Assignment 3 Deep Generative Models

Xinyi Chen
University of Amsterdam – Deep Learning Course
xinyi.chen@student.uva.nl

1 Variational Auto Encoders

1.1 1.1

We first sample z_n from the standard-normal distribution $\mathcal{N}(0, I_D)$. Once we have z_n, we can get $f_\theta(zn))$ using the neural network parameterized by θ. Then, we can sample $x_n^{(m)}$ from the Bernoulli distributions $Bern(x_n^{(m)}|f_\theta(zn)_m)$.

1.2 1.2

By randomly sample from $p(z_n)$, it is very likely to sample z_n from low density regions, which will lead to small $p(x_n|z_n)$. This will make inaccurate estimation for $E_{p(z_n)[p(x_n,z_n)]}$. And there will be high variance in estimators when we sample in this way for multiple times.

1.3 1.3

When $(\mu_q, \mu_p, \sigma_q^2, \sigma_p^2)$ is (0, 0, 0.1, 0,15), KL(q,p) = 0.505, KL-divergence is very small. When $(\mu_q, \mu_p, \sigma_q^2, \sigma_p^2)$ is (0, 0, 100, 0,15), KL(q,p) = 4997.5, KL-divergence is very large.

1.4 1.4

$KL(q(Z|x_n)||p(Z||x_n))$ is always positive, so $logp(x_n)$ is always greater than $logp(x_n) - KL(q(Z|x_n)||p(Z||x_n))$. The right hand side of Equation 14 is a lower bound on the $logp(x_n)$. Using the lower-bound instead of optimizing the logp(x_n)$because$p(Z||x_n) is intractable. So we cannot calculate the $KL(q(Z|x_n)||p(Z||x_n))$.

1.5 1.5

1. When the lower bound is pushed up, the $KL(q(Z|x_n)||p(Z|x_n))$ will be smaller. This means there is smaller difference between the approximate probability distribution $q(Z|x_n)$ and true distribution $p_{theta}(z|x_n)$. We have a better latent representation.

2. When the lower bound is pushed up, the gap between the log likelihood $logE_{q(z_n|x_n)}$ and $Eq(z_n|x_n)log$ is smaller, which means that we have a better density model.

1.6 1.6

The goal of L_n^{recon} is maximize the log likelihood of $logp_\theta(x_n|Z)$, which enables the model to reconstruct x when given latent variable Z, so it's reconstruction loss. The L_n^{reg} term measures the distance between approximate probability distribution $q_\Phi(Z|x_n)$ and true distribution. When the L_n^{reg} is small, $q_\Phi(Z|x_n)$ is closer to the real distribution $p_\theta(Z)$, which plays a similar role as regularization function.

1.7 1.7

1. The encoder take in the input x_n and compute the mean $\mu_\Phi(x_n)$ and covaraince $\Sigma_\Phi(x_n)$ of x_n

2. Sample z_n $q_\Phi(z_n|x)$ from normal distribution $q_\Phi(z_n|x) = (z_n|\mu_\Phi(x_n), diag(\Sigma_\Phi(x_n)))$.

3. Compute $p_\theta(x_n|z_n)$ using $p_\theta(x_n|z_n) = \prod_{m=1}^{M} Bern(x_n^(m)|f_\theta(z_n)_m)$.

4.Compute $L_n^{recon} = -E_{q_\Phi(z|x_n)}[logp_\theta(x_n|Z)] \approx -logp_\theta(x_n|z_n)$. Here we approximate the log-likelihood using Monte-Carlo Integration.

5. Compute $L_n^{reg} = D_{KL}(q_\Phi(Z|x_n)||p_\theta(Z)) = \frac{1}{2}(\Sigma_\Phi(x_n)^T\Sigma_\Phi(x_n) + \mu_\Phi(x_n)^T\mu_\Phi(x_n) - D - log(\Sigma_\Phi(x_n)^T\Sigma_\Phi(x_n))$

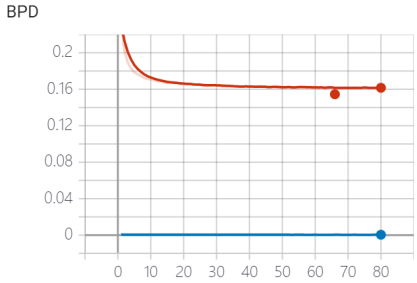
6.Compute $L = L_n^{recon} + L_n^{reg}$

1.8 1.8

From question 1.7, to compute $\nabla_\Phi L$, we need to compute the gradients of z_n $q_\Phi(z_n|x)$. But sampling operation is not differentiable, so it is impossible to compute the gradients. Using the reparameterization trick, we rewrite z_n $q_\Phi(z_n|x)$ to $z_n = \mu_z + \varepsilon\sigma_z, \varepsilon$ $N(0,1)$. In this way, the variables μ_z and σ_z that depends on parameters become deterministic. The only random variable ε does not depends on parameters. So now the trick makes it possible compute $\nabla_\Phi L$.

1.9 1.9

Both the encoder and decoder are 1-layer mlp networks with 512 hidden units. Each layer is sequence of Linear, ReLu modules. To train this model, a learning rate of 1e-3 and a batch size of 128 are used. The optimizer is Adam. The model is trained for 80 epoches. The training and validation loss is shown in 1a.



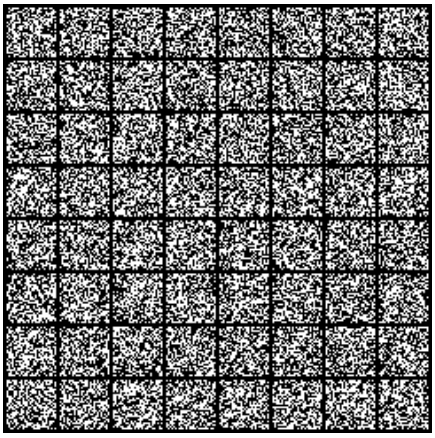
(a) BDP of the model, blue: training loss, orange: validation loss, the final test score is 0.16

1.10 1.10

As shown in Fig.2a, Fig.2b, Fig.2c, the generated images become less noisy and the figures become more recognizable as training epoch increases.

1.11 1.11

The manifold pictures are shown in Fig.3a. The similar numbers appear close to each other in the picture, which suggests that similar numbers have similar latent variables.



(a) samples at 0 epoches for VAE

(b) ssamples at 10 epoches for VAE

(c) samples at 80 epoches for VAE

2 Generative Adversarial Networks

2.1 2.1

2.1.1 a

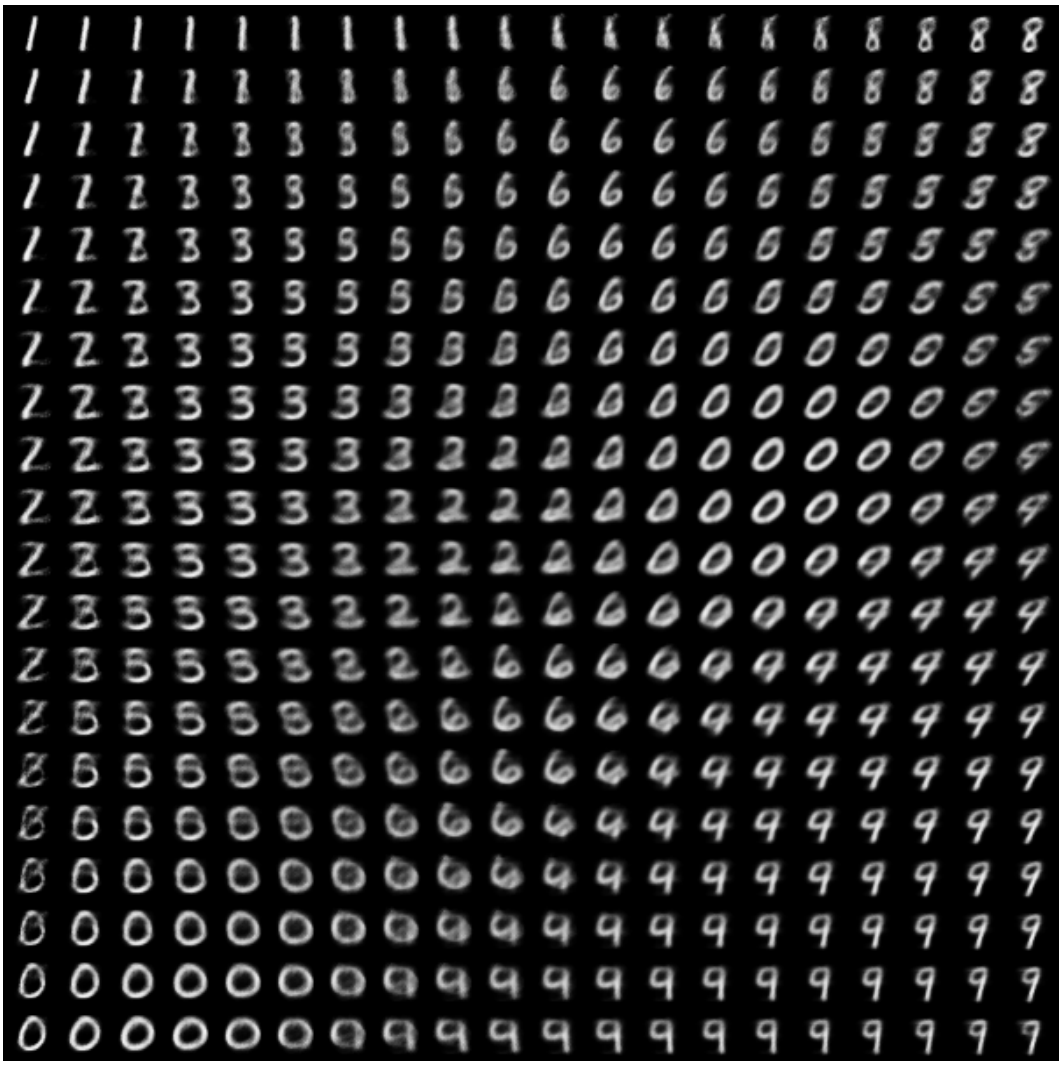
The first term $E_{p_data(x)}[logD(X)]$ maximizes the log-likelihood of the discriminator correctly distinguish real data. The second term $E_{p_z(z)}[log(1 - D(G(Z))]$ maximizes the log-likelihood of the discriminator correctly distinguish synthesized data generated by the generator.

2.1.2 b

After training has converged, the D(x) should be 0.5, which means that the discriminator can no longer determine whether x is real or fake data for any given x. $V(D, G) = log0.5 + log(1 - 0.5) = -2log2$.

2.2 2.2

The loss function to optimize generator is log(1-D(G(Z))). Early on during training, if the discriminator is good, the loss will be close to zero. The gradients will also fall close to zero, making the learning become very slow or no learning. This causes vanishing gradient problem for training the GAN. To solve this problem, we can replace minimizing the logD(1-G(Z)) by maximizing the D(G(Z)) in the loss function [1].



(a) manifold of the vae

2.3 2.3

The generator is a mlp network with 3 hidden layers. The three layers have 128, 256, 512 hidden units respectively. Each layer is sequence of Linear, Dropout and LeakyReLU modules. The output layer uses Tanh to center its output data. The discriminator is a mlp network with 2 hidden layers. Their hidden units are 512 and 256. Each layer has the same structure with the generator. The model is trained for 250 epoches.

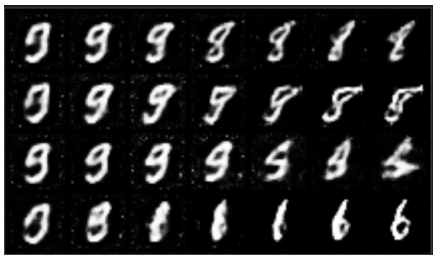
As can be seen in Fig.4a, samples generated at epoch 10 have many noise and it's hard to recognize the figures in the image. As we increase training epoch, the samples at 250 epoch Fig.4b have less noise in its generated images and the figures are recognizable.

2.4 2.4

After training for 250 epoches, using 5 interpolation steps for each of the 4 examples. The image is shown in Fig.5a.

(a) samples at 10 epoches

(b) samples at 250 epoches



(a) BDP of the model, blue: training loss, orange: validation loss, the final test score is 0.16

2.5 2.5

Mode collapse is a common problem in vanilla GANs optimization. The problem occurs when discriminator converges but the generator only generates the most likely points but ignores other modes. It happens because the model underestimates the variance in its generated samples. To solve this problem, we need to make the generated data more diverse. First, classify each sample in the mini-batch. Then, compare samples with other samples in the mini-batch. Penalized the model if samples are similar.

2.6 Generative Normalizing Flows

2.7 3.1

$$p_x(x) = p_z(x^3) * |3x^2| = \begin{cases} \frac{3x^2}{b-a} & x \in [a^{\frac{1}{3}}, b^{\frac{1}{3}}] \\ 0 & otherwise \end{cases}$$

2.8 3.2

2.8.1 a

The h_l should have the same dimension as h_{l-1} and $\frac{dh_l}{h_{l-1}}$ should not be 0.

2.8.2 b

While doing optimization, computing the Jacobian of functions with high-dimensional domain and codomain and computing the determinants of large matrices are very computationally expensive[2]. When using the neural network to sample data points, we need to compute the Jacobian of functions and latent representation of matrices, which might be very computationally expensive. It would be difficult to do with limited computational resource.

2.9 3.3

2.9.1 a

The learned distribution will have -peak probabilities on integer values only while having no volume on any other point. To fix this problem, we can add continuous noise to the input variables.

2.9.2 b

To train a general flow-based model, first take in the input data and apply dequantization to the input data if necessary. Then use the encoder to perform transformation of the input data. The output of encoder is the latent representation z and log determinant Jacobian of the transformations. Using the latent represent z, we can compute the log likehood of the input data and the loss of the model. Finally, use the loss to perform backpropagration to update the parameters of the encoder. Repeat the train step until it meets the stop training criteria. During inference, sample latent represent z from a prior distribution. Then use the inverse transformation to transform the latent represent z to new examples that are similar to the input data for training.

3 Conclusion

In this assignment, we go through theoretical details on VAEs, GANs and Flow-based models. The VAEs learn the distribution of input data and sample a random latent represenation z to generate similar images. In VAEs, we use ELBO to approximate intractable probability distribution of the input data. The GANs use the minimax game to train its generator and discriminator. It learns implicit density and doesn't use latent representation. The GANs have better performance in generating images than VAE because it doesn't use some method to make approximation. The Flow-based models explicitly learns the data distribution p(x). And it learns a 1-1 mapping from input data x to latent representation z. Because of that, it can use the inverse transformation for inference without using additional inference model.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.