Deep Learning Assignment 3. Deep Generative Models

Andrii Skliar
11636785
University of Amsterdam
andrii.skliar@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

Both PCA and PCA and VAEs are techniques that are used for dimensionality reduction. However, VAEs are specifically suited to learn meaningful latent space representation thus hopefully preserving the dimensions with most information while other techniques are just reducing dimensions without actually enforcing meaningful manifold of the new dimensions. This also means that VAEs allow you to sample new data from its latent space and produce new samples while it is not possible neither for PCA nor for FA.

1.2 Decoder: The Generative Part of the VAE

Question 1.2

Sampling in this case consists of three steps:

1. Sample $\mathbf{z}_n$ from $\mathcal{N}(0, \mathbf{I}_D)$
2. Feed sampled $\mathbf{z}_n$ into the decoder, getting $f_\theta(\mathbf{z}_n)$
3. Sample $\mathbf{x}_n^{(m)}$ from $Bern(f_\theta(\mathbf{z}_n)_m)$ for $m = 1 \dots M$

Question 1.3

The reason for this is that $\mathbf{x}_n$ is not directly dependent on $\mathbf{z}_n$ but rather depends on the result of the decoder function f_θ applied to $\mathbf{z}_n$. Therefore, if we train a decoder, that is powerful enough to learn mapping from a sample from a standard-normal distribution to the parameters of a distribution over $\mathbf{x}_n$, this simplistic assumption does not come as an issue anymore.

The problem of non-parametric distribution can be fixed by using amortized inference and reparametrization trick, where we get $\boldsymbol{\mu}$ and from an encoder and use noise sampled from $\mathcal{N}(0, 1)$ to get a sample from the multivariate gaussian with given $\boldsymbol{\mu}$ and .

Question 1.4

(a) $\log p(\mathbf{x}_n) = \log \mathbb{E}_{p(\mathbf{z}_n)}[p(\mathbf{x}_n|\mathbf{z}_n)] = \log \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{x}_n|\mathbf{z}_n^{(m)}), \mathbf{z}_n^{(m)} \sim p(\mathbf{z}_n)$

(b) The main issue is that for most of the sampled $\mathbf{z}_n$, $p(\mathbf{x}_n|\mathbf{z}_n)$ will be infinitesimal or even 0 and won't help during the estimation of the expectation. Therefore, to estimate $\log p(\mathbf{x}_n)$, we need to sample from the regions where $p(\mathbf{x}_n|\mathbf{z}_n)$ has high density. However, as the dimensionality of $\mathbf{z}_n$ increases, ratio of these regions to the regions where $p(\mathbf{z}_n) > 0$ decreases and therefore more samples are needed to estimate the expectation.

1.3 The Encoder: $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.5

(a) KL-divergence has a lower bound of 0 and it reaches it when two distributions match exactly, so $(\mu_q, \sigma_q^2) = (0, 1)$ will result in KL-divergence of 0, while for $(\mu_q, \sigma_q^2) = (1000, \frac{1}{4})$ it will result in ≈ 500000.318.

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

For the case of $p \sim \mathcal{N}(0, 1)$, it becomes: $D_{KL}(q||p) = \log \frac{1}{\sigma_q} + \frac{\sigma_q^2 + \mu_q^2}{2} - \frac{1}{2}$

Question 1.6

Using the fact that KL-divergence is always ≥ 0, we can write following inequality:

$$\log p(\mathbf{x}_n) \geq \log p(\mathbf{x}_n) - D_{KL}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)) = \mathop{\mathbb{E}}_{q(z|\mathbf{x}_n)}[\log p(\mathbf{x}_n|Z)] - D_{KL}(q(Z|\mathbf{x}_n)||p(Z))$$

$$\Rightarrow \log p(\mathbf{x}_n) \geq \mathop{\mathbb{E}}_{q(z|\mathbf{x}_n)}[\log p(\mathbf{x}_n|Z)] - D_{KL}(q(Z|\mathbf{x}_n)||p(Z))$$

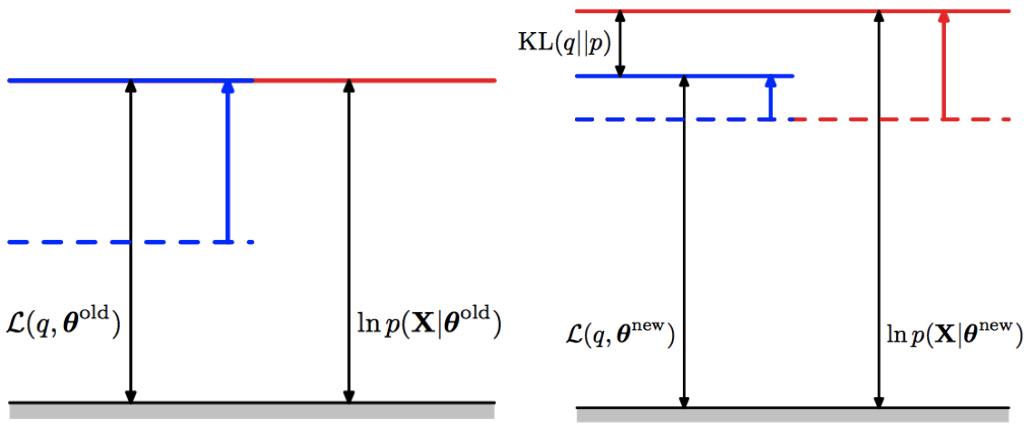
As you can see from the last equation, right hand side will always be less or equal to the log-probability with equality being reached only in case of $D_{KL}(q(Z|\mathbf{x}_n)||p(Z)) = 0$. Therefore, log-probability is lower-bounded by this equation.

Question 1.7

The reasons for that lies in the equation for $\log p(\mathbf{x}_n) = \log \int p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)d\mathbf{z}_n$. This integral is intractable due to the fact that we would need to integrate over all possible states of the latent space. However, both addends in the lower bound on the log-probability can be computed directly and if we maximize the value of this lower-bound, we will automatically maximize the value of log-probability.

Question 1.8

To answer this question we would like to refer to Figures 9.12 and 9.13, page 452 in Bishop [1]. They are describing the question in a visual way and with good comments.



In the first image, we see that the q distribution is set equal to the posterior distribution for the current parameter values θ_{old}, causing the lower bound to move up to the same value as the log likelihood function $\mathcal{L}$, with the KL divergence vanishing.
In the second image, the distribution $q(Z)$ is held fixed and the lower bound $L(q, \theta)$ is maximized with respect to the parameter vector θ to give a revised value θ^{new}. Because the KL divergence is nonnegative, this causes the log likelihood $\log p(\mathbf{x}_n)$ to increase by at least as much as the lower bound does.

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

Question 1.9

- Reconstruction term shows how well encoder and decoder perform. More specifically, it makes sure that the coding learnt by the encoder is good enough so that decoding can produce data, which matching with the original data.
- Regularization term is one of the main part of the VAEs. In makes sure that we can learn more meaningful latent space. This was a famous issue with the classical autoencoders - it was impossible to sample new data from the latent space as it would only be able to produce meaningful results for the latent representation of the datapoints which were already presented to the encoder during training. Therefore, classical autoencoders would not generalize well enough.

  To solve this issue of very sparse latent space, VAE learns a distribution over the latent space by minimizing the regularization term. This results in a much less sparse latent space, sampling from which would still produce meaningful reconstructions.

Question 1.10

Note that in this case we use the fact that all covariance matrices are diagonal, which means that all the dimensions are independent.
Given $\mathcal{D} = \{x_n\}_{n=1}^N$, for each x_n in $\mathcal{D}$, we should do following:

1. Calculate $\mu_\phi(\mathbf{x}_n)$ and $\Sigma_\phi(\mathbf{x}_n)$.
2. Sample $\mathbf{z}_n \sim q_\phi(\mathbf{z}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{z}_n|\mu_\phi(\mathbf{x}_n), diag(\Sigma_\phi(\mathbf{x}_n)))$.
3. Calculate $p_\theta(\mathbf{x}_n|\mathbf{z}_n)$
4. Sample $\mathbf{x}_n \sim p_\theta(\mathbf{x}_n|\mathbf{z}_n)$.
5. Calculate

   $\mathcal{L}_n^{recon} = -\mathbb{E}_{q_\phi(Z|\mathbf{x}_n)}[\log p_\theta(\mathbf{x}_n|Z)] = \log p_\theta(\mathbf{x}_n|\mathbf{z}_n)$, where x_n and z_n have been sampled in the previous steps.

   $\mathcal{L}_n^{reg} = D_{KL}(q_\phi(\mathbf{z}_n|\mathbf{x}_n))||p_\theta(\mathbf{z}_n))$
   $= \frac{1}{2}\sum_{d=1}^{D}\left[-\log\left(\left(\Sigma_\phi^{(d)}(\mathbf{x}_n)\right)^2\right) + \left(\mu_\phi^{(d)}(\mathbf{x}_n)\right)^2 + \left(\Sigma_\phi^{(d)}(\mathbf{x}_n)\right)^2 - 1\right]$

1.5 The Reparametrization Trick

Question 1.11

(a) We need to know $\nabla_\phi \mathcal{L}$ to be able to use backpropagation and perform stochastic gradient descent on the parameters of the encoder.

(b) Act of sampling is done using the Pseudo Random Number Generator, which applies a non-continuous function in order to compute a random sample and therefore its gradient doesn't exist.

(c) Reparametrization trick allows us to utilize the fact that we only need functions applied to the input data to be differentiable but not the data itself. To do that, we add a noise sampled from a specific distribution with parameters generated by the encoding network, which results in z_n having the distribution of the same family as noise. This way we add stochasticity, but all the functions within the model are actually differentiable.

In our specific case with $\mu_\phi(\mathbf{x}_n)$ and $\Sigma_\phi(\mathbf{x}_n)$ coming from the encoder, it would take following form: $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}_D)$; $\mathbf{z}_n = \Sigma_\phi(\mathbf{x}_n) \odot \epsilon + \mu_\phi(\mathbf{x}_n)$, using the fact that all the dimensions are independent.

1.6 Putting things together: Building a VAE

Question 1.12

VAE was implemented in a similar fashion to the original VAE implementation of Kingma et al. [3].

As an encoder, we were using MLP with a single hidden layer of dimension 500, followed by a ReLU nonlinearity and two separate output layers, one for the means and one for the log-variance. We have tried predicting standard deviation instead of log-variance, however, it makes training too unstable and leads to very unsatisfactory results. As a decoder, we were using an MLP with a single hidden layer of dimension 500, followed by a ReLU and an output layer with Sigmoid non-linearity.
As a prior distribution, we were using standard normal distribution with mean of 0 and standard deviation of 1. After that, reparametrization trick was applied to it by mutliplying it by $exp(0.5 * logvar)$, where $logvar$ is the predicted log-variance and adding predicted means. We have used previously mentioned formulas for calculating regularization error (kl-divergence) and reconstruction error (binary cross entropy). They were summed over the batch and then divided in order to get an average ELBO.

Question 1.13

Indeed, **??** confirms the fact that our model learns over time.

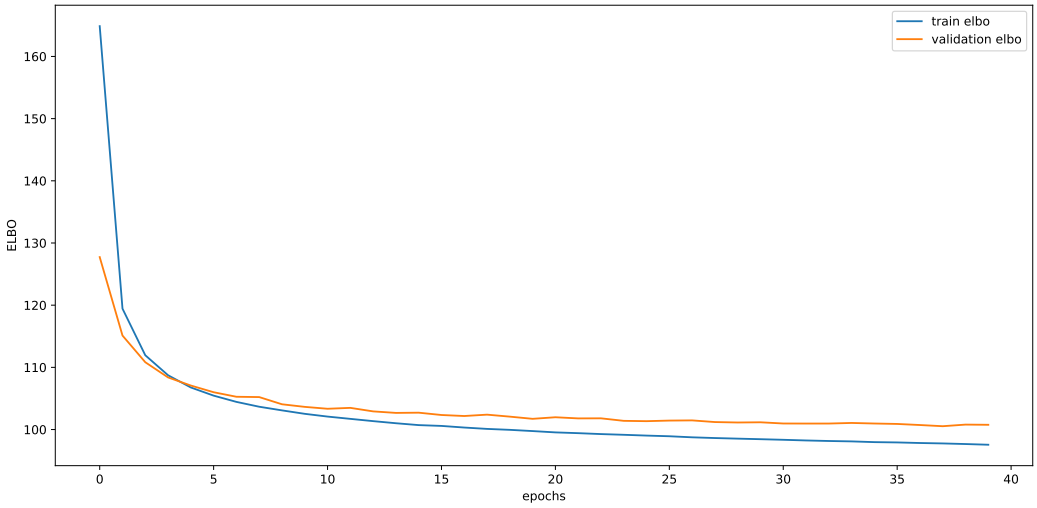


Figure 1: ELBO for 20-dimensional latent space

Question 1.14

We have looked at samples at steps 1, 20 and 40. We have observed that samples clearly improve as training progresses.

However, it looks like for 20-dimensional latent space, VAE takes more epochs to train as samples are less meaningful than ones of 2-dimensional latent space as you can see in fig. 2 and fig. 3.

Also, we have tried both applying bernoulli distribution in the end of the VAE and not applying it. Applying it results in sharper images but with higher amount of noise and improvements are less visible as you can see in fig. 4 and fig. 5.

Question 1.15

We have been using ppf in the range of $[0.0001, 10.0)$ sampling 10 points in this range, which has resulted in the manifold in fig. 6. You can see that the results in the low-density regions (close to ppf of 0.0001) are not producing results, which are meaningful enough. However, as we go over the latent space, we can see how the results change from one digit to another, which means, that the

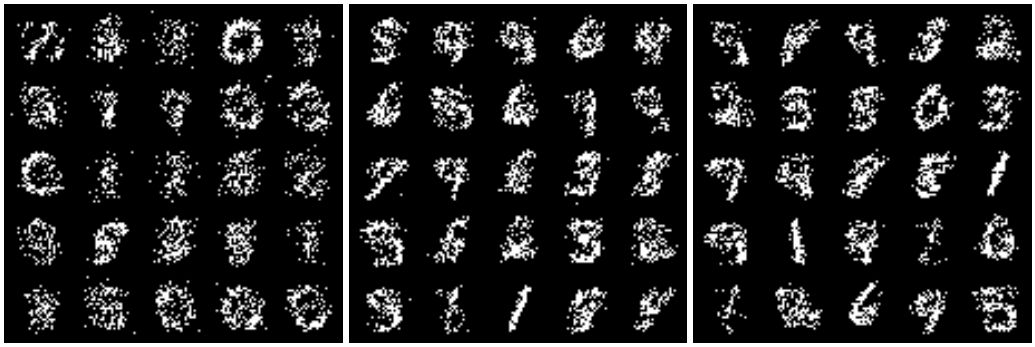


Figure 2: Samples for 2-dimensional latent space at steps 1, 20 and 40 with bernoulli in the end

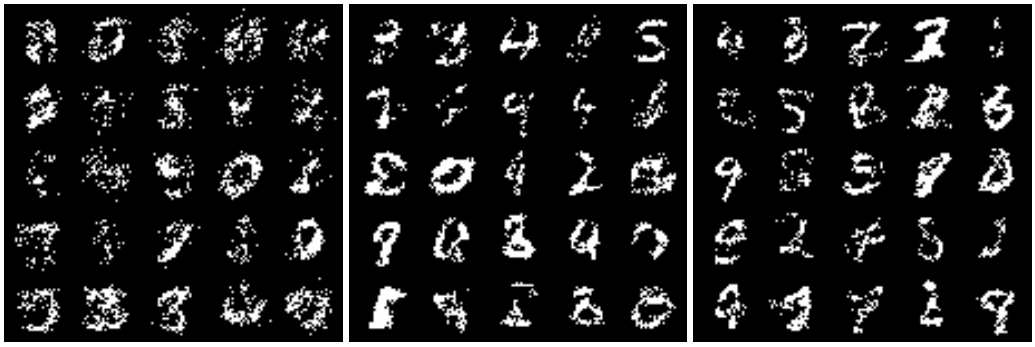


Figure 3: Samples for 20-dimensional latent space at steps 1, 20 and 40 with bernoulli in the end

Figure 4: Samples for 2-dimensional latent space at steps 1, 20 and 40 without bernoulli in the end

Figure 5: Samples for 20-dimensional latent space at steps 1, 20 and 40 without bernoulli in the end

learned manifold is actually meaningful. However, the samples produced are encountering general VAE issue, which is that the results produced are quite blurry.

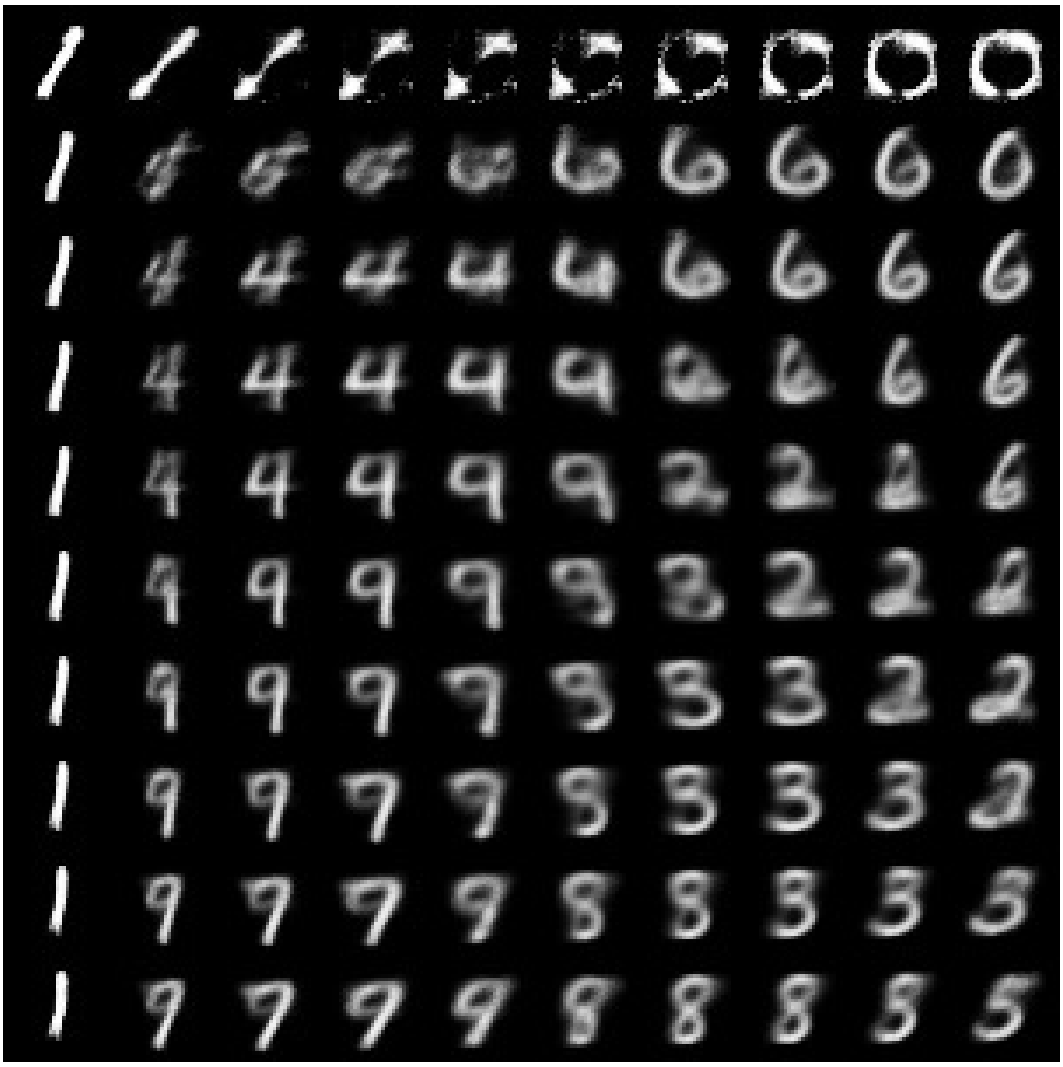


Figure 6: Learned data manifold

2 Generative Adversarial Networks

Question 2.1

- Generator
  - Input: random noise sampled from a previously specified distribution (in our case, $\mathcal{N}(0, 1)$).
  - Output: generated image. More specifically, values of the pixel of the generated image.
- Discriminator
  - Input: image pixel values.
  - Output: probability of the image being real and not fake.

2.1 Training objective: A Minimax Game

Question 2.2

Terms in the GAN objective are doing following:

- $\mathbb{E}_{p_{data}(x)}[\log D(X)]$ is responsible for discriminator correctly classifying real examples as real.
- $\mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$ is responsible for discriminator correctly classifying generated examples as fake.

However, it is also important to specify what $\max_D$ and $\min_G$ are doing. $\max_D$ specifies that the parameters of the discriminator should be updated in a way that would improve the performance of the discriminator thus making it classify generated examples as fake and real examples as real. $\max_G$ specifies that the parameters of the generator should be updated in a way that would make generated examples closer to the real ones. This way we have two adversaries, where generator tries to make images as real as possible and discriminator is trying to be as good as possible at differentiating between real and fake images.

Question 2.3

In case of convergence to Nash equilibrium, generator produces samples, which are indistinguishable from the real ones so discriminator would classify half of the samples as real and half as fake. This would result in $D(G(Z)) = D(X) = \frac{1}{2}$, which results in
$\mathbb{E}_{p_{data}(x)}[\frac{1}{2}] + \mathbb{E}_{p_z(z)}[\log \frac{1}{2}] = \log \frac{1}{2} + \log \frac{1}{2} = 2 \log \frac{1}{2} = -2 \log 2 = -\log 4$

Question 2.4

The main problem with that term is that in the beginning, the discriminator is quite bad and what happens then is that the generator does not get reasonable gradients so it directs generator in a wrong direction which results in bad generator results. A possible solution would be pre-training discriminator before introducing generator into the model.

2.2 Building a GAN

Question 2.5

We have used DCGAN architecture, initially suggested by Radford et al. [4]. The architecture is described in fig. 7. However, different from the original architecture, each convolution layer in both generator and discriminator is followed by a batch normalization layer and LeakyReLU with slope of 0.2. As an input, generator gets noise sampled from a standard gaussian $\mathcal{N}(0, 1)$. Generator outputs images of shape $64x64$ so we had to resize incoming images to the same size in order to use this architecture.
Due to the unstable training of GANs, we have utlizied some of the well-known tips [2]. Namely, we have used a modified loss function for the generator to $\max \log D$ instead of $\min \log(1 - D)$. Also, as suggested in Salimans et al. [5], we have used soft noisy labels for the real data, sampling uniformly between 0.7 and 1.2 and occasionally flipping it to a random value between 0.0 and 0.3 in 5% of the cases. Fake labels were fixed to be 0 in all cases.

Question 2.6

Due to the good performance of DCGANs together with the tricks suggested, we have trained the network only for 40 epochs and have achiveved quite impressive results as you can see in fig. 8. ALso, during training we have noticed that already after first 5 epochs it starts producing quite meaningful results which is very impressive compared to the basic Fully Connected GAN architecture which for us wasn't producing good results at all.

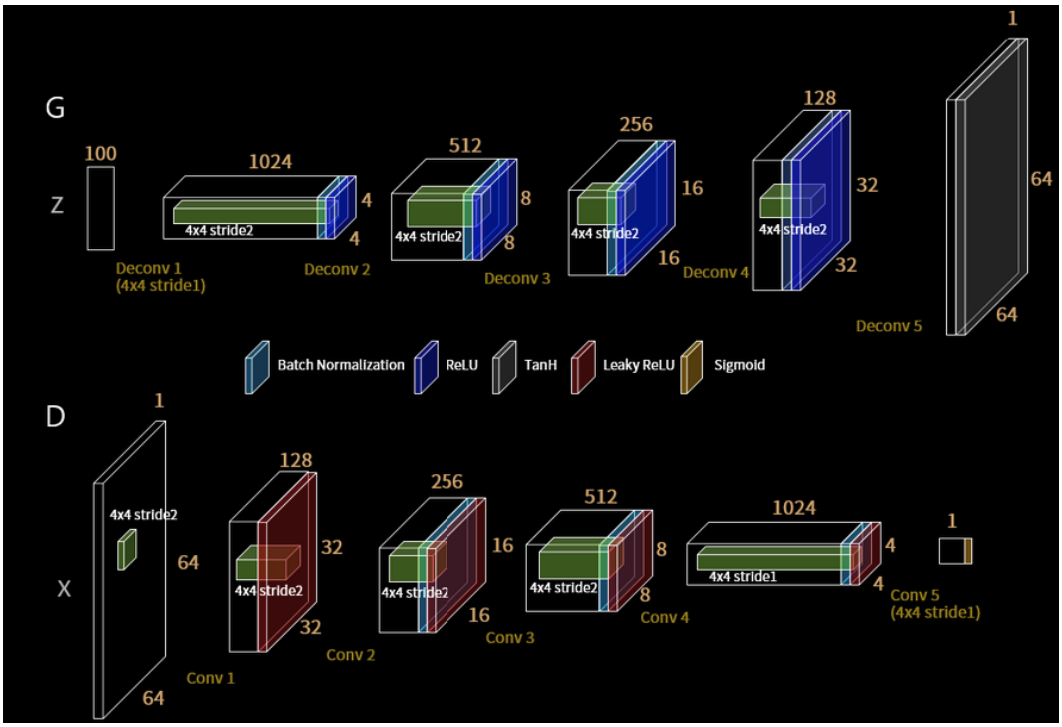


Figure 7: DCGAN Architecture

Question 2.7

It is interesting to observe interpolation between two points in fig. 9. We can see how number 6 gradually changes into 4, which means, that GAN has learned a meaningful space from which we can sample even though it has been only defined implicitly.

3 Conclusion

Question 3.1

VAEs and GANs both belong to the class of generative models. However, while VAEs put their main point as to learn a meaningful latent space representation, GANs specifically try to learn how to generate better samples. This results in that GANs can't be properly evaluated while for VAEs we can directly calculate log-likelihood as a metric.

As we have observed during this assignment, VAEs are somewhat more complicated to create as we need to find a proper reparametrization to make reparametrization trick work, calculate proper KL-divergence formula while GANs seem to be just plug-and-play models.

GANs also tend to be zero-forcing while VAEs tend to be zero-avoiding models. It results in GANs being able to fit data with multiple modes better thus making examples generated more crisp, while VAEs tend to generate more blurry images , learning high-variance latent space. However, due to this property, GANs also have famous problem of mode collapse which often makes them very hard to train.

So, in the end, GANs perform better during generation but are hard to train while VAEs perform worse, but are much more stable and actually learn explicit latent space which seems like a reasonable tradeoff to make before choosing a model for your purposes.

Figure 8: Samples for DCGAN at steps 1, 5, 20 and 40

Figure 9: Interpolation between two points in the space

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Soumith Chintala. How to train a gan? tips and tricks to make $gans$ work, 2016.

[3] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[5] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.