Assignment 3: Deep Generative Models

Volodymyr Medentsiy
id: 12179078
volodymyr.medentsiy@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1
1) The autoencoder's objective is to minimize reconstruction loss (the dissimilarity between original datapoints and reconstructed):

$$Dissimilarity(X, Decoder_{\theta 1}(Encoder_{\theta 2}(X))) \rightarrow \underset{\theta 1, \theta 2}{minimize}$$

L2 or L1 distance could be considered as dissimilarity measure. Whereas VAE's objective is to maximize the log-loglikelihood of the data while minimizing the distance (defined as KL divergence) between true posterior of the data and approximate posterior:

$$log(p_\theta(X)) - D_{KL}(q_\phi(Z|X))||p_\theta(Z|X)) \rightarrow \underset{\theta, \phi}{maximize},$$

which corresponds to maximizing the variational lower bound of the log-likelihood.
2) The standard autoencoder(AE) is not generative, because by definition the model is generative when it models the probability distribution of the data (explicitly, like VAE, or implicitly like GAN). The goal of AE is to learn encoder which maps from the space of data D to another lower dimensional space H, so compresses the data, and decoder, which maps from the space H to D ,decompresses the data, so no probability distributions are modeled. In VAE encoder is used to model the distribution $p(z|x)$ and decoder to model $p(x|z)$, so that given prior distribution $p(z)$, we can construct $p(x) = \sum_z p(x|z)p(z)$.
3) I think that in principle VAE could be used as standard AE but it won't be optimal. For example to obtain the lower dimensional representation of the data, we can pass the data to the encoder which will produce the parameters of the distribution of latent variables and take treat this parameters as compressed representation (code). To reconstruct the data we could either pass mean or mode of the obtained distribution to the decoder or pass sample from the distribution with corresponding parameters, and the output treat as decompressed data. However, Chen and Abbeel [2016] argue that VAE could generate uninformative latent codes, which would be compensated by powerful decoder, so in such a case the encoder is useless to compress the data. The same paper propose solution to such a problem by introducing Lossy VAE (VLAE). VLAE learns code which is useful to reconstruct global structure of the data, but fails to encode local structures of the data.
4) In VAE we consider latent variables z with distribution over them $p(z)$, and conditional distribution $p(x|z)$, which combined will be used to generate new data, $p(x) = \sum_z p(x|z)p(z)$.

1.2 Decoder: The Generative Part of the VAE

Question 1.2
To sample from directed graphical models we can use ancestral sampling, the general idea of this approach is to sample from the first node and then sample consecutively from conditional distribution given observed. In VAE it results in the following procedure: first sample $\tilde{z}$ from $p(z)$, pass it to the decoder, obtain parameters θ, and then sample from $p_\theta(x|\tilde{z})$.

Submitted to the Spring 2019 Amsterdam Deep Learning course

Question 1.3
Initially we sample form N(0, 1), but it does not harm the performance of VAE, because we can model any distribution by taking the normally distributed variables and mapping them through some function. So we can obtain complex distribution by passing samples from N(0, 1) through decoder, which will transform it to samples from any kind of a distribution needed for our model.
Question 1.4
A

$$log(p(X)) = \sum_{n=1}^{N} log(p(x_n)) = \sum_{n=1}^{N} log(\int p(x_n|z_n)p(z_n)dz_n) = \sum_{n=1}^{N} log(E_{p(z_n)}p(x_n|z_n))$$
$$\approx \sum_{n=1}^{N} log\frac{1}{L}\sum_{l=1}^{L} p(x_n|z_n^l) \quad \text{with} \quad z_n^l \sim p(z_n)$$

Where L is defines how many times we will sample to obtain approximation of $E_{p(z_n)}p(x_n|z_n)$.
B
When we use MC method to estimate the expectation, we will sample z from $p(z)$. However the $p(z)$ is concentrated in the region of low density of $p(x|z)$, so it will not contribute much to estimate the expectation. This will only become worse with the increase of z-s dimensionality, because then the region(volume) from which most z will be sampled become even smaller relatively to the region(volume) of high density of $p(x|z)$. Thus the bigger dimensionality of z the more samples we need to get good estimate of expectation.

1.3 The Encoder

Question 1.5
A D_{KL} is minimized when distributions coincide. So the smallest D_{KL} is guaranteed for $\mu_q = 1$ and $\sigma_q^2 = 1$. D_{KL} will be maximized when either $\mu \to \infty$ or $\sigma \to \infty$ or $\sigma \to 0$. So for example for $\mu = 10^6$ and $\sigma_q^2 = 10^{-6}$ it will take very large value.
B

$$D_{KL}(N(\mu_q; \sigma_q^2), N(0; 1)) = -\frac{1}{2} + log\frac{1}{\sigma_q} + \frac{\sigma_q^2 + \mu_q^2}{2}$$

Question 1.6
Considering that $D_{KL}(q||p)$ is always non-negative:

$$log(p(x_n)) \geq log(p(x_n)) - D_{KL}(q(Z|x_n))||p(Z|x_n)) = E_{q(z|x_n)}(log(p(x_n|Z))) - D_{KL}(q(Z|x_n)||p(Z))$$

So the right-hand-side is called lower bound because it gives the lower estimate of $log(p(x_n))$.
Question 1.7
The $log(p(x)) = \int_z p(x|z)p(z)dz$ could take exponential time to compute and thus is intractable. So instead we consider the lower estimate of this quantity, ELBO, which is easier to compute, and optimize it instead.
Question 1.8
When ELBO is increasing two things may happen:

- $log(p(x_n))$ increase, which corresponds to the situation when our model better fits the data.
- $D_{KL}(q(Z|x_n)||p(Z|x_n))$ decrease, which means that $q(Z|x_n)$ better approximates posterior $p(Z|x_n)$.

1.4 Specifying the encoder

Question 1.9
In general, reconstruction loss penalizes for creating bad reconstructions. Reconstruction loss pushes both encoder to produce good latent representations Z, and decoder to produce proper reconstruction of input images given Z. The regularization loss pushes approximate posterior $q(Z|x_n)$ to be close to the prior $p(Z)$.

Question 1.10

$$L(\theta, \phi) = \frac{1}{N} \sum_{n=1}^{N} (L_n^{reg}(\theta, \phi, x_n) + L_n^{rec}(\theta, \phi, x_n)).$$

$L_n^{reg}(\theta, \phi, x_n)$ could be computed with formula derived in 1.5, to obtain the estimation $L_n^{rec}(\theta, \phi, x_n)$ we can use Monte Carlo method. So the each individual losses could be computed as (assuming $z \in R^K$):

$$L_n^{reg}(\theta, \phi, x_n) = D_{KL}(q_\phi(Z|x_n)||p_\theta(Z)) = D_{KL}(N(Z|\mu_\phi(x_n); diag(\Sigma_\phi(x_n))), N(0; I))$$
$$= \sum_{k=1}^{K} \frac{(\sigma_\phi^{(k)}(x_n))^2 + (\mu_\phi^{(k)}(x_n))^2 - 1}{2} + log(\sigma_\phi^{(k)}(x_n))$$

.

$$L_n^{rec}(\theta, \phi, x_n) \simeq \frac{1}{L} \sum_{l=1}^{L} logp_\theta(x_n^{(l)}|Z^{(l)}) \quad \text{with} \quad Z^{(l)} \sim q_\phi(z_n|x_n),$$

where L defines number of times we need to sample to get the estimate of $L_n^{rec}(\theta, \phi, x_n)$. The original paper suggests using reparametrization trick, which will result in sampling from $N(0, I)$ and then applying transformation to get the sample from $q_\phi(z_n|x_n)$.

1.5 The Reparametrization Trick

Question 1.11
A The objective of VAE is to minimize the loss with respect to θ and ϕ. So we need $\nabla_\phi L$, to apply gradient descent algorithm.
B When we sample from q_ϕ directly, we can not apply gradient descent algorithm to optimize for parameters ϕ, because ϕ is parameter of the distribution and we observe only samples from this distrubution and not its parameter explicitly. To overcome this and use backpropagation algorithm for ϕ we apply reparametrization trick.
C The reparametrization trick Kingma [2013] proposes another way to sample from $q_\phi(Z|x_n)$, which will allow to use backprop during training. In general case, when approximate posterior and prior are not Gaussians, instead of sampling from $q_\phi(Z|x_n)$ directly we may sample z from distribution $p(\epsilon)$, which has no learnable parameters, and then apply transformation $g_\phi(\epsilon, x_n)$ to obtain sample from $q_\phi(Z|x_n)$. In case of Gaussian random variables this will result in the following procedure: first sample z from $N(0, I)$ and then multiply z by $diag(\sigma_\phi(x_n))$ and add $\mu_\phi(x_n)$. So now $diag(\sigma_\phi(x_n)) * z + \mu_\phi(x_n)$ will come from $q_\phi(Z|x_n)$ and we can take gradients with respect to ϕ to backpropagate.

1.6 Putting things together: Building a VAE

Question 1.12
The input to the encoder is a batch of vectors each of size 784; the output is a batch of parameters (mean and standard deviation of Normal distribution) of the approximate posterior distribution of 20-dimensional latent space. The encoder of VAE is modeled with a 2-layer NN. The 500-dimensional hidden layer followed by tanh activation is shared for the mean and standard deviation. The activations of a 2-nd layer are an identity for mean, and softplus for standard deviation because standard deviation should be a non-negative quantity. The input to the decoder is a batch of samples from approximate posterior; the output are parameters of 784-dimensional multivariate Bernoulli distribution. To make backpropagation possible, sampling is implemented with a reparametrization trick. The decoder is modeled with a 2-layer NN. The first layer has 500 dimensions and tanh activation. The next layer has sigmoid activation function to guarantee that output corresponds to the probability.
ELBO is optimized with Adam algorithm with default parameters.
Question 1.13
The visualisation ELBO on training and validation data for the model is given in 1.

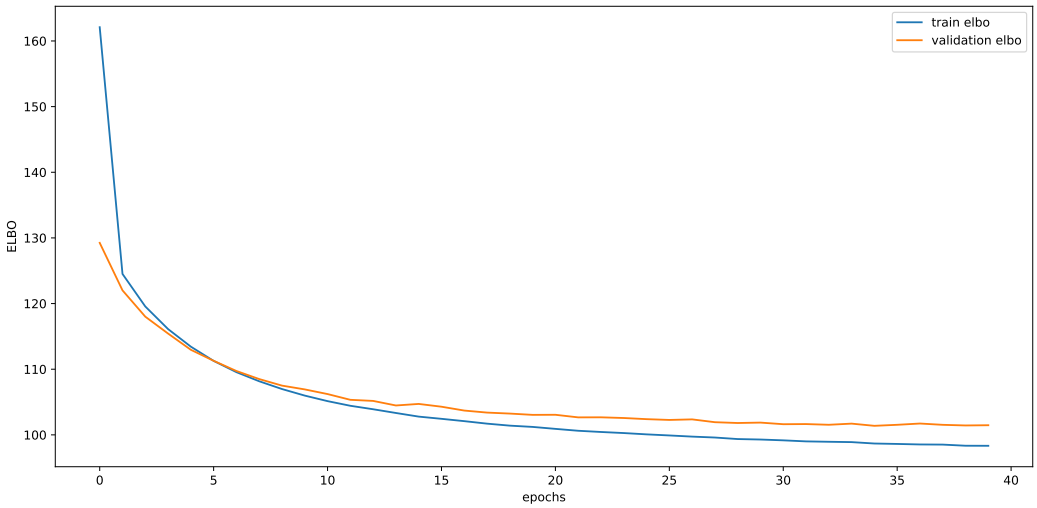
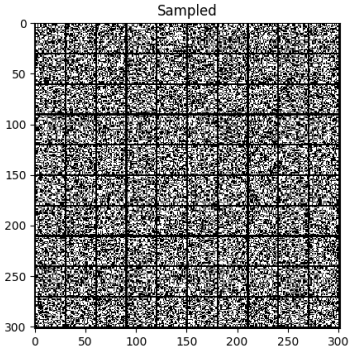
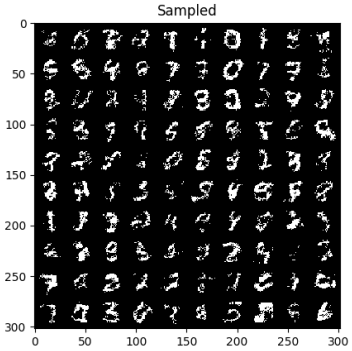


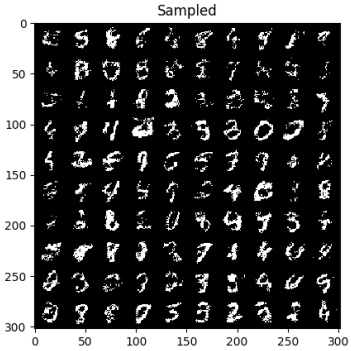
Figure 1: ELBO on training and validation data for VAE with 20-dimensional latent space



((a)) Before training



((b)) Half way through training



((c)) After training

Figure 2: Images sampled from VAE

Question 1.14
Images sampled from VAE at various stage of training are visualised in figure 2. Though the quality of images improves they are still quite noisy and not very realistic.

Question 1.15
The output of VAE's decoder (parameters of $p_\theta(X|Z)$, which corresponds to mean of the multivariate Bernoulli) in the space of 2 dimensional latent variables is visualized in 3. The images are grouped together by digits and we can observe how changes in latent variables correspond to changes in images.

2 Generative Adversarial Networks

Question 2.1
The input to the generator is z - samples from the prior distribution of latent variables, e.g. multivariate Gaussian. The output of generator is the sample from the data distribution, e.g. multivariate Bernoulli where each dimension correspond to a pixel, when the objective is generation of black and white images. The input to the Discriminator could be either sample X_{fake} from fake data distribution, produced by the Generator $X_{fake} = G(Z) \sim p_{fake}$, or sample $X_{real} \sim p_{data}$ from real data

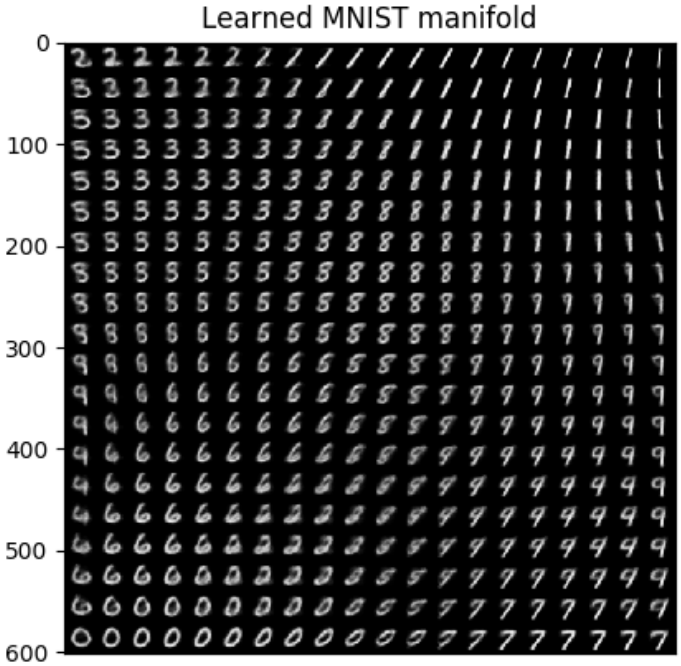


Figure 3: Output of VAE's decoder in 2-dimensional latent space

distribution. The output of the Discriminator is the probability that the input is real, so the perfect Discriminator will output 0 for X_{fake} and 1 for X_{real}.

2.1 Training objective: A Minimax Game

Question 2.2
Given the equation:

$$min_G max_D V(G, D) = min_G max_D (E_{p_{data}(x)} log D(x) + E_{p(z)} log(1 - D(G(Z)))),$$

we can conclude that the Generator's objective is to minimize $E_{p(z)} log(1 - D(G(Z)))$, so to minimize the log-probability of Discriminator predicting its output as fake. It also could be interpreted that Generator wants to produce data, such that Discriminator will classify it as real. The Discriminator aims at maximizing $E_{p_{data}(x)} log D(x) + E_{p(z)} log(1 - D(G(Z)))$. First term corresponds to maximizing the log-probability of assigning 1(true label) to the real data, and the second to maximizing the log-probability of assigning 0(fake label) to fake label. To sum up the Discriminator's objective is to discern between true and fake images and the Generator's objective is to produce images which will be classified as real by Discriminator.
Question 2.3
The optimal Discriminator could be found as:

$$E_{x \sim p_{data}} log D(x) + E_{x \sim p_{model}} log(1 - D(x)) \rightarrow max_D$$

The function $a log(x) + b log(1 - x)$ reaches maximum at $x = \frac{a}{a+b}$ and because the goal of GAN is to learn such Generator that $p_{model} = p_{data}$, we can conclude that (this is not rigorous mathematical proof because we should optimize in the functional space, take functional derivatives, and also we should prove that $p_{model} = p_{data}$ is the only optimal state):

$$D_{optimal}(x) = \frac{p_{data}}{p_{data} + p_{model}} = \frac{1}{2}$$

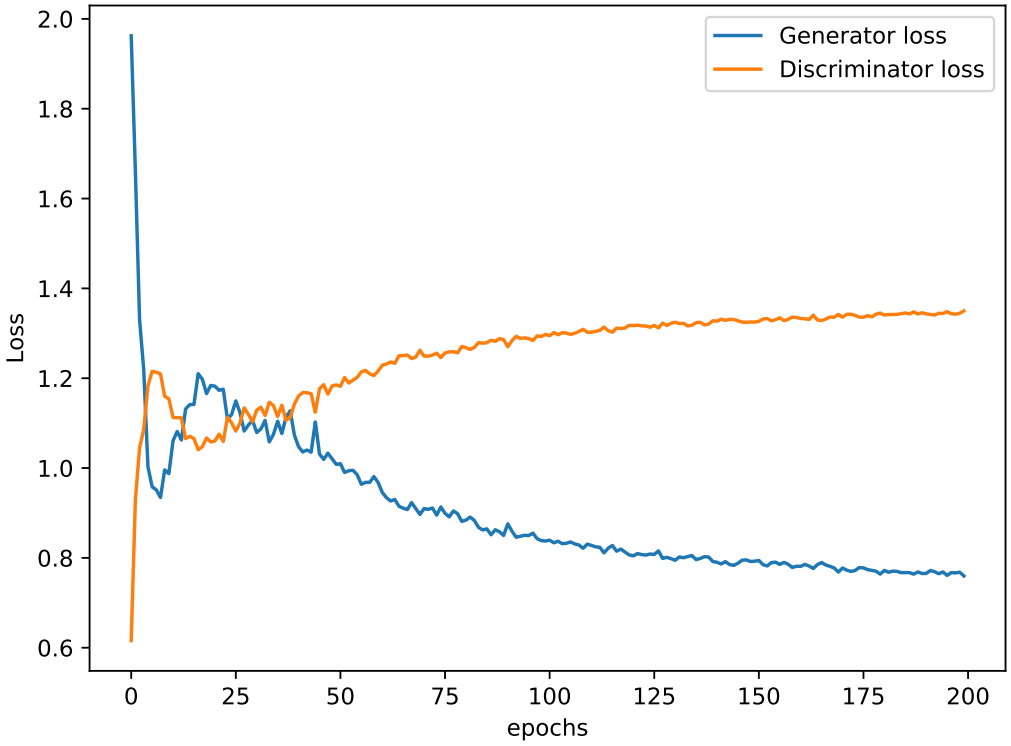


Figure 4: Loss of Generator and Discriminator networks

So the optimal loss is equal to:

$$E_{x\sim p_{data}} log D_{optimal}(x) + E_{x\sim p_{model}} log(1 - D_{optimal}(x)) = E_{x\sim p_{data}} log\frac{1}{2} + E_{x\sim p_{model}} log\frac{1}{2} = -2log2.$$

Question 2.4
As explained in Ian J. Goodfellow [2014], at the beginning of training Generator is weak and produce data which is easily detectable by Discriminator as fake, this leads to vanishing gradients of $log(1 - D(G(z)))$ term and results in poor training of Generator. To overcome this we may instead $max_G E_z(log(D(G(z))))$. These two approaches are equivalent, because $log(1 - x)$ is a monotonically decreasing function. However the former may provide better gradients to train Generator. Rigorous mathematical explanation could be found in Martin Arjovsky [2017], where first they prove that under certain assumptions Generator's gradient is upper bounded, and then prove that with perfect Discriminator the gradients will vanish. They further inspect modified objective and prove that it does not suffer from vanishing gradients but may lead to unstable updates.

2.2 Building a GAN

Question 2.5
To model the Generator I implemented proposed architecture with tanh being the activation of the output layer. The architecture of Discriminator was modified and include two dropout layers ($p_{dropout} = 0.4$), after each hidden layer, and sigmoid as output's activation. Also during training the Discriminator is not learning with probability 0.2. The Generator's and Discriminator's losses during training are visualised in 4. The Discriminator's loss converges to $2log2$ and Generator's to $log2$.

Question 2.6
Samples from model at various stage of training are illustrated at the figure 5. After the first iteration (corresponds to (a)) images are meaningless but at least grouped into clusters. At the middle stage of training GAN already generates plausible images ((b)), however some digits are corrupted. At the end of training sampled images are quite realistic.

((a)) At the beginning of the training ((b)) Half way through training ((c)) At the end of the training

Figure 5: Images sampled from GAN

Figure 6: Interpolating between two images in the latent space

Question 2.7
The interpolation between two images in the latent space is visualised in 6.

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

Question 3.1
Rewriting equation 16:

$$\mathbf{z} = f(\mathbf{x})$$

$$\mathbf{x} = f^{-1}(\mathbf{z})$$

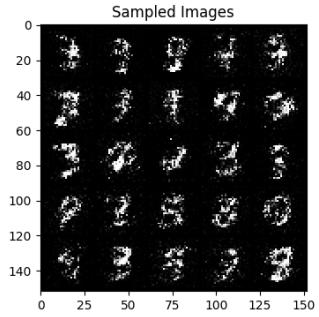
$$p(\mathbf{x}) = p(\mathbf{z})|det\frac{df}{d\mathbf{x}}|$$

Rewriting equation 17:

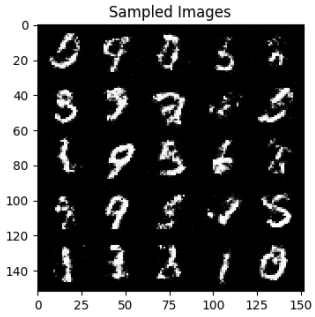
$$log(p(\mathbf{x})) = log(p(\mathbf{z})) + \sum_{l=1}^{L} log|det\frac{dh_l}{dh_{l-1}}|$$

Question 3.2
f should be smooth and the determinant of its Jacobian should be non-zero (which implies that $f : R^D \rightarrow R^D$). These properties ensures that f is invertible.
Question 3.3
The paper Laurent Dinh [2016] mentions the following issues which may arise during training (L - is the number of hidden layers, D - is the dimension of hidden layer):

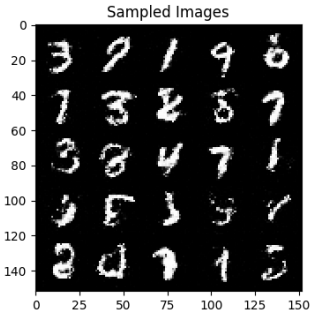
- computing determinant of Jacobian is expensive ($O(LD^3)$ computational complexity)
- computing gradient of the determinant of Jacobian is expensive ($O(LD^3)$ computational complexity)
- numerical instabilities when performing matrix inversion



((a)) After the 1-st epoch



((b)) After the 20-th epoch



((c)) After training

Figure 7: Images sampled from RealNVP

Question 3.4
Modelling discrete random variable with continuous distribution will result in collapsing probability on discrete datapoints, so when computing density function, it will take extremely huge values in these points. Also, one of the consequences is that probability density function will not be smooth anymore, because its derivative is not continuous. To overcome this, we can apply dequantization, transform discrete data into continuous. The paper Jonathan Ho [2019] considers two types of dequantization techniques:

- uniform dequantization. Simply add noise from multivariate uniform distribution: $x_{dequantized} = x + u$, with $u \sim U([0;1]^D)$
- variational dequantization. Instead of uniform distribution introduce dequantization noise distribution $q(u|x)$ (non-zero on $[0;1]^D$), which we can model using flow and learn during training.

3.2 Building a flow-based model

Question 3.5
During training the input to the network is data X (e.g. batch of images), which are then mapped by series of flow transformations into batch of latent representations Z, and by applying inverse flow transformations we output reconstruction of the original data X. During inference, we sample latent codes Z from prior, apply inverse flow transformations and output generated data X.
Question 3.6
I have implemented a simplified version of RealNVP architecture Laurent Dinh [2016]. In this architecture, flows are modeled using coupling layers, which have easily computed Jacobian. During training, when I pass an image to the network, first it should be dequantized, the reason for it are discussed in 3.4. After that, I apply transformation (logit normalize) which reduces boundary effects of the data (each pixel lies in $[0, 256]$, while the model is derived for unbounded spaces). At this stage, we can apply flows on our data, which are modelled with coupling layers with simplified architecture than in the original paper. The output of the flow layers are latent variables, which are used to compute prior log-probability and combined with a sum of log-determinants of Jacobians of coupling layers constitute log-likelihood of the data, which we maximize.
During inference time (generation of new data), we first sample latent variables from the prior distribution, then apply inverse flow transformation, and inverse logit normalize. The output is the generated data.
Question 3.7
Images sampled from the model after the 1-st, 20-th and the last epoch are visualized at 7.

Question 3.8
The average bits per dimension on training and validation sets are illustrated at 8. The model reaches 1.83 bpd at the end of the training.

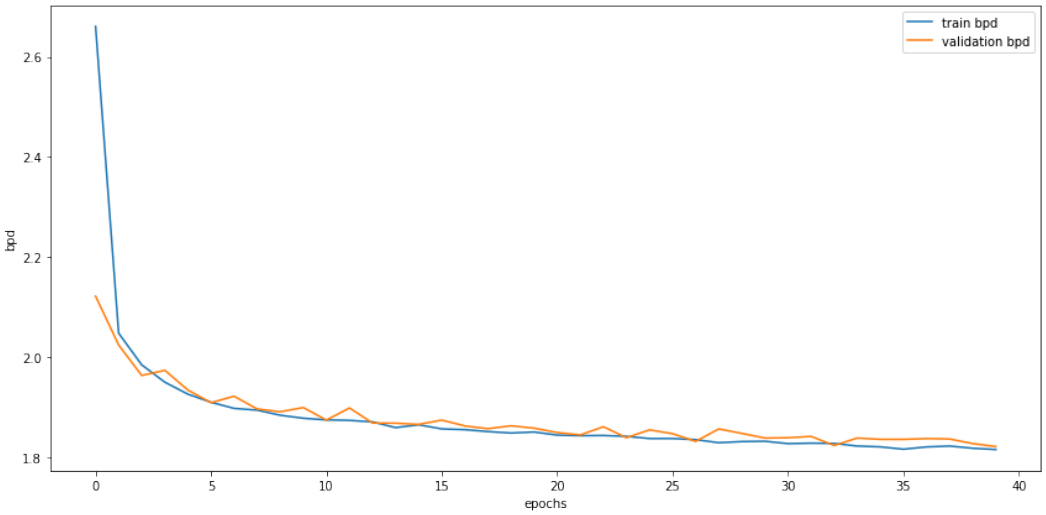


Figure 8: Training and validation performance in bits per dimension

4 Conclusion

In this assignment, I implemented VAE, GAN, and GNF, which are three different generative models. VAE and GNF models density of the data explicitly while GAN implicitly. In my opinion, GAN generates the most plausible images, while images generated with VAE have the worst quality. I think that the greatest advantage of VAE and GNF is that they allow to get the estimate of likelihood and thus quantify its performance. The performance of GNF and VAE could be improved if we use more complex networks, e.g. CNN, to model encoder and decoder. I think that among all three models GNF should be the most powerful because the approach of normalizing flows allows to obtain distributions of various complexity and thus generalizes the VAE, where distribution space is limited, and unlike GAN allows to estimate likelihood. The main downside of GNF is that they are quite slow in training, especially when flows are modelled with complex networks.

References

Kingma D. P. Salimans T. Duan Y. Dhariwal P. Schulman-J. Sutskever I. Chen, X. and P. Abbeel. Variational lossy autoencoder. 2016. URL `https://arxiv.org/pdf/1611.02731.pdf`.

Welling M. Kingma, D. P. Auto-encoding variational bayes. 2013. URL `https://arxiv.org/pdf/1312.6114`.

Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. 2014. URL `https://arxiv.org/pdf/1406.2661`.

Le on Bottou Martin Arjovsky. Towards principled methods for training generative adversarial networks. 2017. URL `https://arxiv.org/pdf/1701.04862.pdf`.

Samy Bengiou Laurent Dinh, Jascha Sohl-Dickstein. Density estimation using real nvp. 2016. URL `https://arxiv.org/pdf/1605.08803`.

Aravind Srinivas Yan Duan Pieter Abbeel Jonathan Ho, Xi Chen. Flow++: Improving flow-based generative models with variational dequantization and architecture design. 2019. URL `https://arxiv.org/pdf/1902.00275`.