Assignment 3. Deep Generative Models

Xiaoxiao Wen
12320323
xiaoxiao.wen@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

1. They are slightly different in terms of their main function. The VAE learns a parametrized probability distribution to encode or represent the training dataset and further generate/synthesize data from the estimated distribution, while the standard autoencoder learns a not parametrized arbitrary function to represent the training dataset and mainly focuses on the encoding functionality.
2. A standard autoencoder is not generative, because it does not learn a probability distribution for the input latent variable, which can be further used to generate or synthesize data accordingly. On the other hand, the main task of a standard autoencoder is to learn the latent representation of the training dataset.
3. Since VAE can be considered as an stochastic extension on the standard autoencoder to learn the probability distribution of the latent variable other than just an arbitrary function of it, it can be used to replace a standard autoencoder to gain better generalization.
4. The learned probability distribution of the latent variable helps VAE to better generalize the training dataset, while in the standard autoencoder only the deterministic function of the latent variable is learned, which is more biased towards the training dataset, thus lacking generality, and is weak in terms of generating or synthesizing new data from the dataset.

1.2 Decoder: The Generative Part of the VAE

Question 1.2

Given the provided graphical model of VAE, we know that during sampling for each image, first we have to sample the latent variable z_n from the Gaussian distribution $\mathcal{N}(0, \mathbf{I}_D)$. Then we feed the sampled z_n through the Neural Network f_θ to compute the mean of the Bernoulli distributions for each pixel in x_n. Finally, we sample each pixel in x_n from the Bernoulli distributions using the computed outputs as parameters.

Question 1.3

In the configuration of VAE, an assumption is made upon the initial latent variable $p(Z)$ that there is no simple interpretation of the dimensions of z, or can be regarded as noise. Thus, it has a simple standard-normal distribution and each dimension of z is independent and identically distributed (i.i.d.) according to the covariance matrix $\mathbf{I}_D$. As the initially standard-normal distribution of the latent variables z can be propagated and modified into the real stochastic distribution by passing z into $f_\theta(\cdot)$, this simplistic assumption made upon $p(Z)$ is not restrictive, and, in fact, efficient for training as it does not require a huge amount of trainable parameters, $D \times D$, for the covariance matrix to define a complicated distribution.

Preprint. Work in progress.

Question 1.4

(a)

$$\begin{aligned}\log p\left(x_n\right) &= \log \mathbb{E}_{p(z_n)}\left[p\left(x_n|z_n\right)\right] \\ &= \log \frac{1}{L}\sum_{l\in L} p(x_n|z_n^l)\end{aligned}$$

where L denotes a larger number of samples z_n drawn from the distribution $p(z_n)$.

(b) The Monte-Carlo Integration requires a large number of sample points from the distribution $p(z_n)$ to better approximate the true value, which is impractical under the current setting. As $p\left(x_n|z_n\right) = \prod_{m=1}^{M} \text{Bern}\left(x_n^{(m)}|f_\theta\left(z_n\right)_m\right)$ where $f_\theta(\cdot)$ is a neural network of multiple layers, each sampled z_n^l out of the total L points needs to be propagated through $f_\theta(\cdot)$, which is inefficient in terms of computation. Also, in practice, for most of sampled z_n^l, $p(x_n|z_n^l)$ tends to 0, which does not contribute to the estimation of $p(x_n)$ [1]. Furthermore, as can be observer in Figure 2 from the manual, with larger dimension of z, the total number of sampled points L required to have a proper approximation is exponentially increasing, which makes the multiple forward propagations of $f_\theta(\cdot)$ even costly.

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

Question 1.5

(a) Given $D_{\text{KL}}(q\|p) = -\mathbb{E}_{q(x)}\left[\log\frac{p(X)}{q(X)}\right] = -\int q(x)\left[\log\frac{p(x)}{q(x)}\right]dx$, we know that in order to obtain a very small KL-divergence between q and p, q has to be as close to p as possible, which indicates if $\mu_q = 0$ and $\sigma_q^2 = 1$, $D_{\text{KL}}(q\|p) = 0$. On the other hand, if q deviates a lot from p, with, for instance $\mu_q = 10$ and $\sigma_q^2 = 100$, the resulting $D_{\text{KL}}(q\|p)$ is very large.

(b)

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

Question 1.6

Given that

$$\log p\left(x_n\right) - D_{\text{KL}}\left(q\left(Z|x_n\right)\|p\left(Z|x_n\right)\right) = \mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\text{KL}}\left(q\left(Z|x_n\right)\|p(Z)\right)$$

we know the KL-divergence $D_{\text{KL}}\left(q\left(Z|x_n\right)\|p\left(Z|x_n\right)\right) \geq 0$, so we have

$$\log p\left(x_n\right) \geq \mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\text{KL}}\left(q\left(Z|x_n\right)\|p(Z)\right)$$

Hence, $\mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\text{KL}}\left(q\left(Z|x_n\right)\|p(Z)\right)$ becomes the lower bound on the log-probability $\log p(x_n)$.

Question 1.7

We must optimize the lower-bound Equation 11, because on the left-hand-side, there are no trainable parameters in the log-probability $\log p(x_n)$ and $p(Z|x_n)$ cannot be computed analytically. So only by optimizing the lower-bound on the right-hand-side of the equation, can we simultaneously maximize the log-probability and minimize the KL-divergence between $q(Z|x_n)$ and $p(Z|x_n)$, which also makes computing $p(Z|x_n)$ tractable via computing $q(Z|x_n)$ instead [1].

Question 1.8

Firstly, the first term on the left-hand-side of Equation 11 is the log-probability $\log p(x_n)$, which we can maximize by pushing up the lower-bound on the right-hand-side. Secondly, we also minimize the second term $D_{\text{KL}}(q(Z|x_n)$ on the left-hand-side simultaneously, which matches $q(Z|x_n)$ with $p(Z|x_n)$.

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

Question 1.9

$$\mathcal{L}_n^{\text{recon}} = -\mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta(x_n|Z)\right]$$
$$\mathcal{L}_n^{\text{reg}} = D_{\text{KL}}\left(q_\phi(Z|x_n)\,\|p_\theta(Z)\right)$$

As shown in the first equation, with drawing one sample to approximate the expectation, the reconstruction loss becomes $-\log p_\theta(x_n|Z)$, which is the negative log probability of obtaining the pixel values x_n of image n given the latent variable z_n. So by minimizing this loss, we are maximizing the probability of reconstructing the original image from its latent representation.

In the second equation, the regularization loss is the KL-divergence between the encoder distribution $q_\phi(Z|x_n)$ and the manually chosen latent variable distribution $p_\theta(Z)$. So while minimizing the loss, the KL-divergence is minimized which regularizes $q_\phi(Z|x_n)$ to match the distribution of $p_\theta(Z)$.

Question 1.10

Given that

$$p(z_n) = \mathcal{N}(0, I_D)$$
$$p_\theta(x_n|z_n) = \prod_{m=1}^{M} \text{Bern}\left(x_n^{(m)}|f_\theta(z_n)_m\right)$$
$$q_\phi(z_n|x_n) = \mathcal{N}\left(z_n|\mu_\phi(x_n), \text{diag}\left(\Sigma_\phi(x_n)\right)\right)$$

and the derivation we from section 1.3 and its generalization in [1] as

$$D_{\text{KL}}(q\|p) = \frac{1}{2}\left(\log\frac{1}{\sigma_q^2} + \sigma_q^2 + \mu_q^2 - 1\right)$$
$$D_{\text{KL}}(q\|p) = \frac{1}{2}\left(\log|\Sigma_q|^{-1} + \text{tr}\left(\Sigma_q\right) + \mu_q^T\mu_q - D\right)$$

We can substitute these terms to obtain the followings

$$\begin{aligned}\mathcal{L}_n^{\text{recon}} &= -\mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta(x_n|Z)\right] \\ &= -\mathbb{E}_{q_\phi(z|x_n)}\left[\log \prod_{m=1}^{M} \text{Bern}\left(x_n^{(m)}|f_\theta(z_n)_m\right)\right]\end{aligned}$$

$$\begin{aligned}\mathcal{L}_n^{\text{reg}} &= D_{\text{KL}}\left(q_\phi(Z|x_n)\,\|p_\theta(Z)\right) \\ &= D_{\text{KL}}\left(\mathcal{N}\left(z_n|\mu_\phi(x_n), \text{diag}\left(\Sigma_\phi(x_n)\right)\right)\|\mathcal{N}(0, I_D)\right) \\ &= \frac{1}{2}\left(\log|\,\text{diag}\left(\Sigma_\phi(x_n)\right)|^{-1} + \text{tr}\left(\text{diag}\left(\Sigma_\phi(x_n)\right)\right) + \mu_\phi^T(x_n)\,\mu_\phi(x_n) - D\right) \\ &= \frac{1}{2}\left(-\log\prod_{d=1}^{D}(\Sigma_\phi(x_n)_d) + \sum_{d=1}^{D}(\Sigma_\phi(x_n)_d) + \mu_\phi^T(x_n)\,\mu_\phi(x_n) - D\right) \\ &= \frac{1}{2}\left(-\sum_{d=1}^{D}\log\Sigma_\phi(x_n)_d + \sum_{d=1}^{D}\Sigma_\phi(x_n)_d + \sum_{d=1}^{D}\mu_\phi^2(x_n)_d - \sum_{d=1}^{D}1\right) \\ &= \frac{1}{2}\sum_{d=1}^{D}\left(-\log\Sigma_\phi(x_n)_d + \Sigma_\phi(x_n)_d + \mu_\phi^2(x_n)_d - 1\right)\end{aligned}$$

1.5 The Reparametrization Trick

Question 1.11

(a) We need to compute $\nabla_\phi\mathcal{L}$ such that we can update the parameters ϕ for $q_\phi(z_n|x_n)$ in order to match $p_\theta(Z)$, and further optimize the lower-bound $\mathcal{L}$ which is the main objective.

(b) Because after sampling, the dependency of the reconstruction loss $\mathcal{L}_n^{\text{recon}}$ w.r.t. the parameters ϕ disappears for $\mathcal{L}_n^{\text{recon}} = \log p_\theta(x_n|Z)$ where $\nabla_\phi \mathcal{L} = 0$. Furthermore, sampling from $q_\phi(z_n|x_n)$ is a non-continuous operation and has no gradient itself [1]. So during backward-propagation, there is no update for the parameters ϕ, which is undesired.

(c) The reparametrization trick is to decompose the sampling from $q_\phi(z_n|x_n) = \mathcal{N}(z_n|\mu_\phi(x_n), \text{diag}(\Sigma_\phi(x_n)))$ into first sampling ϵ from $\mathcal{N}(0, I)$, then computing $z_n = \mu_q(x_n) + \text{diag}^{1/2}(\Sigma_\phi(x_n)) * \epsilon$ [1]. With this decomposition, the reconstruction loss becomes $\mathcal{L}_n^{\text{recon}} = \log p_\theta(x_n|z_n = \mu_q(x_n) + \text{diag}^{1/2}(\Sigma_\phi(x_n)) * \epsilon)$ which is dependent on ϕ after sampling ϵ from $\mathcal{N}(0, I)$, so the $\nabla_\phi \mathcal{L} \neq 0$ and the update can be computed.

1.6 Putting things together: Building a VAE

Question 1.12

Firstly, the classes `Encoder` and `Decoder` are implemented according to the architectures for the encoder and the decoder introduced in [2]. For better performance, all the Tanh activation layers from the original work are replaced by ReLU activation layers. For the encoder, when estimating the standard deviations for the approximate posterior distribution, an additional ReLU activation is imposed to enforce positivity. Then, the class `VAE` is implemented based upon the `Encoder` and `Decoder` class. The forward propagation is implemented as described in [2, 1], while the binary cross entropy loss is used for the reconstruction loss according to [2]. Furthermore, when computing the logarithms, a small constant value is added to clip $\log(x)$ when $x \to 0$.

Question 1.13

The estimated negative lower-bounds are shown in Figure 1. Given a random initialization for the parameters, the reconstruction loss $\mathcal{L}_n^{\text{recon}} \approx -784 \log 0.5 = 543$, as every pixel in an image has a 50% chance of being correct; the regularization loss $\mathcal{L}_n^{\text{reg}} \approx 0$ as the initialized encoder distribution is similar to the prior distribution. Hence, the estimated negative lower-bounds in the plot are valid.

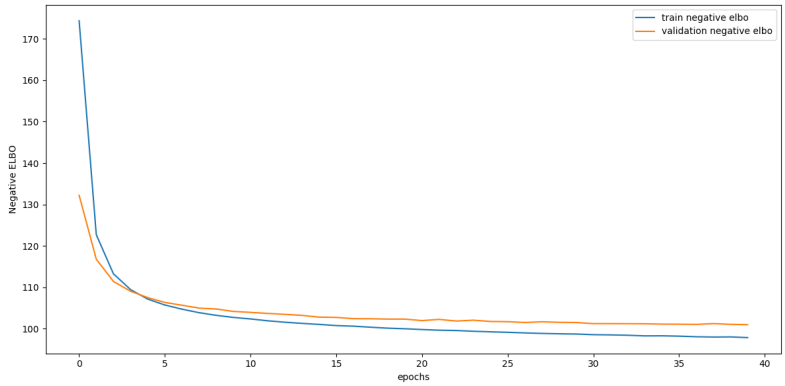


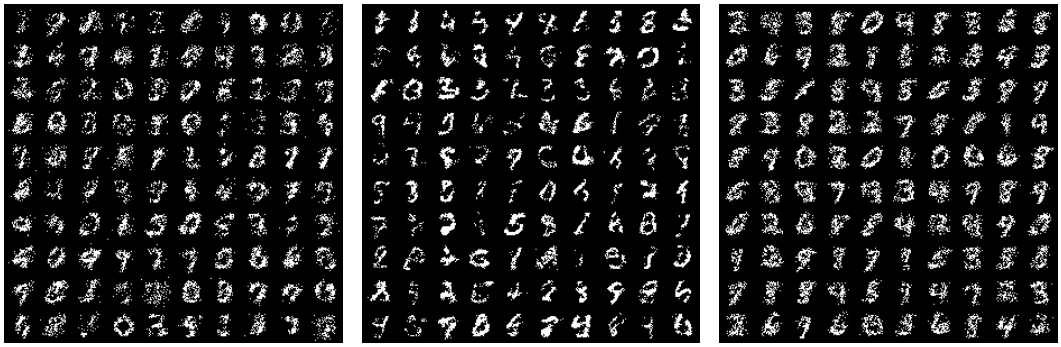
Figure 1: *Average negative ELBO per epoch for a VAE using 20-dimensional latent space.*

Question 1.14

The results are shown in Figure 2. As can be observed, the quality of the generated digits are gradually improved during the training. A drastic improvement can be seen half way through the training compared to that before training and the improvement is marginal comparing the digits after the training and half way through, which is also indicated in the estimated negative lower-bounds in Figure 1.

Question 1.15

The result is shown in Figure 3. As can be seen, as the two latent dimensions range from 0 to 1 with 20 steps in the inverse probability distribution, the corresponding digits generated are shown, which



(a) **(b)** **(c)**

Figure 2: *Sampled images (a) before training, (b) during training and (c) after training for a VAE using a 20-dimensional latent space.*

covers a relatively large proportion of the digits/datapoints in the original MNIST dataset. This shows that the VAE can properly learn the latent representation of the training dataset.

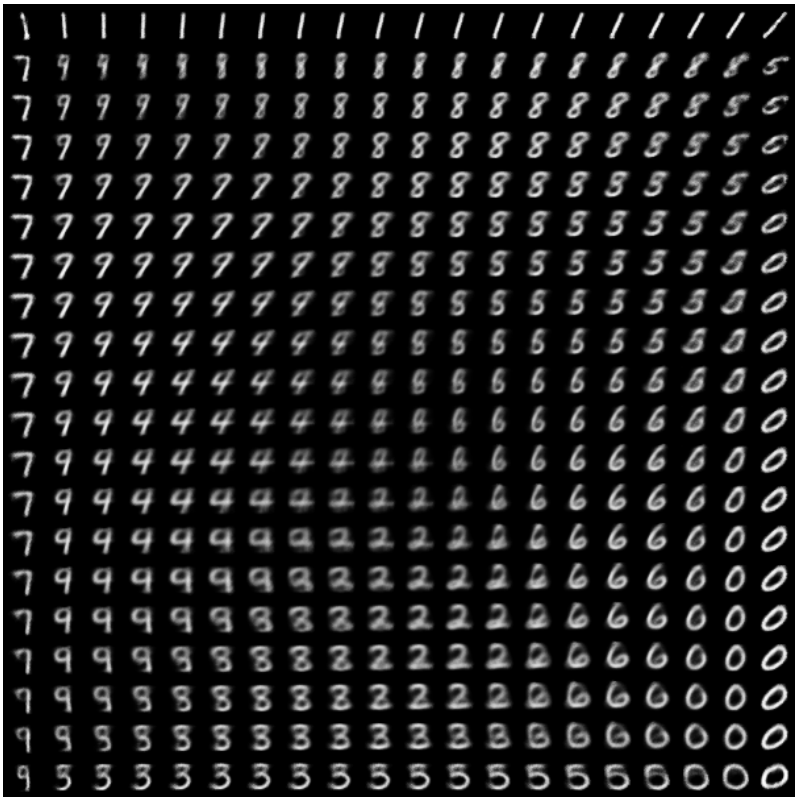


Figure 3: *Manifold for a VAE with a 2-dimensional latent space.*

2 Generative Adversarial Networks

Question 2.1

The input to the generator is random noise and the output of it is the generated image. The input to the discriminator is an image, either synthesized or from the training set, and the output of it is the confidence of the discriminator for the input image being real (tends to 1 if real).

2.1 Training objective: A Minimax Game

Question 2.2

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

In the equation, the first term is interpreted as when drawing samples x from the training set $p_{data}(x)$, the (log) probability of the discriminator function $D(X)$ judging the real image to be real is maximized by D; the second term is interpreted as when drawing noise samples z from the noise distribution $p_z(z)$, the (log) probability of the discriminator function $1 - D(G(Z))$ judging the generated sample $G(Z)$ to be fake is maximized by D and minimized by G.

Question 2.3

Let $p_g(x)$ denote the distribution of the generated samples of the generator G, then we can rewrite the loss function as

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

By first fixing the discriminator D, we would like to have the generator G to learn the distribution of the dataset to generate perfect samples, so we have $p_{\text{data}}(x) = p_g(x)$. Then writing out the expectations, we have

$$\begin{aligned}\min_G \max_D V(D,G) &= \min_G \max_D \mathbb{E}_{p_{\text{data}}(x)}[\log D(X)] + \mathbb{E}_{p_g(x)}[\log(1 - D(X))] \\ &= \min_G \max_D \int_x \left(p_{\text{data}}(x)\log(D(x)) + p_g(x)\log(1 - D(x))\right) dx\end{aligned}$$

Trivially, for $(p_{\text{data}}(x)\log(D(x)) + p_g(x)\log(1 - D(x)))$, the optimal value is $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)} = \frac{1}{2}$. So the optimal value for $V(D,G)$ is $2\log(\frac{1}{2})$.

Question 2.4

At early stages, the generator $G(Z)$ cannot generate realistic samples and, therefore, the discriminator $D(G(Z))$ would be able to recognize and reject these fake samples and the gradient w.r.t. $G(Z)$ vanishes. In order to solve the problem, a second loss function is introduced as

$$\max_G V'(D,G) = \max_G \mathbb{E}_{p_z(z)}[\log D(G(Z))]$$

such that the generator $G(Z)$ maximizes the probability of the discriminator $D(G(Z))$ being incorrect and the gradient still exists w.r.t. $G(Z)$ even if the discriminator $D(G(Z))$ rejects the fake samples with high probability.

2.2 Building a GAN

Question 2.5

The `Generator` nad `Discriminator` classes are implemented as required in the templates, where the final output layer non-linearities are imposed by using a Tanh function for the generator and using a Sigmoid function for the discriminator. Furthermore, as a trick to stabilize the optimization between the two losses and, thus, to increase the performance, a dropout layer with drop-out probability of 0.5 is added to after the first activation layer in the discriminator architecture. For training, the implementation first starts with training the generator to generate images and to minimize the confidence of the discriminator judging the these images to be fake. Then the implementation continues with training the discriminator with the real images from the dataset to maximize the confidence of the discriminator judging these to be real and then training with the generated/fake images from the generator to maximize the confidence judging these to be fake.

Question 2.6

The results are shown in Figure 4 and Figure 5. As can be observed in Figure 5, the randomly initialized Generator can only generate white noises before training, while half way through and after training, the quality of the generated digits are rather impressive.

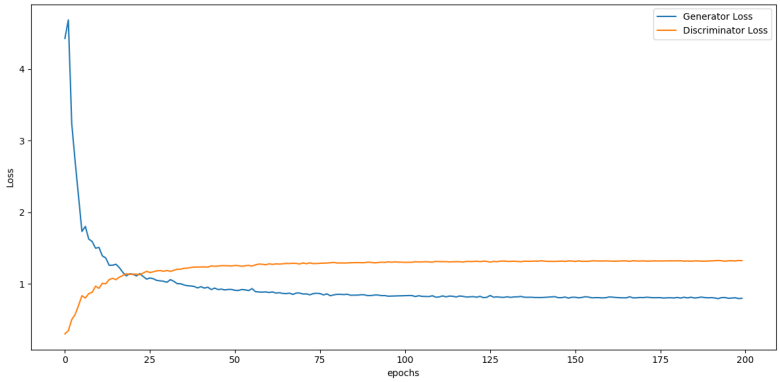


Figure 4: *Training losses for the Generator and the Discriminator.*

(a) **(b)** **(c)**

Figure 5: *Sampled images (a) before training, (b) during training and (c) after training for a GAN.*

Question 2.7

The result is shown in Figure 6. Interpolation is done between a sampled 2 and a sampled 6 in the latent space. The transformation in the shape is clearly shown in the 7 steps.

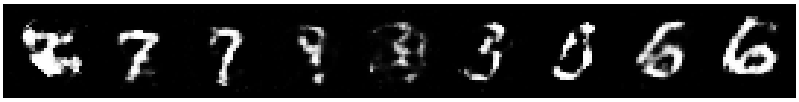


Figure 6: *Interpolation between 2 samples from the trained GAN.*

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

Question 3.1

Given that

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

for $x \in \mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}^m$ we can know that

$$\begin{aligned} p(z) &= p(x) \left| \det(\frac{df}{dx}) \right|^{-1} \\ &= p(x) \left| \det(\frac{df}{dx})^{-1} \right| \\ &= p(x) \left| \det(\frac{df^{-1}}{dz}) \right| \end{aligned}$$

and

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

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

Question 3.2

In order to compute the determinant of either f or f^{-1}, f needs to be a square matrix, which indicates that the mapping from x to z needs to stay in the same dimension, so x and z are of the same dimension.

Furthermore, to ensure that the function f is invertible, the determinant of the Jacobian matrix $\frac{df}{dx}$ needs to be non-zero, so $\det(\frac{df}{dx}) \neq 0$.

Question 3.3

For a single mapping, given that x and z have the dimension of M, directly computing the determinant of f has a computational complexity of $O(M^3)$ and is rather computationally heavy. If there are L mappings in the sequence, then the total computational complexity becomes $O(LM^3)$ per optimization, which is even costly.

Question 3.4

As the continuous values are quantized into discrete integers, the probability distribution becomes discrete and the probability mass are placed on discrete points, which produces a degenerate solution[3]. In order to tackle this problem, dequantization is used to convert the discrete integers into continuous values and then the resulting continuous distribution is modelled by the flow-based model (NF).

3.2 Building a flow-based model

Question 3.5

During training, the input of the flow-based model is the datapoint x from the dataset and the outputs are the latent representation z corresponding to x after transformations through the sequence of normalization flows and the computed (log-)likelihood $\log p(x)$; during inference time, the input of the flow-based model is the latent variable z sampled from the latent distribution $p(z)$ and the output is the generated datapoint $\hat{x}$ after the transformations through the reversed sequence of normalization flows.

Question 3.6

During training, first a datapoint/sample x needs to be drawn from the training dataset, then x is dequantized to convert it to continuous values and then logit-normalized as described in [4]. Subsequently, x is propagated through the sequence of normalizing flows to compute its latent representation z and the cumulative determinants. During the propagation, an affine coupling layer is used to do affine transformation on a split of (half) of the input and the split is alternating for each affine transformation. The affine coefficients are computed via a neural network. Then the

log-likelihood $\log p(x)$ is computed accordingly and its negated loss is backward propagated to optimize the model.

During inference, first a latent variable z is sampled from the standard normal distribution and propagated in the reversed sequence of normalizing flows. Then the result is propagated to the reversed logit-normalization to generate the sample $\hat{x}$.

Question 3.7

The implementation is done according to [4] and as described in section 3.2 with some slight modifications. A shared neural network architecture with 3 fully-connected layers and ReLU activations in between is used for estimating the affine coefficients s and t. To split the input data during affine coupling, the checkerboard pattern is used. Furthermore, for computational stability, for the scale coefficient s and the determinant of the Jacobians, logarithmic operation is done.

Question 3.8

The results are shown in Figure 7 and Figure 8. As can be seen, the model reaches a performance below 1.85 bits per dimension after 40 epochs, and there are gradual improvements in the generated samples before training, half way thorough the training and after training.

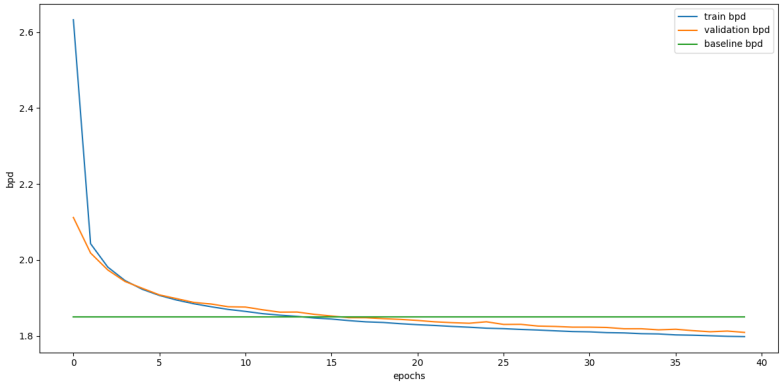
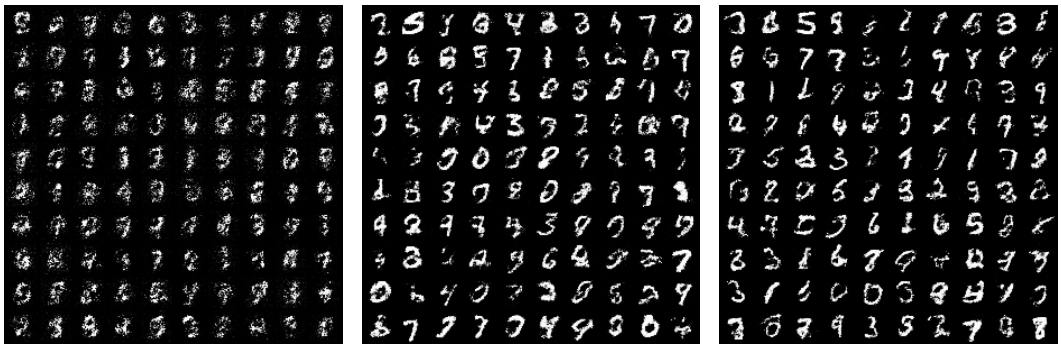


Figure 7: *Training and validation performance in bits per dimension for the flow-based model.*



(a) **(b)** **(c)**

Figure 8: *Sampled images (a) before training, (b) during training and (c) after training for a flow-based model.*

4 Conclusion

Question 4.1

By comparing the quality of the generated samples Figure 2, Figure 5 and Figure 8 respectively for VAE, GAN and NF, we can observe that from high to low, the order is roughly GAN->NF->VAE. The training time for these models on the same setting are 324s, 5048s and 492s respectively with respect to 40epochs, 200epochs and 40epochs. By computing the unit time per epoch training for these models, we can have a trivial proxy for the computational cost for the models, which are $8.1\text{s}/\text{epoch}$, $25.24\text{s}/\text{epoch}$ and $12.3\text{s}/\text{epoch}$, ordered by GAN->NF->VAE. So combining the quality of the generated samples with the computational cost for the three models, to generate realistic samples, GAN has the best performance in terms of image quality but is also the most computationally costly; VAE has the worst quality but is least computationally costly; NF is in between the two models and has a slightly better image quality but also slightly more computational cost.

According to their theories, the coverage of the data distribution of the three models can be ordered as NF->VAE->GAN, because NF learns a latent distribution of the same dimension as the dataset, VAE learns an approximate latent distribution of decreased dimension and GAN does not the model the distribution in a stochastic manner.

References

[1] C. Doersch, "Tutorial on Variational Autoencoders," jun 2016.

[2] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," dec 2013.

[3] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design," feb 2019.

[4] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP," may 2016.