Assignment 3 Deep Generative Models

Qiao Ren UvAnetID: 11828668 university of amsterdam Dec, 2020

Question 1.1

Step1 draw samples z_n from normal distribution: $\mathrm{p}(z_n) = N(0, I_D)$

Step2 compute: $\mathrm{de}coder(z_n) = f_\theta(z_n)$. the result of decoder will be the mean of the probability distribution of each pixel, in the next step.

Step3 Each pixel is drawn from a bernoulli distribution. The mean of the bernoulli distribution is $f_\theta(z_n)_\mathrm{m}$. So the probability distribution of a single pixel is:

$$\mathrm{p}(x_n^{(m)}) = Bern(x_n^{(m)} \mid f_\theta(z_n)_\mathrm{m})$$

Step4 For the nth img, it has M pixels in total. So the probability distribution of the nth img is the joint probability of all the pixels.

$$\mathrm{p}(x_n \mid z_n) = \prod_{m=1}^{M} Bern(x_n^{(m)} \mid f_\theta(z_n)_\mathrm{m})$$

Question 1.2

Monte-Carlo integration is inefficient. Because:

1) As shown in Figure 2, the region of p(z|x) only covers a small part of the region p(z). When we draw samples z_i from p(z), there are lots of samples in the region of p(z) but out of the region of p(z|x). For these samples, $p(x_n|z_i)$ is almost zero. Therefore they don't contribute to computing log $p(x_n)$. It means that we draw a large amount of useless samples z_i.

2) Monte-Carlo integration also suffers from the curse of dimensionality. When the dimensionality of z is high, each z_i is high dimentional. The computation cost is huge. We have to draw a huge amount of samples to cover the whole space of z. However it is infeasible to do so.

Question 1.3

Example 1	$q = \mathrm{N}(\mu_q, {\sigma_q}^2) = \mathrm{N}(1, \quad 10)$ $p = \mathrm{N}(\mu_p, {\sigma_p}^2) = \mathrm{N}(1.001, 10.001)$	$\mathrm{D}_{KL}(q \| p)$ is very small
Example 2	$q = \mathrm{N}(\mu_q, {\sigma_q}^2) = \mathrm{N}(1, \quad 10)$ $p = \mathrm{N}(\mu_p, {\sigma_p}^2) = \mathrm{N}(8000, 8000)$	$\mathrm{D}_{KL}(q \| p)$ is very large

Question 1.4

$$\log p(x_n) - KL(q(Z \mid x_n) \| p(Z \mid x_n)) = ELBO$$
$$\log p(x_n) - KL(q(Z \mid x_n) \| p(Z \mid x_n)) = \mathrm{E}_{q(z_n|x_n)}[\log p(x_n \mid z_n)] - KL(q(z \mid x_n) \| p(z))$$

We have computed that: $\log(\mathrm{p}(x_n)) = \log(E[\frac{p(z_n)}{q(z_n \mid x_n)} p(x_n \mid z_n])$

We apply the Jensen's Inequality. Because log is a concave function. So it satisfies Jensen's Inequality.

$$\mathrm{h}(E[function]) \ge E[h(function)]$$
$$\Rightarrow \log(E[\frac{p(z_n)}{q(z_n \mid x_n)} p(x_n \mid z_n]) \ge E[\log(\frac{p(z_n)}{q(z_n \mid x_n)} p(x_n \mid z_n)]$$
$$\Rightarrow \log(E[\frac{p(z_n)}{q(z_n \mid x_n)} p(x_n \mid z_n]) \ge E[\log(p(x_n \mid z_n)] + E[\log(\frac{p(z_n)}{q(z_n \mid x_n)}]$$
$$\Rightarrow \log(E[\frac{p(z_n)}{q(z_n \mid x_n)} p(x_n \mid z_n]) \ge E[\log(p(x_n \mid z_n)] - \mathrm{KL}(q(z \mid x_n) \| p(z))$$
$$\Rightarrow \log(p(x_n)) \ge E[\log(p(x_n \mid z_n)] - \mathrm{KL}(q(z \mid x_n) \| p(z))$$

We must optimize the lower bound, instead of optimizing $\log p(x_n)$ directly. Because:

$$\mathrm{p}(x) = \int p(z) p(x \mid z) dz$$

In p(x), the integral is intractable, especially when z has high dimension.

Question 1.5

$$\log p(x_n) - KL(q(Z \mid x_n) \| p(Z \mid x_n)) = ELBO$$

When the lower bound is pushed up, two things can happen:

1) Log likelihood $\log p(x_n)$ is maximized.

2) $KL(q(Z \mid x_n) \| p(Z \mid x_n))$ is minimized. This means that the KL divergence between approximated posterior and true posterior become smaller. The learned posterior approaches to the true posterior. Ideally, when the learned latent variable is perfectly equal to the true latent variable, $KL(q(Z \mid x_n) \| p(Z \mid x_n))$=0, $\log p(x_n) = ELBO$

Question 1.6

Reconstruction loss: $L_n^{rec} = -\mathrm{E}_{q_\theta(z_n|x_n)}[\log p(x_n \mid z_n)]$ it computes the negative log likelihood of the reconstructed data, with the given latent variable. In the equation, $q_\phi(z \mid x_n)$ is the approximated posterior. We firstly sample z from this approximated posterior. Then we reconstruct x_n . The log likelihood $\log p_\theta(x_n \mid z)$ assigns high probability to the reconstruction. When minimizing the loss function, the smaller the loss L_n^{recon} is, the larger the log likelihood $E_{q_\phi(z|x_n)}[\log p_\theta(x_n \mid z)]$ is. We want the likelihood $p_\theta(x_n \mid z)$ to be as large as possible, which means the reconstructed data x_n has a high possibility to be generated from z.

Regularization loss: $\mathrm{L}_n^{rec} = \mathrm{D}_{KL}(q_\phi(z \mid x_n) \| p_\theta(z))$ it computes the KL divergence (dissimilarity) between the approximated posterior $q_\phi(z \mid x_n)$ and the prior $p_\theta(z)$. We want L_n^{rec} to be as small as possible. It means that we force the approximated posterior (or the encoder) $q_\phi(z \mid x_n)$ to be similar as the prior. The regularization loss is used to avoid overfitting.

Question 1.7

$$\mathrm{L}_{\mathrm{n}}^{\mathrm{recon}} = -E_{q_\phi(z|x_n)}[\log(p_\theta(x_n \mid z))]$$
$$= -\log(p_\theta(x_n \mid z))$$
$$= -\log\prod_{m=1}^{M}(Bern(x_n^{(m)} \mid f_\theta(z_n)_m)$$
$$= -\sum_{m=1}^{M}\log(Bern(x_n^{(m)} \mid f_\theta(z_n)_m))$$
$$= -\sum_{m=1}^{M}\log(f_\theta(z_n)_m{}^{x_n^{(m)}} \bullet (1-f_\theta(z_n)_m)^{1-x_n^{(m)}})$$
$$= -\sum_{m=1}^{M}\{\log(f_\theta(z_n)_m{}^{x_n^{(m)}}) + \log(1-f_\theta(z_n)_m)^{1-x_n^{(m)}})\}$$
$$= -\sum_{m=1}^{M}\{x_n^{(m)}\log(f_\theta(z_n)_m) + (1-x_n^{(m)})\log(1-f_\theta(z_n)_m)\}$$

$$\mathrm{L}_{\mathrm{n}}^{\mathrm{reg}} = \mathrm{D}_{KL}(q_\phi(z \mid x_n) \,\|\, p_\theta(z))$$
$$= \mathrm{D}_{KL}(N(z_n \mid \mu_\phi(x_n), diag(\Sigma_\phi(x_n)) \,\|\, N(z_n \mid 0, I_{\mathrm{D}}))$$
$$= \sum_{d}^{D}\frac{1}{2}\{\mathrm{var}_q + mean_q^2 - 1 - \log(\mathrm{var}_q)\}$$
$$= \frac{1}{2}\sum_{d}^{D}\{diag(\Sigma_\phi(x_n)) + \mu_\phi(x_n)^T\mu_\phi(x_n) - 1 - \log(diag(\Sigma_\phi(x_n))\}$$
$$= \frac{1}{2}\{\sum_{d}^{D}diag(\Sigma_\phi(x_n)) + \sum_{d}^{D}\mu_\phi(x_n)^T\mu_\phi(x_n) - \sum_{d}^{D}1 - \sum_{d}^{D}\log(diag(\Sigma_\phi(x_n)))\}$$
$$= \frac{1}{2}\{Tr(diag(\Sigma_\phi(x_n))) + \sum_{d}^{D}\mu_\phi(x_n)^T\mu_\phi(x_n) - D - \sum_{d}^{D}\log(diag(\Sigma_\phi(x_n)))\}$$

$$\mathrm{L} = \frac{1}{\mathrm{N}} \sum_{n=1}^{\mathrm{N}} (\mathrm{L}_{\mathrm{n}}^{\mathrm{recon}} + L_n^{reg})$$
$$= \frac{1}{\mathrm{N}} \sum_{n=1}^{\mathrm{N}} \{ -\sum_{m=1}^{M} (x_n^{(m)} \log(f_\theta(z_n)_m) + (1 - x_n^{(m)}) \log(1 - f_\theta(z_n)_m)) +$$
$$\frac{1}{2} \sum_{d}^{D} (diag(\Sigma_\phi(x_n)) + \mu_\phi(x_n)^T \mu_\phi(x_n) - 1 - \log(diag(\Sigma_\phi(x_n)))\}$$

D: dimention of latent space

N: total amount of images

M: total amount of pixels in each image

X_n: the nth image

Question 1.9 Reparameterization trick

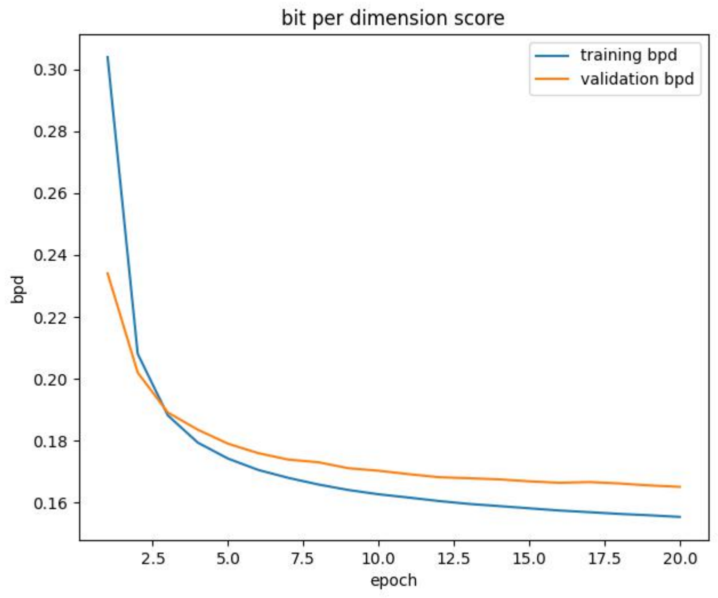
The problem is that we want to compute the gradient of loss with respect to θ and ϕ. However, it is impossible to compute the gradient. Because sampling process in the forward pass is stochastic. So it is not differentiable. Therefore, we use the reparameterization trick.

Reparameterization trick is: we add noise to the standard deviation of the latent variable. The noise is drawn from normal distribution N(0,1). After adding noise, latent variable z is written as:

$$\mathrm{z} = \mu_z + \varepsilon \sigma_z$$
$$\varepsilon \in \mathrm{N}(0,1), \mathrm{z} \in \mathrm{N}(\mu_z, \sigma_z)$$

Reparameterization trick allows us to compute the gradient in back propagation. Because it disconnects the random sampling by using the external noise epsilon. In backpropagation, the gradients flow through deterministic functions: mean and standard deviation of z. Randomness has been moved to the noise.

Question 1.9



Architecture

Encoder: its first layer is Linear (in=784, out=512). Its second layer is Relu. Based on this output, it splits into two layers: one linear layer for mean and another linear layer for log variance. The mean and log var both have the same shape: batch size * dimensionality of latent space.

Decoder: It has a linear layer (in=z_dim, out=512), a relu, and a linear layer (in=512, out=784). Output of decoder is reconstructed images which has shape: batch size *channel*H*W

Question 2.1

A) $\mathrm{E}_{p_{data}(x)}[\log D(X)]$ is the expectation of log probability that descriminator classifying x. x is drawn from the distribution of real data. If descriminator classifies real data as real, then D(x)=1.

$\mathrm{E}_{p_z(x)}[\log(1-D(G(X)))]$ is the expectation of log probability that the descriminator classifying the fake x. fake x is G(x), which is generated by the generator function G. If descriminator classifies fake data as real, then D(G(x))=1. If descriminator classifies fake data as fake, then D(G(x))=0.

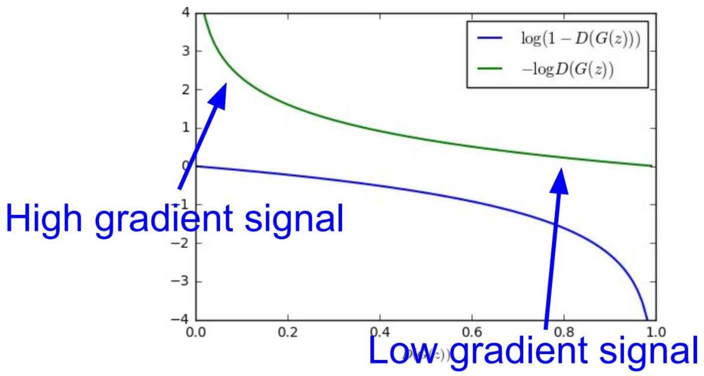
B) After training has converged: the learned generator distribution is identical to p_data. $p_{\text{data}} = p_g$. So the optimal discriminator is 1/2. Descriminator and generator reaches equilibrium, when training is converged.

$$D^{optimal}(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} = \frac{1}{2}$$

$$V(D^{\text{optimal}}, G) = \log(\frac{1}{2}) + \log(1 - \frac{1}{2}) = -\log 4$$

Question 2.2

Problem: When sample is likely fake, we want to learn from it to improve the generator. However, log(1-D(G(z))) is too flat in the region where D(G(z)) is close to 0. It means that log(1-D(G(z))) has low gradient signal when the discriminator classifies data as fake.



Solution: Instead of minimizing likelihood of discriminator being correct,we maximize likelihood of discriminator being wrong. We assign higher gradient signal for bad samples. We convert log(1-D(G(z))) to log(D(G(z))):

$$\min E[\log(1 - D(G(z)))] \Leftrightarrow \max E[\log(D(G(z)))]$$

Now the gradient has high signal in the region where D(G(z)) is close to zero.

Question 2.3

Architecture:

generator	discriminator
It contains 4 hidden modules. Each module contains: a linear layer, a batchnorm layer, a leakyrelu layer and a dropout layer. In the end, it has a linear layer and a tanh layer.	It contains 2 modules. Each module contains:a linear layer, a leakyrelu layer and a dropout layer. In the end, it has a linear layer.

Question 2.5

One problem is: Batch-normalization causes strong intra-batch correlation. A lot of samples generated within the batch tend to be similar. Because the activation between different minibatches depends on each other. So the generations (product of activation) depends on each other. Solution: virtual batch normalization. We select a reference batch which is fixed during training. We append the reference batch to regular mini-batch. We combine it with the statistics of the current batch.

Question 3.1

$\mathrm{p_x}(x) = p(z)\lvert\frac{dz}{dx}\rvert$ $= \frac{1}{b-a}\lvert\frac{d}{dx}f(x)\rvert$ $= \frac{1}{b-a}\lvert\frac{d}{dx}x^3\rvert$ $= \frac{1}{b-a}3x^2$ $z = f(x) = x^3 \Rightarrow x = f(z)^{-1} = z^{\frac{1}{3}}$ $z \in (a,b) \Rightarrow x \in (a^{\frac{1}{3}}, b^{\frac{1}{3}})$	$verify: \int_{-\infty}^{\infty} p_x(x)dx = 1$ $\int_{-\infty}^{\infty} p_x(x)dx$ $= \int_{-\infty}^{\infty}\frac{1}{b-a}3x^2dx$ $= \frac{3}{b-a}\int_{-\infty}^{\infty}x^2dx$ $= \frac{3}{b-a}\int_{a^{\frac{1}{3}}}^{b^{\frac{1}{3}}}x^2dx$ $= \frac{3}{b-a}(\frac{1}{3}x^3\Big\rvert_{x=b^{\frac{1}{3}}} - \frac{1}{3}x^3\Big\rvert_{x=a^{\frac{1}{3}}})$ $= \frac{3}{b-a}(\frac{1}{3}(b^{\frac{1}{3}})^3 - \frac{1}{3}(a^{\frac{1}{3}})^3)$ $= \frac{1}{b-a}(b-a)$ $= 1$

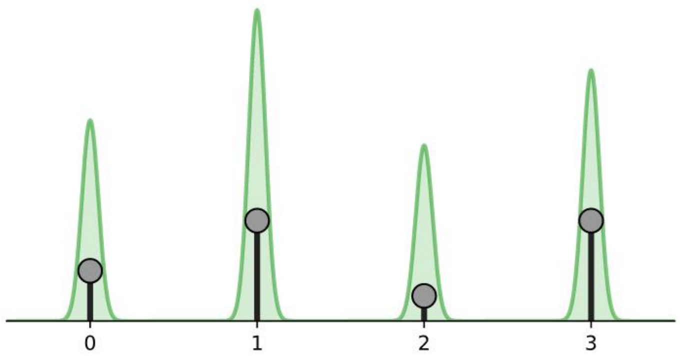
3.2

a) h_l must be 1) differentiable 2) invertible 3) smooth 4) easy to compute the inverse of h_l 5) easy to compute the jacobian determinant . Because when the number of dimension in h_l and h_{l-1} is high, $\left\lvert\frac{\mathrm{dh_l}}{\mathrm{dh_{l-1}}}\right\rvert$ will have high dimension as well. Otherwise, it is hard to compute $\log p_x(x)$.

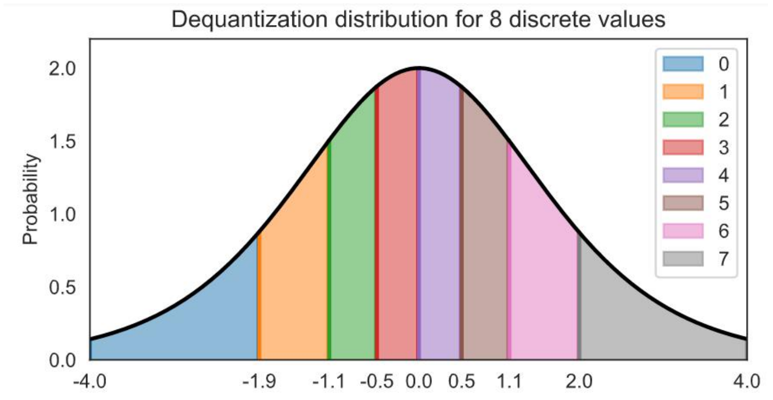
b) The first optimization issue: very likely, h_l can not satisfy all the above requirements. The second optimization issue: in equation 18, inverse of a function and jacobian determinant are very hard to compute.

3.3

A) When variables are discrete, like images, instead of continuous, the learned distribution looks like the following image. The black points represent the discrete points, and the green volume the density modeled by a normalizing flow in continuous space. For x=0,1,2,3, they have infinite high likelihood. Because their interval is 0 in the continuous space. Because $\int p(x)dx = 1$ .The learned density does give any useful information about the distribution among the discrete points.



To fix it, we use dequantization. Dequantized v =x+u, where $u \in [0,1)^D$. It makes the probability of x to become smooth. As shown in the following image, the shape of p(x=0) is smoothly connected with p(x=1).



c) the steps to train a general flow based model, evaluate it after training, sample from it.

Step 1 assume a distribution function for p(z)

Step 2 go through forward pass, from x to z

Step 3 compute loss between the p(z) that we want and the p(z) that the model computes

Step 4 go backward and update the parameters of f(x).

Step 5 repeat step 3 and 4 untill the p(z) reaches our goal

Step 6 now we are able to compute the likelihood p(x). because we have learned the function f(x). we go back from z to x, which is the inverse of functin f, then we compute p(x)

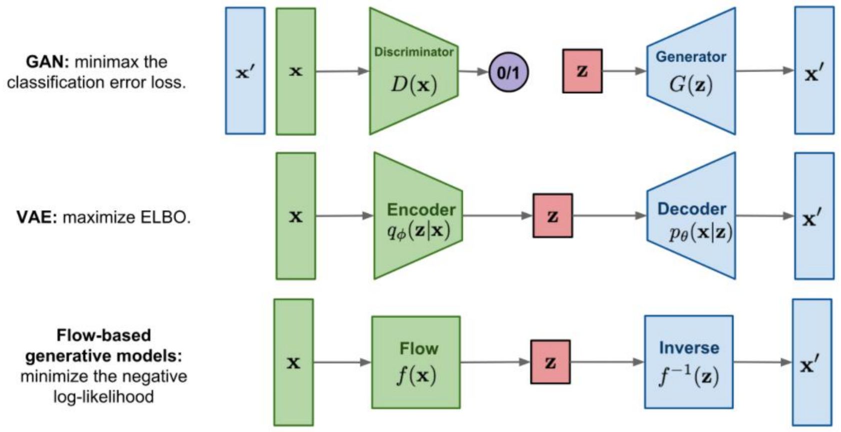
Conclusion

GAN uses minimax game to maximize the likelihood. Generator aims at fooling the descriminator by generating data which are fake but looks real. Discriminator aims at correctly distinguishing real and fake data. Latent variable is the source of noise.

VAE uses lower bound to maximize the likelihood. VAE uses latent variable z to capture the feature of input variables. Then it reconstructs new data based on z. It encode x to z, then it decode z to x. It is like doing zipping. The dimension is reduced through encoder.

Flow model: Each input data has a corresponding latent variable. The dimension of z is the same as the dimension of x.

VAE computes an approximated likelihood. Flow Model computes the exact likelihood directly. GAN computes imlicity density while VAE computes explicity density.



GAN: minimax the classification error loss.
$\mathbf{x}'$
$\mathbf{x}$
Discriminator $D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator $G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize ELBO.
$\mathbf{x}$
Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Flow-based generative models: minimize the negative log-likelihood
$\mathbf{x}$
Flow $f(\mathbf{x})$
$\mathbf{z}$
Inverse $f^{-1}(\mathbf{z})$
$\mathbf{x}'$