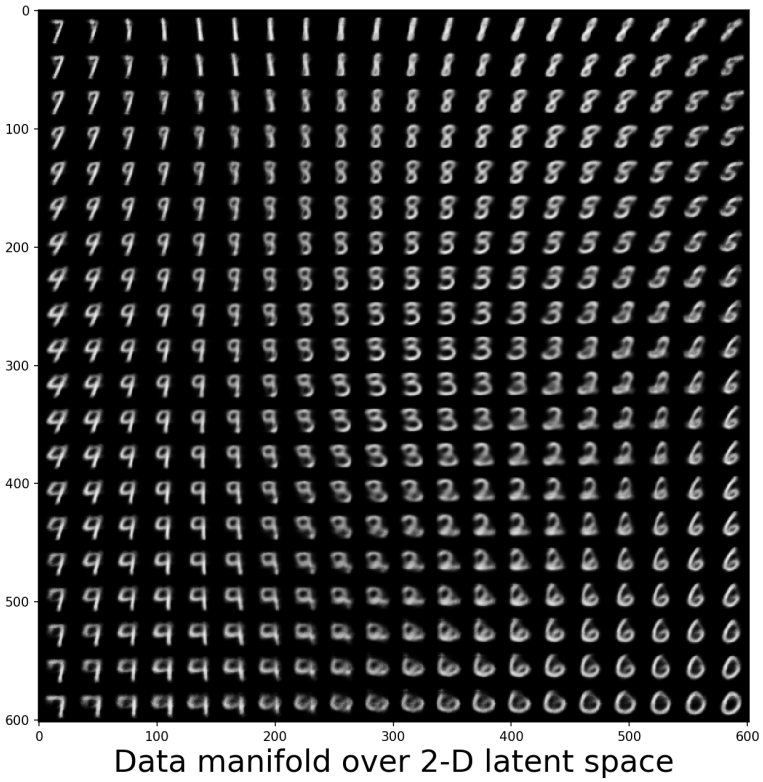


Figure 3: Visualization of the learned manifold of a 2-D latent space.

2 Generative Adversarial Networks

Question 2.1

The generator network G receives a sample from the *input noise* $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}|\boldsymbol{I})$ which it uses to *generate* an instance resembling an observation from the data X. Therefore, the output will have the same dimensionality as samples from X.

The discriminator network D receives an observation from the data X or a generated instance from the generator as input and decides whether the respective input is real (actually comes from the data X) or fake meaning produced by the generator. This decision is represented by a single probability describing the network's confidence that the input is real.

2.1 Training objective: A Minimax Game

Question 2.2

The training objective of the GAN is the following:

$$\min_G \max_D V(D,G) = \min_G \max_D \mathop{\mathbb{E}}_{p_{data}(\boldsymbol{x})}[\log D(X)] + \mathop{\mathbb{E}}_{p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(Z)))]$$

Due to the binary classification task the discriminator's goal is to assign probability 1 to observations coming from the data distribution $p_{data}(\boldsymbol{x})$ and probability 0 to observations coming from the generator $G(\boldsymbol{z}), \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$.

The left term $\mathbb{E}_{p_{data}(\boldsymbol{x})}[\log D(X)]$ is the expected log-probability of the discriminator classifying an observation from the data distribution $p_{data}(\boldsymbol{x})$ as *real*.

The right term $\mathbb{E}_{p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(Z)))]$ is the expected log-probability that the discriminator classifies an observation coming from the generator as *fake*.

Both terms reflect the discriminator's confidence in making the correct classification decision. It is obvious that the generator wants to minimize this objective as this effectively maximizes the probability that the discriminator is mistaken. By contrast, the discriminator wants to maximize this objective as it would result in it distinguishing perfectly between *real* and *generated* observations.

Question 2.3

The optimal behavior of the discriminator is to classify observations produced by the generator (coming from lets say p_G) as *fake* (0 probability) and observations coming from the data distribution p_{data} as *real* (1.0 probability). Given this the optimal discriminator can be described as:

$$D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

By contrast, the optimal behavior the generator can reach is producing observations that are indistinguishable from the ground-truth observations from the data distribution p_{data}. This implies that p_{data} is identical to the distribution p_G learned by the generator.

Combining both perspectives, it is logical that at the equilibrium to which this setup converges, the discriminator can only take random guesses, since observations from p_{data} and p_G will be indistinguishable. This leads to $D^*(\boldsymbol{x}) = 1/2$. If we insert this into our training objective $V(D,G)$ we see:

$$V(D,G) = \log\left(\frac{1}{2}\right) + \log\left(1 - \frac{1}{2}\right) = \log\left(\frac{1}{4}\right) = -\log 4$$

Question 2.4

The task of the generator to generate observations similar to the distribution of p_{data} is much harder than just distinguishing between *real* and *generated* images. Especially, in the beginning of training when the generator outputs mostly noise, the discriminator will be way faster in learning a suitable decision boundary than the generator in learning to generate competitive observations. In this situation

the second loss term containing $1 - D(G(Z))$ is problematic, as when the discriminator classifies almost perfectly, the generator's loss will be small (or even non-existent) leading to slow or no learning at all.

In order to tackle this problem we split the loss of the generator and discriminator such that the generator still gets a strong loss even if the discriminator does a perfect job. The new losses are as follows:

$$J^{(D)} = - \mathop{\mathbb{E}}_{p_{data}(\boldsymbol{x})}[\log D(X)] - \mathop{\mathbb{E}}_{p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(Z)))]$$
$$J^{(G)} = - \mathop{\mathbb{E}}_{p_{\boldsymbol{z}}(\boldsymbol{z})}[\log D(G(Z))]$$

Now the loss of the generator $J^{(G)}$ is large if its generated observations are rejected and small if they are accepted allowing strong gradients and thus better learning.

2.2 Building a GAN

Question 2.5

See code in Python file `a3_gan_template.py`.

For constructing the GAN I closely sticked to the suggested baseline architectures with some added adjustments after experimenting. This means, the generator consists of five layers, with Leaky ReLU activation functions with a negative slope of 0.2, that are preceded by a batch normalization module in layer two, three and four. For the output non-linearity I opted for Tanh as it maps the values in the range of the normalized images. The discriminator network has three layers with the same Leaky ReLU activation functions and a Sigmoid activation in the last layer to map the outputs in the range of binary predictions. To improve performance I included a Dropout module with dropout probability 0.5 as suggested by the 'ganhacks' github that emerged from NeurIPS 2016. Dropout supplements the discriminator with additional regularization preventing it from becoming too powerful too early. Additionally, I split the loss of the generator and discriminator to prevent vanishing gradients of the generator. The GAN is trained over 200 epochs using a batch size of 64. Both generator and discriminator are optimized by using Adam with a learning rate of $2 \cdot 10^{-4}$ (Using Adam is also motivated by ganhacks).

The loss of the generator and discriminator during training can be observed in Figure 4. After small fluctuations in the first few epochs, we see generator and discriminator converging to stable values.

Question 2.6

Figure 5 shows digits generated by the GAN at four points during training. While we see random noise before any training happened, we already see accumulations of pixels in the image center hinting to be numbers after 10 epochs. In epoch 100 we observe recognizable numbers although contours are fuzzy and some digits are not recognizable at all. Finally, after training at epoch 200, we see clearly sharper contours compared to before, but still some noisy and even unrecognizable digits. Compared to the VAE a person will find it way easier to fluently read all the generated digits.

Question 2.7

In Figure 6 interpolations of 5 digit pairs are shown, where each rows describes the interpolation of a single pair. Although, we take the same model that generated the digits in Figure 5d, we do see way worse qualitative results and no stable latent space interpolations. Even after rigorous debugging I was not able to figure out the reason for this. However, transitions in the latent space are remotely observable.

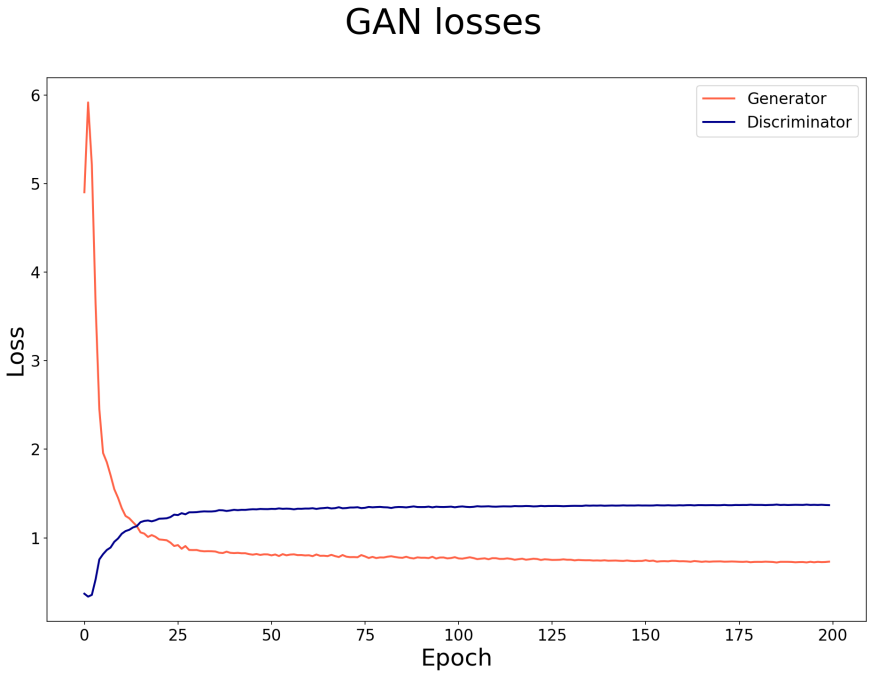


Figure 4: Loss of the generator and discriminator during training.

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

Question 3.1

In order to rewrite equations 16 and 17 for the case of $f : \mathbb{R}^m \to \mathbb{R}^m$ being an invertible and smooth mapping not much needs to change. Equation 16 transfers naturally to this multivariate scenario:

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}); \quad \boldsymbol{x} = \boldsymbol{f}^{-1}(\boldsymbol{z})$$
$$p(\boldsymbol{x}) = p(\boldsymbol{z}) \left| \det \frac{d\boldsymbol{f}}{d\boldsymbol{x}} \right|$$

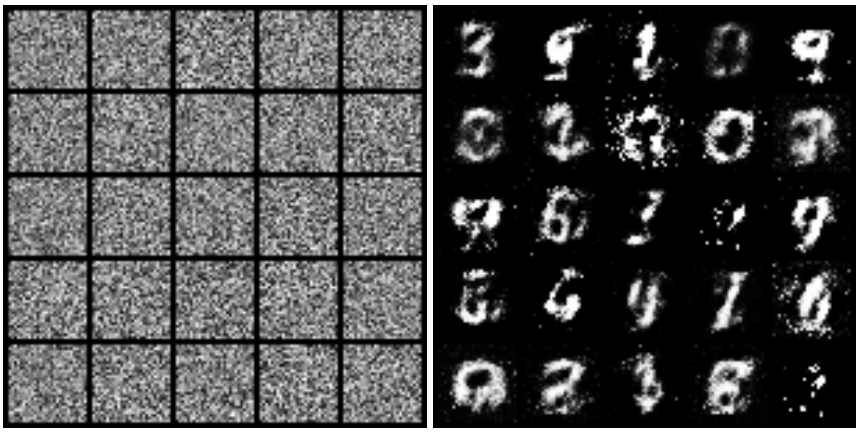
Equation 17 changes as follows:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}) + \sum_{l=1}^{L} \log \left| \det \frac{d\boldsymbol{h}_l}{d\boldsymbol{h}_{l-1}} \right|$$

The major difference is that we now have the *log-determinant* of a *Jacobian*.

Question 3.2

In order for the normalizing flow procedure to work all the functions in the chain need to be invertible. This is the case if the matrix representing this function is square and has a non-zero determinant. The squareness property naturally leads to the constraint that all functions in the chain need to describe the same mapping in terms of their input and output dimensions. This means if $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^m$, each $\boldsymbol{h}_l$ also needs to map from $\mathbb{R}^m$ to $\mathbb{R}^m$, as otherwise, a non-square and thus non-invertible mapping would exist in the chain. Therefore, the latent space needs to have the same dimensionality as the data. Besides the invertibility of the function itself, we want to calculate the determinant of the corresponding Jacobian which would not be square if the dimensions would change. For a non-square matrix, the determinant is not defined.



(a) (b)

(c) (d)

Figure 5: Generated images by the GAN during training. (a) Before training. (b) After 10 epochs and $10,000$ batches. (c) Halfway through training at epoch 100 and after $93,500$ batches. (d) End of training at epoch 200 and after $187,500$ batches).

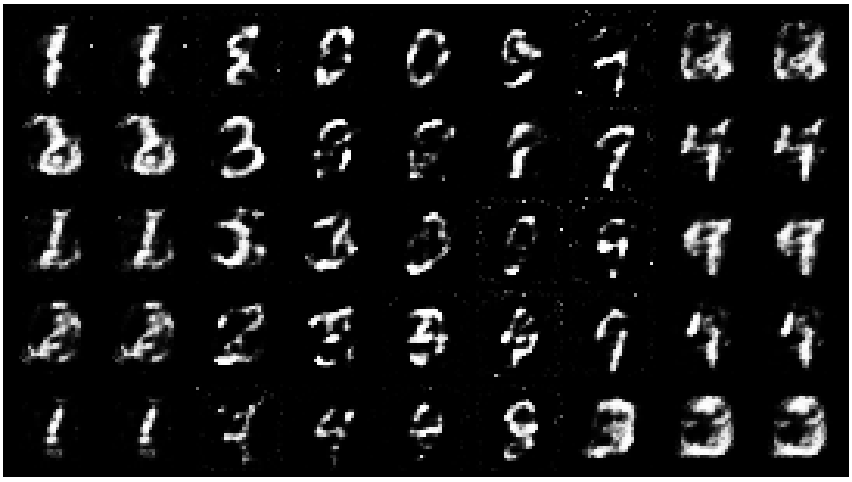


Figure 6: Visualizations of latent space manipulations. In each row we randomly sample two points (left and right borders) and interpolated between them in 7 steps.

Question 3.3

Looking at the equations we derived in exercise 3.1 we see two expensive operations: calculating the determinant of a matrix and inverting a matrix. First, during training, we have to calculate the determinant of the Jacobian $\frac{d\boldsymbol{h}_l}{d\boldsymbol{h}_{l-1}}$ which is (with slight variations depending on the procedure) in $\sim O(m^3)$, if $\boldsymbol{h}_l \in \mathbb{R}^m$. This needs to be computed for each of the L intermediate layers $\boldsymbol{h}_l$.

Later, calculating $\boldsymbol{f}^{-1}$ to map the latent representation back to an observation is also considered expensive. Again, there are different ways to invert a matrix and depending on the computational constants of each method it ranges from $O(m^{2.373})$ to $O(m^3)$.

Because of these points, flow transformations need to satisfy two basic properties: They should be easily invertible and the determinant of the Jacobian should be easy to compute.

Question 3.4

The consequence of having non-continuous random variables as inputs is that the data is then represented as an accumulation of Dirac delta functions. If we fit a "continuous density model" to such discrete data all probability mass will be placed on the discrete data points as described in section 3.1 in Ho et. al. paper [3]. In order to get a smooth, continuous probability density we use *dequantization* to make the originally discrete distribution continuous. One approach is using uniform dequantization, where we simply add uniform noise to the each data point $\boldsymbol{x} \in \mathbb{R}^m$.

$$\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{u}, \quad \boldsymbol{u} \sim [0, 1)^m$$

This leads to $\boldsymbol{x}$ not accumulating around particular values. Other approaches such as variational dequantization also exist.

3.2 The coupling-layers of Real NVP

3.3 Building a flow-based model

Question 3.6

For training a general flow-based model we forward pass the data $\boldsymbol{x} \in X$ as our model input (we assume a single observation for the sake of argument) through all intermediate layers $\boldsymbol{h}_l$ of $\boldsymbol{f}$ until it reaches its latent representation $\boldsymbol{z} \in Z$. From here, we calculate the log probability of our prior distribution p_Z w.r.t. the estimated latent vector $\boldsymbol{z}$ (i.e. $p_Z(\boldsymbol{z}|\boldsymbol{\theta})$) and add it together with the sum of the log determinant Jacobians $\sum_{l=1}^{L} \log \left| \det \frac{d\boldsymbol{h}_l}{d\boldsymbol{h}_{l-1}} \right|$ we collected on the way. This provides us with the log-likelihood estimate $\log p(\boldsymbol{x})$ of the data which is the objective we aim to maximize, by minimizing $-\log p(\boldsymbol{x})$. With backpropagating this loss, we can update the parameters of all $\boldsymbol{h}_l$ by using SGD. After training, we can sample latent vectors $\boldsymbol{z}$ from our prior distribution p_Z and generate new observations $\hat{\boldsymbol{x}}$ by passing $\boldsymbol{z}$ through the inverse function $\boldsymbol{f}^{-1}$.

Question 3.7

See code in the Python file `a3_nf_template.py`. This implementation is based on the *Density estimation using Real NVP* paper from Dinh et. al. [4].

The model is trained over 40 epochs with a batch size of 128 and Adam for optimization with a learning rate of $1 \cdot 10^{-3}$. For the coupling layer I used the suggested architecture as small experiments did not lead to any improvements. This means it consists of three linear layers with ReLU activation for layer one and two. The size of the hidden layers is 1024. The output is split to model the translation and scale output, where the scale output is subsequently transformed by a Tanh non-linearity to increase stability.

Question 3.8

The bits per dimension (bpd) obtained during training can be observed in Figure 7. We see that both train and validation bpd steadily decrease and finally reach a validation bpd of 1.8255 after 40 epochs.

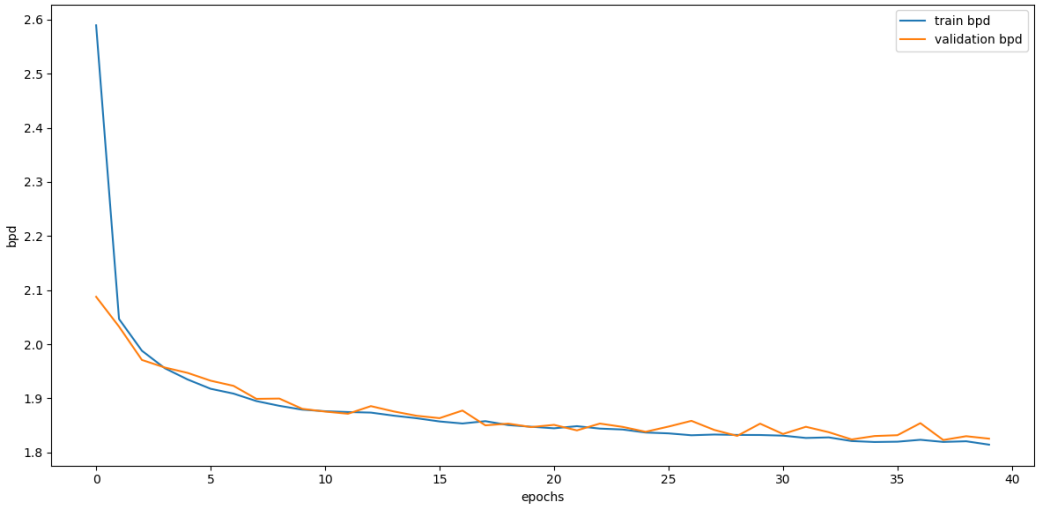


Figure 7: Bits per dimension of the RealNVP model during training.

Further, in Figure 8 we present samples from the model across different epochs. Although we see clear qualitative improvements over the epochs, the flow-based model produces comparably noisy results with fuzzy contours. Nevertheless, when looking the the generated digits in epoch 40 after training most of them are decipherable. The generated digits are not as good as those of the GAN, but compared to the VAE's results a person will find it slightly easier to fluently read the generated digits.

4 Conclusion

Question 4.1

Over the course of this assignment we qualitatively assesses the digits generated by different models by determining whether a person (myself, sample size $N = 1$) could fluently read them. From this it can be concluded that the GAN provides the best quality results, since the generated digits are the sharpest and best readable. Both VAEs and flow-based models produce rather noisy results with generating even some unreadable digits.

VAEs offer a lot of control as we can choose a prior distribution and determine the size of the latent space. Also VAEs allow compression as we can encode an input to the smaller latent space. Further, they are stable to train. VAEs approximate the likelihood by maximizing the ELBO which is a clear objective corresponding to the data distribution and makes model comparisons possible.

GANs do not optimize likelihood as it is optimized by a zero-sum game between generator and discriminator. This makes the GAN unstable during training as generator and discriminator need to maintain a balance. Also it is hard to quantitatively evaluate GANs. Further, due to the missing encoder, no compression is possible.

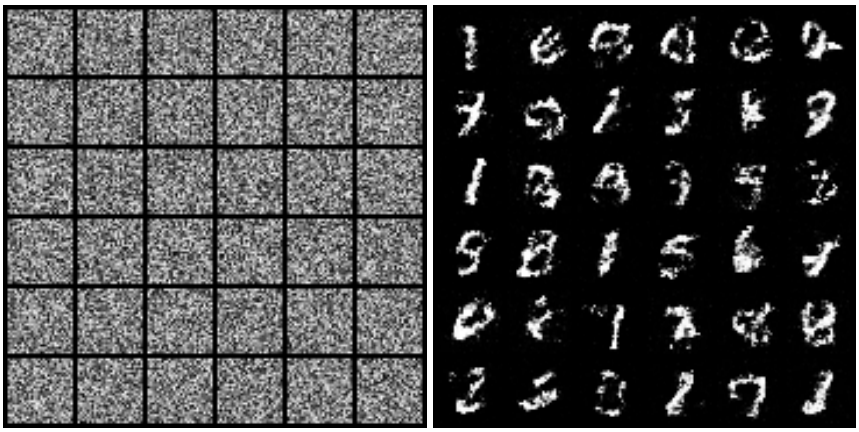
Flow-based model consists of a sequence of invertible transformations which allow it to explicitly learn the distribution of the data. Due to the convertibility constraint it cannot perform compression. In contrasts to GANs and VAEs it consists of only one network (for encoding and decoding) as it can be inverted.

Note: For the sake of readability I included line breaks. When removing them the text boils down to 15 lines.

References

[1] Carl Doersch. Tutorial on variational autoencoders, 2016.

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.



(a) (b)

(c) (d)

Figure 8: Generated images by the RealNVP model during training. (a) Before training. (b) After 10 epochs. (c) Two thirds through training at epoch 30. (d) End of training at epoch 40.

[3] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *CoRR*, abs/1902.00275, 2019.

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.