Assignment 3. Deep Generative Models

Kai Liang
University of Amsterdam
kai.liang@student.uva.nl

Contents

1 Variational Auto Encoders

Question 1.1

The VAE is related to and/or different from a standard autoencoder in the following aspects.

1. Their main functions are different. For the standard autoencoder, the objective is to minimize the reconstruction loss, which is simply the mean square error. On the other hand, as the VAE is embedded with probability distributions, the main objective is to maximize the data log-likelihood, or minimize the negative ELBO (which consists of a reconstruction loss and a regularization term).
2. Although a standard autoencoder can be used for generation (e.g. by feeding a latent representation to the decoder), it is not considered generative for two reasons. Firstly, the typical task of a standard autoencoder is not for generating synthetic data. Secondly, a standard autoencoder does not model a probability distribution of the latent variable.
3. There are cases where a standard autoencoder is preferred over the VAE, for instance, when the size of the dataset is small (since standard autoencoders are easier to train). Besides, a shortcoming of the VAE is that, when the decoder is too powerful, the model will just ignore the latent representations, meaning that we cannot use them for better representing our data[1].
4. The VAE is able to generalize in contrast to the standard autoencoders as it defines a probability distribution over the latent variable (instead of mapping deterministically), from which we can sample to generate synthetic data. Therefore, the VAE is not biased towards the training data.

Question 1.2

The procedure to sample from such a model contains the following steps.

[1] http://paulrubenstein.co.uk/variational-autoencoders-are-not-autoencoders/

1. Sample z_n from the $\mathcal{N}(0, I_D)$ distribution.
2. Compute $f_\theta(z_n)$ according to the values of z_n and the function approximator f_θ.
3. Sample $x_n^{(m)}$ from the Bern $(f_\theta(z_n)_m)$ distribution for every m in M.

Question 1.3

The point is that, according to Doersch [2016], any distribution in D dimensions can be approximated by sampling D variables from a standard normal distribution and mapping them with a function that is complicated enough (e.g. f_θ in Equation 4). Hence, by using a neural network which is a powerful function approximator, we can learn a function that maps the values of z_n to the latent variables that are needed for generating every x_n.

Question 1.4

(a) Using Monte-Carlo Integration, the expression can be rewrite as:

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

where $z_n^{(s)}$ is sampled from $p(z_n)$ and S is the total number of samples.

(b) Using this approach to estimate $\log p(x_n)$ is inefficient, which based on Doersch [2016], is because for most $z_n^{(s)}$, $p\left(x_n|z_n^{(s)}\right)$ will be around 0, which hardly contribute to the estimation. To speed up sampling, we want to sample $z_n^{(s)}$ that are likely to produce x_n, which should fall only in a small region of the original $p(z_n)$ (i.e. $p(z_n|x_n)$ in Figure 2). Otherwise, given that in high-dimension spaces, the region where $p\left(x_n|z_n^{(s)}\right)$ is near 0 will be larger, if the dimensionality of z increases, the efficiency will scale negatively.

Question 1.5

(a) Two examples are:

- When $\mu_q = 0.1$ and $\sigma_q^2 = 1$, $D_{\mathrm{KL}}(q\|p) = 0.005$, which is very small.
- When $\mu_q = 100$ and $\sigma_q^2 = 100$, $D_{\mathrm{KL}}(q\|p) = 5047.197$, which is very large.

(b) The closed-form formula for $D_{\mathrm{KL}}(q\|p)$ is:

$$\begin{aligned}D_{\mathrm{KL}}(q\|p) &= -\int q(x)\left[\log\frac{p(x)}{q(x)}\right]dx \\ &= \log\frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2}\end{aligned}$$

Given $\mu_p = 0$ and $\sigma_p^2 = 1$, $D_{\mathrm{KL}}(q\|p) = -\log\sigma_q + \frac{\sigma_q^2 + \mu_q^2}{2} - \frac{1}{2}$.

Question 1.6

The reason is that, the term $D_{\mathrm{KL}}\left(q\left(Z|x_n\right)\|p\left(Z|x_n\right)\right)$ is always positive (i.e. ≥ 0), thus:

$$\begin{aligned}\log p(x_n) - D_{\mathrm{KL}}\left(q\left(Z|x_n\right)\|p\left(Z|x_n\right)\right) &= \mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\mathrm{KL}}\left(q\left(Z|x_n\right)\|p(Z)\right) \\ \log p(x_n) &\geq \mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\mathrm{KL}}\left(q\left(Z|x_n\right)\|p(Z)\right)\end{aligned}$$

Hence, the right-hand-side of the equation is called the lower bound on the log probability.

Question 1.7

We need to optimize the lower-bound instead of the log-probability directly, because optimizing the log-probability is not tractable as there is no analytical solution to $p\left(Z|\boldsymbol{x}_n\right)$. Thus, we cannot move $D_{\mathrm{KL}}\left(q\left(Z|\boldsymbol{x}_n\right)\|p\left(Z|\boldsymbol{x}_n\right)\right)$ to the right-hand side for optimization [Doersch, 2016].

Question 1.8

The two things which could happen when the lower bound is pushed up are as follows. Firstly, the $\log p\left(\boldsymbol{x}_n\right)$ term (a.k.a. the log likelihood of our data) is increased, which is the quantity we want to maximize. Secondly, the $D_{\mathrm{KL}}\left(q\left(Z|\boldsymbol{x}_n\right)\|p\left(Z|\boldsymbol{x}_n\right)\right)$ term is decreased, because it is an error term between the intractable real distribution $p\left(Z|\boldsymbol{x}_n\right)$ and our approximated distribution $q\left(Z|\boldsymbol{x}_n\right)$, which will be lower during training as $q\left(Z|\boldsymbol{x}_n\right)$ is pulled to match $p\left(Z|\boldsymbol{x}_n\right)$ [Doersch, 2016].

Question 1.9

The $\mathcal{L}_n^{\text{recon}}$ term is called the reconstruction loss because z is sampled from $q_\phi\left(z|\boldsymbol{x}_n\right)$, which is encoded by $\boldsymbol{x}_n$, and the expectation is computed for $p_\theta\left(\boldsymbol{x}_n|Z\right)$ (a.k.a. 'decoding' z to reconstruct $\boldsymbol{x}_n$). The $\mathcal{L}_n^{\text{reg}}$ term is called the regularization term because it is a KL-divergence between the prior and the estimated posterior distribution of the latent space, which needs to be minimized as we want the encoder $q_\phi\left(Z|\boldsymbol{x}_n\right)$ to be similar to the prior $p_\theta(Z)$.

Question 1.10

To minimize $\mathcal{L}$, for each $\boldsymbol{x}_n$ in $\mathcal{D}$, we should:

1. Pass $\boldsymbol{x}_n$ into the encoder, compute $\mu_\phi\left(\boldsymbol{x}_n\right)$ and $\Sigma_\phi\left(\boldsymbol{x}_n\right)$.
2. Sample $\boldsymbol{z}_n$ from $q_\phi\left(\boldsymbol{z}_n|\boldsymbol{x}_n\right)=\mathcal{N}\left(\mu_\phi\left(\boldsymbol{x}_n\right),\mathrm{diag}\left(\Sigma_\phi\left(\boldsymbol{x}_n\right)\right)\right)$.
3. Pass $\boldsymbol{z}_n$ into the decoder, compute $f_\theta\left(\boldsymbol{z}_n\right)$.
4. Compute $p\left(\boldsymbol{x}_n|\boldsymbol{z}_n\right)=\prod_{m=1}^{M}\mathrm{Bern}\left(\boldsymbol{x}_n^{(m)}|f_\theta\left(\boldsymbol{z}_n\right)_m\right)$.
5. Compute $\mathcal{L}_n^{\text{recon}}=-\mathbb{E}_{q_\phi\left(\boldsymbol{z}_n|\boldsymbol{x}_n\right)}\left[\log p_\theta\left(\boldsymbol{x}_n|\boldsymbol{z}_n\right)\right]=-\log\prod_{m=1}^{M}\mathrm{Bern}\left(\boldsymbol{x}_n^{(m)}|f_\theta\left(\boldsymbol{z}_n\right)_m\right)$.
6. Compute $\mathcal{L}_n^{\text{reg}}=\frac{1}{2}\left(\mathrm{tr}\left(\Sigma_\phi\left(\boldsymbol{x}_n\right)\right)+\mu_\phi\left(\boldsymbol{x}_n\right)^\top\mu_\phi\left(\boldsymbol{x}_n\right)-D-\log\det\left(\Sigma_\phi\left(\boldsymbol{x}_n\right)\right)\right)$ [Doersch, 2016].
7. Compute $\mathcal{L}_n=\mathcal{L}_n^{\text{recon}}+\mathcal{L}_n^{\text{reg}}$.

Question 1.11

(a) We need $\nabla_\phi\mathcal{L}$ for the backpropagation to perform stochastic gradient descent to update the parameters of the encoder.

(b) The act of sampling prevents us from computing $\nabla_\phi\mathcal{L}$ as sampling z from $q_\phi\left(z|\boldsymbol{x}_n\right)$ is a non-continuous operation and thus has no gradient [Doersch, 2016].

(c) The reparametrization trick is to move the sampling to an input layer. Given that z is sampled from $\mathcal{N}(\mu_\phi\left(\boldsymbol{x}_n\right),\Sigma_\phi\left(\boldsymbol{x}_n\right))$, we can first sample $\boldsymbol{\epsilon}$ from $\mathcal{N}(\mathbf{0},\mathbf{I})$, and compute z according to $z=\mu_\phi\left(\boldsymbol{x}_n\right)+\Sigma_\phi\left(\boldsymbol{x}_n\right)\odot\boldsymbol{\epsilon}$ [Kingma and Welling, 2013].

Question 1.12

The detailed implementation of the VAE model could be found in `a3_vae_template.py`. In short, the architecture of the network are as follows. For the encoder, the network first applies a linear layer to transform the data from the input dimension to the hidden dimension, followed

by a ReLU activation layer. Then, the result is separately passed to two linear layers to the size of the latent dimension as the mean and log variance (instead of the variance, which cannot be negative). For the decoder, the network first applies a linear layer to transform the data from the latent dimension to the hidden dimension, followed by a ReLU activation layer. Then, the result is passed to another linear layer to the input dimension, followed by a Sigmoid activation layer, which are then the means of the Bernoulli distribution. It is worth noting that, to compute the reconstruction loss, the model uses directly the output of the decoder as the reconstructed input, and the loss is calculated using the binary cross entropy function [Kingma and Welling, 2013].

Question 1.13

Using a 20-dimensional latent space, the estimated negative lower-bounds of the training and validation set as training progresses are shown in Figure 1. According to the graph, the negative lower-bounds starts at around 170, which gradually decreases and stabilizes at around 100 in the end.

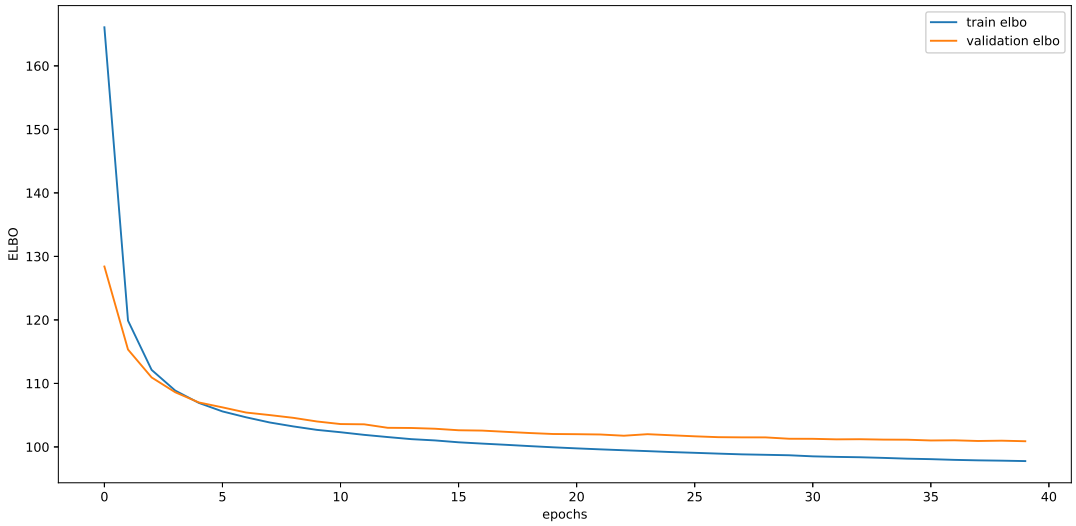


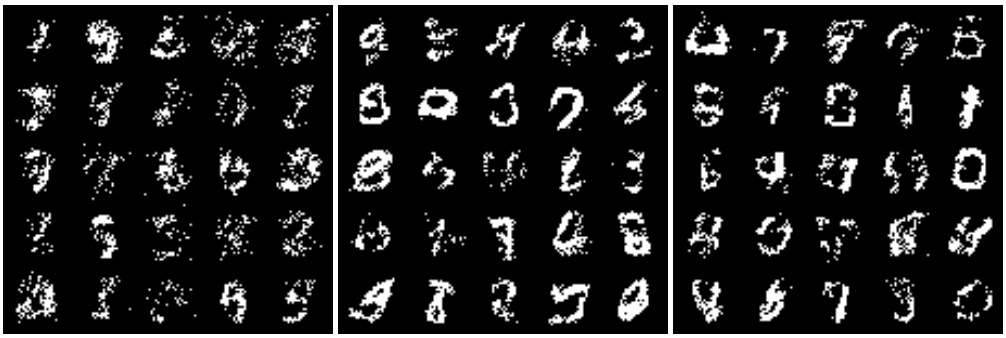
Figure 1: *The estimated negative lower-bounds of the training and validation set.*

Question 1.14

The network is trained using a 20-dimensional latent space and for 40 epochs, and the samples from the model at epoch 0, epoch 20 and epoch 40 with and without sampling from the Bernoulli distribution are displayed in Figure 2 and Figure 3 respectively. Comparing the two figures, we have the following observations. Firstly, there is an improvement in the quality of samples for both methods as training processes. Secondly, sampling from the Bernoulli distribution results in sharper images, while taking the means directly results in smoother images.

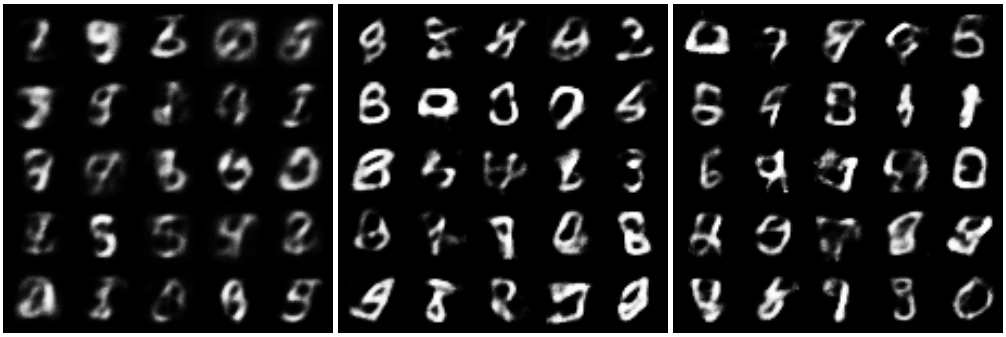
Question 1.15

With a 2-dimensional latent space, the data manifold after training is displayed in Figure 4. In short, the percent point function is used to cover the part of Z-space that has significant density in the range of $[0.0001, 0.9999]$ with 15 points in each dimension.



(a) $Epoch = 1$. **(b)** $Epoch = 20$. **(c)** $Epoch = 40$.

Figure 2: *Sample images generated by the VAE during different phases of training with sampling.*



(a) $Epoch = 1$. **(b)** $Epoch = 20$. **(c)** $Epoch = 40$.

Figure 3: *Sample images generated by the VAE during different phases of training without sampling.*

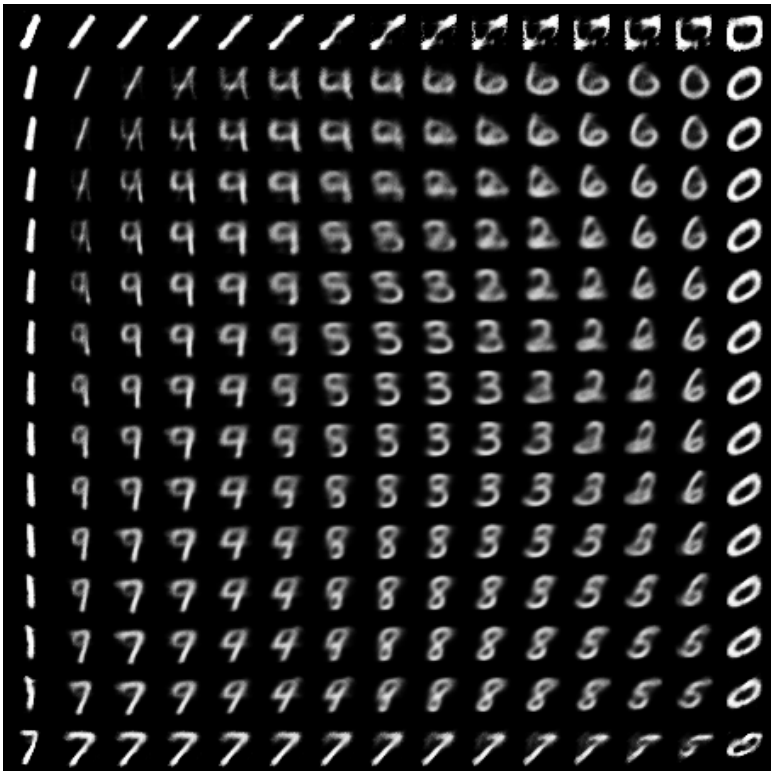


Figure 4: *The data manifold after training with a 2-dimensional latent space.*

2 Generative Adversarial Networks

Question 2.1

For the generator, the input is an encoding vector sampling from the noise distribution, and the output is an generated image (i.e. a matrix or tensor of pixel values). For the discriminator, the input is an image which is either an generated image by the generator or a real image in the training set, and the output is a value between 0 and 1 indicating the probability of the image being a real image.

Question 2.2

The first term (i.e. $\mathbb{E}_{p_{\text{data}}(x)}[\log D(X)]$) in the GAN's optimization objective denotes the expected log-probability that the discriminator correctly classifies real images as real, while the second term (i.e. $\mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$) denotes the expected log-probability that the discriminator correctly classifies fake (a.k.a. generated) images as fake.

As this is a minimax game, the objective of the generator is to minimize the two terms above, and on the other hand, the objective of the discriminator is to maximize the two terms.

Question 2.3

After the training has converged, the discriminator would not be able to distinguish fake (a.k.a. generated) images from the real images, so the output of the discriminator would always be 0.5 (i.e. equal chance of being real or fake). Hence, the value of $V(D, G)$ in this case is: $\log \frac{1}{2} + \log \frac{1}{2}$, which is $-\log 4$.

Question 2.4

Early on during training, the generator is poor and its generated images can be easily recognized as fake images by the discriminator with high confidence. Therefore, the term $\log(1 - D(G(Z)))$ can be problematic as its value will be very close to 0, which causes the problem of vanishing gradient and makes the learning very hard for the generator. To solve the problem, Goodfellow et al. [2014] suggested to replace the term with $\log D(G(Z))$, which the generator is then trained to maximize.

Question 2.5

The final implementation of the GAN model could be referred in `a3_gan_template.py`. In short, the architecture of the network generally follows the given template. For the generator, it uses multiple linear layers and LeakyReLU as activation functions in between with batch normalization applied. In the last layer, the Tanh activation function is used to map the values of output to $[-1, 1]$. Similarly, for the discriminator, it uses multiple linear layers and LeakyReLU as activation functions in between to map down the input size, and uses a Sigmoid activation in the last layer to output a classification score between 0 and 1. Different from the given template, a dropout layer is used in the discriminator both during training and testing to ensure better generalization. Besides, according to the online guide of training GANs[2], the following tricks are implemented to have a better performance of the generator. Firstly, the noise used for the input of the generator is sampled from a standard normal distribution. Secondly, a modified cross entropy loss is used for the generator to avoid the vanishing gradient problem. Finally, noisy labels are used for both the discriminator and the generator.

Question 2.6

The network is trained for a total of 200 epochs on the MNIST data, in Figure 5 below, 25 sample images generated by the network are shown at the start of training, halfway through training and after training has terminated. As can be inferred from the figure, the network indeed learns

[2]https://github.com/soumith/ganhacks

(a) $Epoch = 1$. **(b)** $Epoch = 100$. **(c)** $Epoch = 200$.

Figure 5: *Sample images generated by the GAN during different phases of training.*

the correct pattern of digits for generation as the training evolves.

Question 2.7

After training, 2 images of different classes are sampled and interpolated for 7 different steps in the latent space, which gives an indication of transformation between the 2 images. In Figure 6 below, we can clearly see that the digit 1 is first transformed into the digit 9, and finally into the sampled digit 0.

Figure 6: *Images obtained by linear interpolation between* 2 *samples in the latent space.*

3 Generative Normalizing Flows

Question 3.1

If $f : \mathbb{R}^m \to \mathbb{R}^m$ is an invertible mapping and $\mathbf{x} \in \mathbb{R}^m$ is a multivariate random variable[3]:

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

$$\log p(\mathbf{x}) = \log p(\mathbf{z}) + \sum_{l=1}^{L} \log \left| \det \frac{\partial h_l}{\partial h_{l-1}} \right| \tag{2}$$

As can be inferred from Equation 1 and 2, the difference between uni- and multivariate variables on the equations lies in that the partial derivative for multivariate variables are replaced by the determinant of the Jacobian matrix.

Question 3.2

Thinking about the number of dimensions in $\mathbf{z}$ and $\mathbf{x}$, the constraints that have to be set on the function f is that it needs to be an $\mathbb{R}^m \to \mathbb{R}^m$ mapping (given $\mathbf{x} \in \mathbb{R}^m$), which ensures that the Jacobian determinant is computable. Besides, the function f should be invertible so that the input to the logarithm in Equation 2 will not be 0.

Question 3.3

A computational issue that arises when optimizing the network is that computing the Jacobian of high-dimensional data and computing the determinants of such matrices are computationally

[3]https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

expensive [Dinh et al., 2016]. Hence, we need to make sure that the function f is easily invertible and its Jacobian determinant can be easily computed.

Question 3.4

The consequence of using flow models on discrete data is that, the continuous density models will place all probability mass on the possible discrete datapoints, which will result in arbitrarily high density values [Uria et al., 2013]. To solve the problem, we could first transform the discrete data into a continuous distribution via dequantization (e.g. adding uniform noise to each pixel), and then apply the flow models on the resulting distribution [Ho et al., 2019].

Question 3.5

During training, the input of the flow-based model is the real data (e.g. images), and the output is the log probability of the input. During inference time, the input of the model is latent vectors sampled from the prior distribution, and the output is generated data (e.g. images) of the input.

Question 3.6

To train a flow-based model, the steps are:

1. Prepare real data (e.g. images) and feed to the model.
2. Dequantize the data and apply the logit normalization.
3. Pass the data to each coupling layer to apply the masked convolution according to:
$$y = b \odot x + (1 - b) \odot (x \odot \exp(s(b \odot x)) + t(b \odot x))$$
where b is the checkerboard pattern mask [Dinh et al., 2016]. Also, keep track of the log Jacobian determinants.
4. Compute the log-probability of the data according to the final data transformation and the log Jacobian determinants.
5. Compute the loss as the negative data log-probability and backpropagate to update the parameters for $s(\cdot)$ and $t(\cdot)$.

During inference time, the steps are:

1. Sample latent vectors from the prior distribution.
2. Pass the vectors to the reverse coupling layers and apply the inverse logit normalization.

Question 3.7

The detailed implementation of the RealNVP model based on Dinh et al. [2016] can be referred in `a3_nf_template.py`. The model is correctly implemented and passes both the unit tests.

Question 3.8

The training and validation performance in bits per dimension is shown in Figure 7. As can be inferred from the graph, the model is able to reach below 1.85 bits per dimension in 17 epochs.

Besides, Figure 8 shows 25 sample images generated by the network respectively at epoch 1, epoch 20 and epoch 40, from which we can observe an improvement as the training processes.

4 Conclusion

Question 4.1

In this project, three type of generative models (i.e. VAE, GAN and RealNVP) are implemented, discussed and compared on the MNIST dataset. Firstly, for GAN, there is no explicit probability

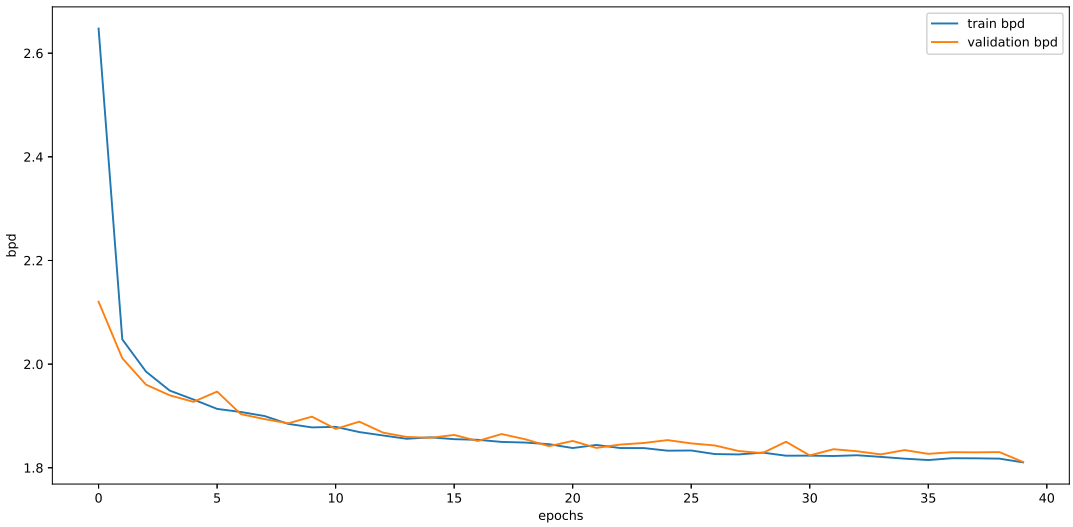


Figure 7: *The training and validation performance in bits per dimension of the RealNVP model.*

(a) $Epoch = 1$. **(b)** $Epoch = 20$. **(c)** $Epoch = 40$.

Figure 8: *Sample images generated by the RealNVP during different phases of training.*

distribution of the data, and the model is trained based on a zero-sum game of two players (i.e. discriminator and generator). Among these three models, GAN achieves the best generation quality, however, as the model commonly suffers from the stability problem, it also requires the most number of training epochs and parameter tuning. On the other hand, for VAE, it provides an approximation to an intractable probability distribution of the data, thus, to train the model, we need to instead maximize the evidence lower bound. VAE is in general easy to train, but the generated images are either too sharp or blurry. However, the model has the advantage that it is able to compress data for extracting useful information. In terms of the flow-based models like RealNVP, unlike GAN and VAE, they model an explicit probability distribution of data by applying a sequence of invertible transformations. Flow-based models are heavier to train than VAE, but the image generation quality is better.

References

Carl Doersch. Tutorial on Variational Autoencoders. *arXiv e-prints*, art. arXiv:1606.05908, Jun 2016.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, art. arXiv:1312.6114, Dec 2013.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv e-prints*,

art. arXiv:1406.2661, Jun 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv e-prints*, art. arXiv:1605.08803, May 2016.

Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator. *arXiv e-prints*, art. arXiv:1306.0186, Jun 2013.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. *arXiv e-prints*, art. arXiv:1902.00275, Feb 2019.