Assignment 3. Deep Generative Models

Martine Toering
11302925
Deep Learning Course Assignment 3
University of Amsterdam
December 15, 2019
martine.toering@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1 How does the VAE relate to a standard autoencoder? 1. Are they different in terms of their main function or intended purpose? How so? 2. A VAE is generative. Can the same be said of a standard autoencoder? Why or why not?

An *autoencoder* is an encoder-decoder model where the encoder maps the input to a latent representation and the decoder maps it to a reconstruction of the input. The autoencoder attempts to generate a representation that is close to the input. An autoencoder learns a representation unsupervised but there is no mechanism to generate novel examples as the autoencoder does not learn the distribution of the latent variables $p(z)$. The purpose of a autoencoder has mostly been dimensionality reduction as the model can learn a compressed representation (**1**). A *Variational Auto Encoder* (VAE) is a generative model that has the capabilities to generate data by sampling from the latent space. An autoencoder does often not produce good generative samples as the latent space is not regular enough. An autoencoder will (quite logically) overfit severely on the input. Therefore, a standard autoencoder is not exactly generative (**2**). A VAE explicitly attempts to add 'regularization' by encoding input as a probability distribution.

1.2 Decoder: The Generative Part of the VAE

Question 1.2 Describe the procedure to *sample* from such a model. (Hint: ancestral sampling)

To sample from probability distributions such as Bernoulli we use *ancestral sampling* or forward sampling. First we sample an uniform distributed sample or we use the prior if we have one and then we sample based on the new sample. We repeat this process of sampling based on the samples we already have where we repeatedly apply the product rule until we have enough samples:

$p(x_1, x_2, x_3, ..., x_d) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)...p(x_d|x_{d-1}, ..., x_1)$.

Thus, in ancestral sampling we repeatedly sample from distributions of new variables conditioned on already sampled values.

Question 1.3 This model makes a very simplistic assumption about $p(z)$. i.e. It assumes our latent variables follow a standard-normal distribution. Note that there is no trainable parameter in $p(Z)$. Describe why, due to the nature of Equation 4, this is not such a restrictive assumption in practice. (Hint: See Figure 1 and the accompanying explanation in Carl Doersch's tutorial)

A VAE needs to generate something similar to the input x. The graphical model in figure 1 shows the VAE that maps the latent variable z to the input x. A VAE is able to simply *learn* a function that

maps the Gaussian distributed z to the input only by sampling from the latent variable z $p(x|z)$ and by using not any input as seen in figure 1. This is possible because it will sample from $p(x|z)$ which is conditioned on every possible value of z. It will still produce examples that are somewhat similar to the input because of the prior $p(z)$.

Question 1.4 (a) Evaluating $log\ p(x_n) = log\ E_{p_{z_n}}\ [p(x_n|z_n)]$ involves an intractable integral. However, equation 5 hints at a method for approximating it. Write down an expression for approximating $log\ p(x_n)$ by sampling. (Hint: you can use Monte-Carlo Integration).

$log\ p(x_n) = log\ \mathbb{E}_{p_{z_n}}\ [p(x_n|z_n)]$

$log\ p(x_n) = \int p_(z_n)p(x_n|z_n)dz$

$log\ p(x_n) = \frac{1}{N}\sum_{i=1}^{N}\ p(x_n|z_i)$ where $z_i \sim p(z_n)$

Question 1.4 b) Although this approach can be used to approximate $log\ p(x_n)$, it is not used for training VAE type of models, because it is inefficient. In a few sentences, describe why it is inefficient and how does this efficiency scale with the dimensionality of z. (Hint: you may use Figure 2 in your explanation.)

It is possible to use Monte carlo sampling to approximate $log\ p(x_n)$ but in practice, this would be too expensive and slow. The reason for this is that we need a lot of samples for z to even get close to the true distribution as the distribution can be quite large in volume. We can see in figure 2 for example that the samples plotted are clearly not enough samples to capture the distribution.

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

Question 1.5 Assume that q and p in Equation 6, are univariate gaussians: $q = N(\mu_q, \sigma_q^2)$ and $p = N(0, 1)$. (a) Give two examples of (μ_q, σ_q^2): one of which results in a very small, and one of which has a very large, KL-divergence: $D_{KL}(q||p)$.

- (μ_q, σ_q^2) = (0, 2) results in a small KL-divergence. (μ_q, σ_q^2) = (0, 1) results in a KL-divergence of zero which means that the distributions are identical.
- (μ_q, σ_q^2) = (10, 10) results in a very large KL-divergence as this probability distribution q is very different from p.

Question 1.5 b) Find the (closed-form) formula for $D_{KL}(q||p)$.

$D_{KL}(q||p) = \int q(x)\ log\ \frac{p(x)}{q(x)}\ dx$

$= -\int q(x)\ log\ p(x)\ dx + \int q(x)\ log\ q(x)\ dx$

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

When (μ_p, σ_p^2) = (0, 1) this results in

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

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

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

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

Question 1.6 Why is the right-hand-side of Equation 11 called the lower bound on the logprobability?

The right side of the equation is the ELBO and consists of $KL = D_{KL}(q(Z|x_n)||p(Z))$, which is the KL divergence between the approximate posterior $q(Z|x_n)$ and the latent distribution $p(Z)$. It also consists of $E_{q(z|x_n)}\ [log\ p(x_n|Z)]$. Together these terms are the ELBO or the lower bound on the log-probability. If we rewrite the equation we can see that the marginal log-probability consists of $log\ p(x_n) = ELBO + KL$ where KL is here the KL divergence between the approximate posterior and the *true posterior*. We know that this KL divergence is positive, greater than zero. That means that the ELBO is the lower bound of the log-probability as $log\ p(x_n) \geq ELBO$. As the KL divergence between the approximate posterior and the true posterior in the log-probability is difficult to compute, we can instead only optimize the lower bound (ELBO). The ELBO is only equal to the log-probability if the approximation is equal to the true distribution.

Question 1.7 Looking at Equation 11, why must we optimize the lower-bound, instead of optimizing the log-probability directly?

The log-probability is intractable as we would need the true posterior $p(Z|x_n)$ and thus the evidence $p(x)$ that involves an integral. We would need that true posterior in the log-probability for the KL divergence between the approximate posterior and the true posterior as mentioned before. The lower bound does not include this KL divergence term with the posterior and is therefore a lot easier to optimize than the log-probability.

Question 1.8 Now, looking at the two terms on left-hand side of 11: Two things can happen when the lower bound is pushed up. Can you describe what these two things are?

One thing is that we obtain a better latent representation if KL divergence is closer to zero. Therefore, the higher the lower bound, the better the approximate posterior approaches the true posterior and the better the representation is.
As the lower bound is pushed up, the marginal log-probability goes up as well. Therefore, the second thing that can happen is that the log-likelihood increases and we have a better approximation of the likelihood.

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

Question 1.9 can be seen as a reconstruction loss term and an regularization term, respectively. Explain why the names reconstruction and regularization are appropriate for these two losses.

Reconstruction loss is the loss of maximizing the reconstruction accuracy as it is approximating the marginal $p(x)$. It is called *reconstruction* loss as the decoder attempts to learn to reconstruct the input. This can be seen as maximum likelihood. Poor reconstruction of the data means a high reconstruction loss. Regularization loss is the regularization part of the loss and is about minimizing the KL distance between the latent distribution and the approximate posterior. It acts as regularization because we penalize when the samples are too different from the prior.

Question 1.10 Write down expressions (including steps) for as our final objective. Make any approximation explicit.

The regularization loss is given by

$$\mathcal{L}_n^{reg} = D_{KL}(q_\phi\ (Z|x_n)||p_\theta\ (Z))$$

$$\mathcal{L}_n^{reg} = -\frac{1}{2}\Big(log(\sigma_n^2) - \sigma_n^2 - \mu_n^2 + 1\Big)$$

We sum over the latent dimension z:

$$\mathcal{L}_n^{reg} = -\frac{1}{2}\sum_{l=1}^{L}\Big(log(\sigma_n^2) - \sigma_n^2 - \mu_n^2 + 1\Big).$$

The reconstruction loss is given by

$$\mathcal{L}_n^{recon} = \ E_{q_\phi(z|x_n)}\ [log\ p_\theta\ (x_n|Z)]$$

And we again sum over the latent dimension

$$\mathcal{L}_n^{recon} = \sum_{l=1}^{L} \mathbb{E}_{q(z|x_n)}\ [log\ p(x_n|Z^{(n,l)})]$$

We sum over the individual datapoints to get the final objective

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

$$argmin\ \mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\left(-\frac{1}{2}\sum_{l=1}^{L}\Big(log(\sigma_n^2) - \sigma_n^2 - \mu_n^2 + 1\Big) - \sum_{l=1}^{L} \mathbb{E}_{q(z|x_n)}\ [log\ p(x_n|Z^{(n,l)})]\right)$$

1.5 The Reparametrization Trick

Question 1.11 Read and understand Figure 4 from the tutorial by Carl Doersch. In a few sentences each, explain why: (a) we need $\nabla_\phi \mathcal{L}$ (b) the act of sampling prevents us from computing $\nabla_\phi \mathcal{L}$ (c) What the reparametrization trick is, and how it solves this problem.

First, we need the gradient of the loss $\nabla_\phi \mathcal{L}$ in order to perform backpropagation and be able to train the network. Sampling from $q_\phi\ (z|x)$ results in a non-deterministic model / a non-continous distribution. Therefore, the gradient can not be passed through it as the model is not differentiable with respect to the learned parameters. We use the reparamatrization trick to get a differentiable function for z. To make z deterministic we use deterministic μ and σ from the output and from a unit Gaussian noise ϵ that we sample z becomes $z = \mu + \sigma * \epsilon$. Because of this Gaussian noise z is transformed to a random variable.

1.6 Putting things together: Building a VAE

Question 1.12 Build a Variational Autoencoder in PyTorch, and train it on the Binary MNIST data. Start with the template in a3_vae_template.py in the code directory for this assignment. Following standard practice — and for simplicity — you may assume that the number of samples used to approximate the expectation in $\mathcal{L}_n^{recon}$ is 1. Provide a short (no more than ten lines) description of your implementation. You will get full points on this question if your

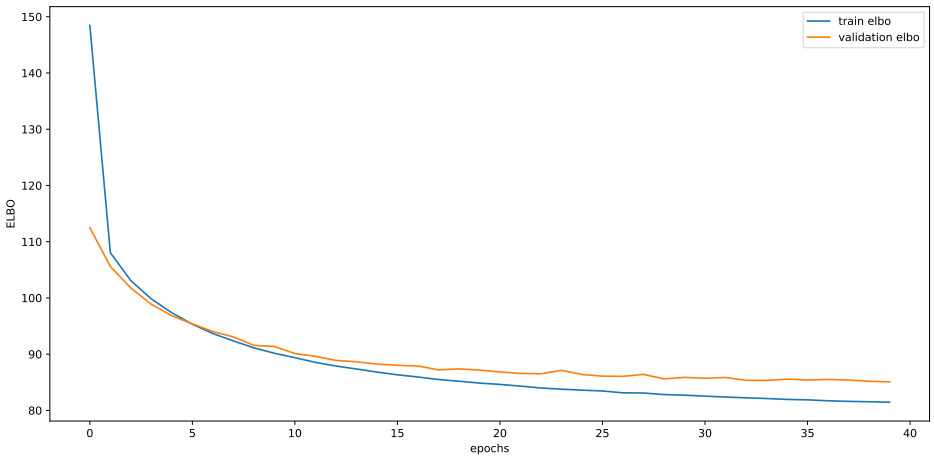

Figure 1: Plot of the curve of the lower bound (ELBO) for the training and validation phases of the Variational Autoencoder

answers to the following questions indicate that you successfully trained the VAE and your description is sufficient, however, do not forget to include your code in your final submission.

The Variational Autoencoder has an encoder part and a decoder part. Like specified in (Kingma and Welling 2013) a linear layer is used that maps to the hidden dimension and a tanh activation function. The encoder outputs mean and std that are implemented as linear layers while the decoder outputs sigmoid non-linearity and returns the means of the sampled images. During training, the VAE model receives the input, performs an encoding step and with the output, mean and std of the distribution, we perform the Reparametrization trick and feed this sample z to the decoder. We compute the ELBO (lower bound) with the formula from question 1.10 and perform backward on this loss function. During training, we can sample images by using the means for the bernoullis, the output of the decoder, to sample from bernoulli. The Variational Autoencoder was trained for 40 epochs with batch size of 128 to learn a 20-dimensional latent space.

Question 1.13 Plot the estimated lower-bounds of your training and validation set as training progresses — using a 20-dimensional latent space. (Hint: Think about what reasonable average elbo values are for a VAE with random initialized weights (i.e., the predicted mean of the output distribution is random) to make sure you compute the correct lower bound over multiple batches)

See figure 1 for the training lower bound and the validation lower bound over 40 epochs of training. We used Adam as optimizer with default learning rate as Adam has been shown to work faster and more reliable than other optimizers (Kingma and Ba 2015) and is in favor for complex problems (Reddi et al. 2018). The figure shows that the lower bound decreases from around 150 to around 80.

Question 1.14 Plot samples from your model at three points throughout training (before training, half way through training, and after training). You should observe an improvement in the quality of samples.

See figure 2 for plots of the sampled images from before training, halfway through training (after the 20th epoch), and after training (after the 40th epoch). The first sampled image shows only noise from the unit gaussian distribution as there is nothing learned yet. The sampled images from halfway through training are recognizable but not of great quality. The sampled images from after training show only marginal improvements in quality over the previous sampled images.

Figure 2: Plot of the sampled images of the Variational Autoencoder before training, halfway through training and after training

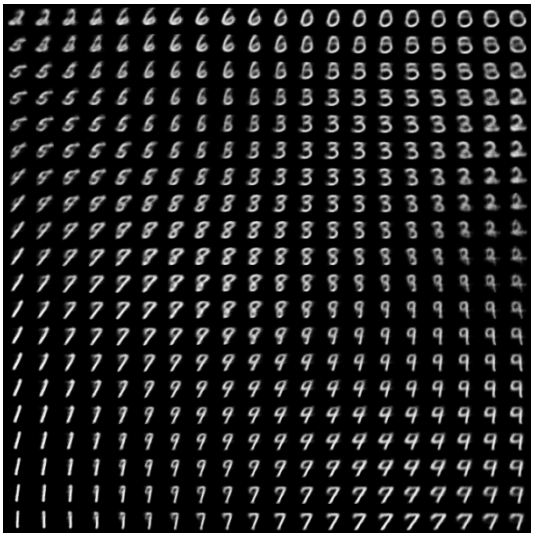

Figure 3: Plot of the manifold of a 2-dimensional latent space of the Variational Autoencoder after training

Question 1.15 Train a VAE with a 2-dimensional latent space (or encoding). Use this VAE to plot the data manifold as is done in Figure 4b of [4]. This is achieved by taking a two dimensional grid of points in Z-space, and plotting $f_\theta(Z) = \mu|Z$. Use the percent point function (ppf, or the inverse CDF) to cover the part of Z-space that has significant density.

See figure 3 for a plot of the manifold. We can recognize the digits 2, 7, 0 and 1 in the corners of the plot. These digits are the digits that the Variational Autoencoder perceives as the most different from each other in the learned latent space.

2 Generative Adversarial Networks

Question 2.1 We can think of the generator and discriminator as functions. Explain the input and output for both. You do not have to take batched inputs into account. Keep your answer succinct.

The input of the generator is random noise z from $p(z)$ and the output are candidates $\hat{x} = G(z)$ which are generated/fake images. The input of the discriminator is the generated data $G(z) = \hat{x}$ OR true data x and the output is the probability of the input being true data $D(G(z)$ or $D(x)$.

2.1 Training objective: A Minimax Game

Question 2.2 Explain the two terms in the GAN training objective defined in Equation 15.

The GAN training objective is $min_g\ max_d\ V(D,G)\ \ E_{p_{data(x)}}\ logD(x) + E_{p_{z(z)}}\ log(1 - D(G(z)))$

This objective resembles the Jenssen-Shannon divergence.

$E_{p_{data(x)}} logD(x)$
This is the loss of real input.
$D(x)$ is the probability of the input being true data as believed by the discriminator. This is what the discriminator tries to maximize.

$E_{p_{z(z)}} log(1 - D(G(z)))$
This is the loss of generated data or 'fake' input.
$D(G(z))$ the probability of the generated data being true data as believed by the discriminator. This is what the generator wants to have a high probability.
$1 - D(G(z))$ the probability of the generated data being false data as believed by the discriminator. This is what the generator thus tries to minimize, and the discriminator tries to maximize.

Again, the generator tries to minimize this training objective $E_{p_{z(z)}} log(1 - D(G(z)))$ and thus get a low probability of the generated data being false as decided by the discriminator. The discriminator tries to maximize the loss of the real input and the negative loss of the fake input and thus discriminate between real and fake input.

Question 2.3 What is the value of $V(D,G)$ after training has converged?

Theoretically, after training has converged, this point is a Nash equilibrium as the distribution of the real data is then equal to the distribution of the generated data. In reality, Generative Adversarial Networks do not always achieve convergence as they have a number of stability problems in training.

Question 2.4 Early on during training, the $log(1 - D(G(z)))$ term can be problematic for training the GAN. Explain why this is the case and how it can be solved.

This term can be problematic if we have a discriminator that is too strong. We get the problem of vanishing gradients as the gradients get too small for the generator. With these small gradients we can not train the generator as it does not provide strong learning signals. It is also possible that the discriminator is too weak. Then the generator is difficult to train as it gets not enough feedback. We can (somewhat) solve the problem of vanishing gradients with $log(1 - D(G(z)))$ by using the *Non saturating heuristic* instead of the Jenson-Shannon divergence:

$J^{(G)} = min_g\ max_d\ V(D,G)\ \ E_{p_{z(z)}}\ log(D(G(z)))$.

When we have a strong discriminator, the generator can now still learn because it maximizes the probability of the discriminator being mistaken.

2.2 Building a GAN

Question 2.5 Build a GAN in PyTorch, and train it on the MNIST data. Start with the template in a3_gan_template.py in the code directory for this assignment. Provide a short (no more than ten lines) description of your implementation. You will get full points on this question if your answers to the following questions indicate that you successfully trained the GAN and your description is sufficient, however, do not forget to include your code in your final submission.

Figure 4: Plot of the sampled images from the Generative Adversarial Network before training, halfway through training and after training

Figure 5: Result of interpolating between two digits (9 and 1) in the latent space of the trained Generative Adversarial Network using 7 interpolation steps

The implementation of the GAN consists of the generator model and the discriminator. First, we train the generator with noise data from a normal distribution; the loss for the generator is the output of the discriminator for the generated data. Next, we train the discriminator where we feed the discriminator both real data and the generated (fake) data; the total loss is the average of both losses. The Generator outputs tanh like in (Radford et al. 2015) while the discriminator outputs sigmoid in range $[0, 1]$ as it discriminates between two choices. The GAN learns by adversarial training a 100-dimensional latent space and we Adam optimizer with a learning rate 0.0002 like in (Radford et al. 2015, batch size of 64 and train for 200 epochs to make it more likely that we iterate a sufficient amount of times through the data.

Question 2.6 Sample 25 images from your trained GAN and include these in your report (see Figure 2a in [2] for an example. Do this at the start of training, halfway through training and after training has terminated.

See figure 4 for images from the GAN at the start of training, halfway through training and after training. The GAN samples images after training that are of better quality than the images from halfway through the training process. There seem to be less anomalies in the pixels. The sampled images appear to be significantly better than the Variational Autoencoder images.

Question 2.7 Sample 2 images from your GAN (make sure that they are of different classes). Interpolate between these two digits in latent space and include the results in your report. Use 7 interpolation steps, resulting in 9 images (including start and end point).

Figure 5 shows the interpolating between two digits in the latent space of GAN using 7 interpolation steps. The interpolation was performed by computing the difference between the start and end image and interpolating by adding a part of this difference for each interpolation step. To perform interpolation in latent space the flag *–interpolate* should be set to *True*.

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

Question 3.1 Rewrite equations 16 and 17 for the case when $f : \mathcal{R}^m \rightarrow \mathcal{R}^m$ is an invertible smooth mapping and $x \in \mathcal{R}^m$ is a multivariate random variable.

$f : \mathcal{R}^m \to \mathcal{R}^m$ and $x \in \mathcal{R}^m$ with distribution $p(x)$:

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

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

Question 3.2 What are the theoretical constraints that have to be set on the components h_l of the function f to make this equation computable? (Hint: think about the number of dimensions in h_l and h_{l-1})

To make this function f computable and invertible, the input and the output (x and z) dimensions must be the same. The distributions must be continuous.

Question 3.3 Even if we ensure the theoretical properties mentioned in the previous question, what might be a computational issue that arises when you optimize your network using the objective you derived in question 3.1? What might be a problem when using your network after training for sampling datapoints?

We need the inverse f^{-1} to be tractable. The jacobian log determinant $\left| det \, \frac{\partial h_l}{\partial h_{l-1}} \right|$ accounts for volume changes in space as a result of the transformation f. This jacobian log determinant must be easy and efficient to compute. For this jacobian log determinant we can therefore use triangular matrices (Dinh et al. 2016) to make it tractable. These requirements are to make sampling fast and efficient. In autoregressive models for instance, sampling is slow. If the log jacobian determinant would not be tractable we would need to resort for example to (slow) Monte carlo sampling.

Question 3.4 The change-of-variables formula assumes continuous random variables. However, images are often stored as discrete integers. What might be the consequence of that and how would you fix it? (Hint: take a look at [3])

Images are often stored as discrete integers, the data is already quantized from continuous signals. Therefore, it is not optimal to fit a continuous distribution to it as this result will be of lower quality. We can use dequantization to transform the images into a continuous distribution again. It is possible to use uniform quantization where we just add noise u from a uniform distribution. We could also find the u with a neural network $q(u|x)$ which is variational quantization (Ho et al. 2019).

3.2 The coupling-layers of Real NVP

3.3 Building a flow-based model

Question 3.6 Within 10 lines, describe the steps you need to take to train a general flow-based model and how to use it after training. In particular, explain the input and output of the model during and after training.

To train a general flow-based model, the input is dequantized and this dequantized input is encoded to a complex function in the latent space with a *flow*. We apply two coupling layers, half of the data stays the same and on the other half scale-and-shift transformations are performed. We split this data with masked convolutions. The scale and shift parameters are output of neural architecture. The log determinant of the jacobian is computed. The prior used in the transformation is a standard Gaussian. After sampling we pass these samples from the forward transformation to the inverted

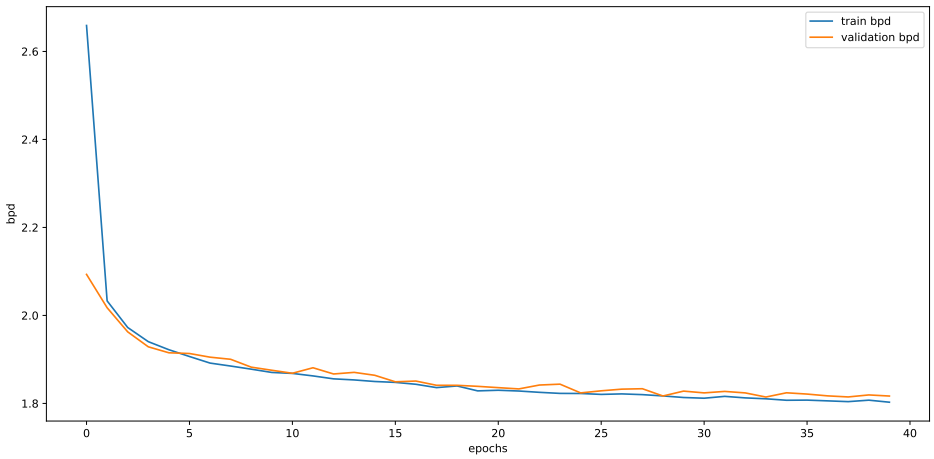

Figure 6: The training and validation loss in bits per dimension for the flow-based model based on RealNVP (Dinh et al. 2016)

function (decoder). We get the negative log likelihood as loss function. The model allows for exact reconstruction of data instead of approximate reconstruction and therefore after training we can directly sample from this distribution. Adam was again used with the learning rate of 0.003 corresponding to the experiments from (Dinh et al. 2016).

Question 3.8 Plot your training and validation performance in bits per dimension (negative log_2 likelihood divided by the size of the image). Your model should be able to reach below 1.85 bits per dimension after 40 epochs.

See figure 6 for the training and validation performance in bpd (bits per dimensions). The model was trained for 40 epochs with a batch size of 128. The loss appears to decrease well and the model reaches below 1.85 bits per dimension around halfway through training.

4 Conclusion

In this assignment, we have seen Variational Autoencoders, Generative Adversarial Networks and Flow-based models as examples of Deep Generative Networks. While Variational Autoencoders approximate the likelihood by optimizing the lower bound, Generative Adversarial Networks learn by adversarial training of two networks with the goal of finding the nash equilibrium. Flow-based models attempt to learn the distribution directly by transforming samples using sequences of invertible transformations. We have seen that Variational Autoencoders are powerful models where we learn an efficient compression in latent space. The results from the implementation of a Generative Adversarial Network however show that the samples from this network are of better quality in comparison to samples from the Variational Autoencoder. As discussed Generative Adversarial Networks are unfortunately associated with a number of problems in stability and convergence which are an active area of research. Lastly, have shown that the implementation of the flow-based RealNVP network functions well. As it exactly models the likelihood in an efficient way, this Deep Generative model is preferred over both the Variational Autoencoder and the Generative Adversarial Network for this task.

References

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). "Density estimation using Real NVP". In: *CoRR* abs/1605.08803.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *CoRR* abs/1902.00275.

Kingma, D. P. and Ba, J. (2015). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980.

Kingma, D. P. and Welling, M. (2013). "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114.

Radford, A., Metz, L., and Chintala, S. (2015). "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434*.

Reddi, S. J., Kale, S., and Kumar, S. (2018). "On the Convergence of Adam and Beyond". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=ryQu7f-RZ`.