# Deep learning assignment 3

**Changxin Miao**
Changxin.Miao@student.uva.nl

## 1 Variational Auto Encoders

### 1.1 Latent Variable Models

#### 1.1.1 Question 1.1

With regards to VAE and other two methods, they are try to model a latent variable z given the data x. However, distribution of z could be any form in VAE. In pPCA and VAE, it is always assumed to be gaussian distribution. Since pPCA uses the isotropic noise covariance $\sigma^2 I$, it is covariant under rotation of the original data axes, while factor analysis is covariant under component-wise rescaling ($\Phi = diag(\phi_1, \phi_2, \phi_3, \ldots)$). Furthermore, In factor analysis, neither of the factors found a two-factor model will be the same as the one found in single-factor model. In pPCA, the principle axes could be found incrementally.

### 1.2 Question 1.2

Ancestral sample is a sampling method based on Bayesian net. For different variables, we create a graph which indicates the dependent relationship between them. Them we sample each variables based on the directed edge, which makes sampled variables conditioned on previous variables.
In our example, it could be described as following process,

1. Sample $z_n$ from $\mathcal{N}(0, I_D)$
2. Calculate $f_\theta(z_n)$ based on initiated parameters
3. Sample $y_m \sim Bern(f_\theta(z_n))$, this could be achieved by some rule-based methods
4. Arrange previously sampled pixels $y_m$ for image $x_1$, for $n = 1, 2, \ldots, n$, we repeat above procedures.

### 1.3 Question 1.3

The model $P(z)$, original is represented as $P(z;\theta)$, tries to produces samples that are similar to training samples $X$ while avoids producing dissimilar samples. Even though z follows a normal distribution, due to the strong modeling ability, the neural network could generate different distributions. By having a Gaussian distribution $P(z)$, gradient decent could be implemented to make sure that $f(z;\theta)$ approaches x for some z. From the graphical model below, we could clearly see that $\theta$ will be a fixed parameter that determines the distribution of x.

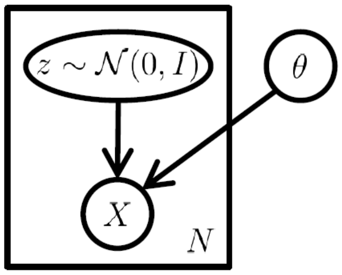

Figure 1: plate notation of VAE

### 1.4 Question 1.4

(a

$$\begin{aligned} logp(x_n) &= E[log(x_n)] \\ &= E[log[(b-a)\frac{1}{N}log\sum_{i=1}^{N} p(x_n|z_n)p(z_n)]] \\ &= log[(b-a)\frac{1}{N}p(X|Z)p(Z)] \\ &= log[(b-a)\frac{1}{N}\sum_{k=1}^{N}\int_a^b p(x_i|z_k)p(z_k)] \end{aligned}$$

If we set a as 0 and b as $\inf$, it is obvious that we could obtain $\frac{1}{K}\sum_0^N p(x_n|z_k)$ as an approximation of $p(x_n)$. Therefore, if we sample enough $p(x_n|z_k)$, we could use the average of it as approximation of $p(x_n)$.
(b) It is inefficient because we need to generate a significant amount of samples so that $p(x_n|z_k) \neq 0$. As we could observe from figure 2, with the dimension of 2, the distribution of data is 2D. $p(x|z)$ is also focused on the 2D-dimensional sphere and it is unlikely to get samples out of that. However, as the dimension grows, the probability of $p(x|z)$ is not close to the k-dimensional sphere anymore, then it becomes harder to get non-zero samples. In the meanwhile, we could imaging the computational power could grow exponentially along with the growth of dimension. Thus it costs a lot of time and resources to generate the data.

### 1.5 Question 1.5

(a) Small KL-divergence: q = $\mathcal{N}$(1,0.5)
Large KL-divergence: q = $\mathcal{N}$(15,10)
(b)

$$\begin{aligned} D_{KL}(q\|p) &= log\frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} \\ &= log\frac{1}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - 0)^2}{2 * 1^2} \\ &= log\frac{1}{\sigma_q} + \frac{\sigma_q^2 + \mu_q^2}{2} \end{aligned}$$

### 1.6 Question 1.6

We know that

$$logp(x_n) - D_{KL}(q(z|x_n)\|p(z|x_n)) = E_{q(z|x_n)}[logp(x_n|z)] - D_{KL}(q(x|x_n)\|p(z))$$

It is obvious that $D_{KL}(q(z|x_n)\|p(z|x_n))$ stays positive all the time. Hence, the right-hand side will be the lower bound to P(x)

$$logp(x_n) \geq E_{q(z|x_n)}[logp(x_n|z)] - D_{KL}(q(x|x_n)\|p(z))$$

## 1.7 Question 1.7

The $p(z|x)$ is not easily tractable and the quantity, thus we could not evaluate or differentiate the marginal likelihood. $p(x)$ is very difficult to calculate. After the reconstruction, the lower-bound will be easier to calculate, therefore we choose to optimize the lower bound.

## 1.8 Question 1.8

The $logp(x)$ is maximized while $D_{KL}(q(z|x_n)\|p(z|x_n))$ is minimized. If we could use a high-capacity $q(z|x)$, then $q(x|z)$ could actually match $p(x|z)$. During the training process, the KL-divergence will change towards 0. Then we could optimize $logp(x)$ directly.

## 1.9 Question 1.9

Reconstruction: Since we make use of latent variables sampled from the $q_\theta(z|x_n)$ distribution to construct $x_n$ $\mathcal{L}_n^{recon} = -E_{q_\phi(z|x_n)}[logp_\theta(x_n|z)]$ could be regarded as the expected log-likelihood of reconstructing $x_n$.
Regularization: It nudges the approximate posterior to be close to the prior $p_\theta(z)$. We could also view it as a sparsity regulation in sparse autoencoders

## 1.10 Question 1.10

$$\begin{aligned}
z_n &\sim \mathcal{N}(0, I_D)\\
x_n &\sim p_X(f_\theta(z_n))\\
p(z_n) &= \mathcal{N}(0, I_D)\\
p(x_n|z_n) &= \prod_{m=1}^{M} Bern(x_n^{(m)}|f_\theta(z_n)_m)\\
\mathcal{L}_n^{recon} &= -E_{q_\phi(z|x_n)}[logp_\theta(x_n|z)]\\
&= -\int log(p_\theta(x_n|z))q_\phi(z|x_n)dz\\
&= -\int (\sum_{m=1}^{M}[x_n^{(m)}log(f_\theta(z_n)_m) + (1 - x_n^{(m)}log(1 - f_\theta(z_n)_m))])\mathcal{N}(z_n|\mu_\phi(x_n), diag(\sum_\phi(x_n))d_z\\
\mathcal{L}_n^{reg} &= D_{KL}(q_\phi(z|x_n)\|p_\theta(z))\\
&= D_{KL}(\mathcal{N}(\mu(X), \Sigma(X))\|\mathcal{N}(0, I_D))\\
&= \frac{1}{2}(tr(\Sigma(X)) + (\mu(X))^T(\mu(X)) - K + log(det\Sigma(X)))
\end{aligned}$$

## 1.11 Question 1.11

(a) If we use just one sample to approximate the expectation $E_{q_\phi(z|x_n)}[logp_\theta(x_n|z)]$, the expression could be simplified to compute the gradient of $logp(x_n|z)$. However, here we should notice that the expectation does not only depend on the parameter of p but also parameter of q. Hence, we need to calculate $\nabla_\phi\mathcal{L}$.

(b)We know that it is important calculate $\nabla_\phi\mathcal{L}$. In order to do that, we could back-propogate the error through a layer that samples z from $q_\phi(z|x_n)$, which is a non-continuous function and has no gradient. Therefore, we could try to move the sampling to an input layer.

(c)"Reparameterization trick" is to move the sampling to an input layer. More specifically, we can sample $\epsilon \sim \mathcal{N}(0, I)$. Then we could compute $z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon$. In the end, the distribution that we need to take the gradient becomes: $E_{q_\phi(z|x_n)}[logp_\epsilon(x_n|z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon)] - D_{KL}(q_\phi(z|x_n)\|p_\theta(z))$. Instead of sampling z from the distribution $\mathcal{N}(\mu, \Sigma)$ directly, we sample the $\epsilon$ from $\mathcal{N}(0, I)$ and then use it to calculate z. With this trick, none of the expectation are with respect to distributions that depend on our model parameters$\theta$. For instance, given a fixed X and $\epsilon$, the total loss is deterministic and continuous in the parameters of P and Q distribution and we could compute the gradient with SGD.

## 1.12 Question 1.12

Following the standard architecture design of VAE, there are two hidden layers in the encoder. For the activation function, I chose tahn and sigmoid, which produce satisfactory results. The decoder has only one hidden layer. For every 20 ephoch, I sample the constructor and obtain the images.

## 1.13 Question 1.13

The estimated lower-bound after training on the MNIST dataset could be visualized as below.

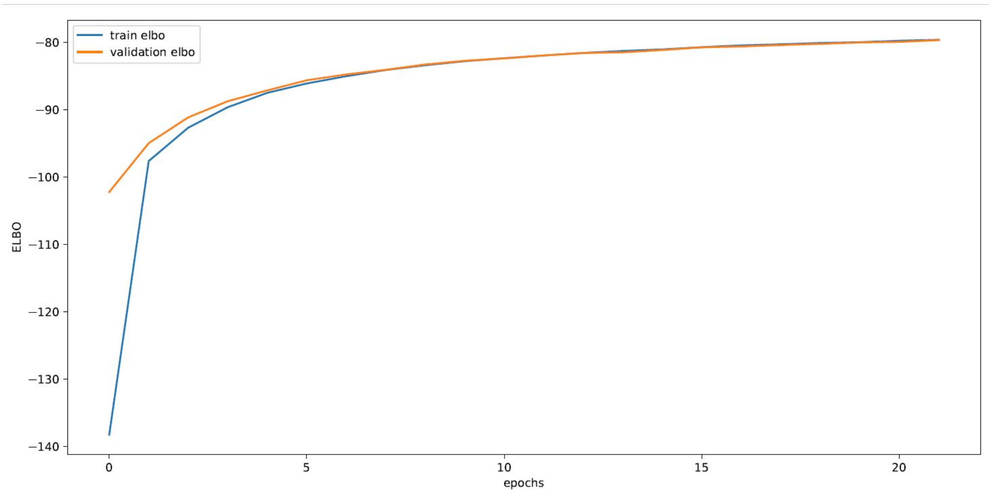

Figure 2: Lower-bound with a 20-dimensional latent space

## 1.14 Question 1.14

Three sampled images could be visualized as below:
before the training:

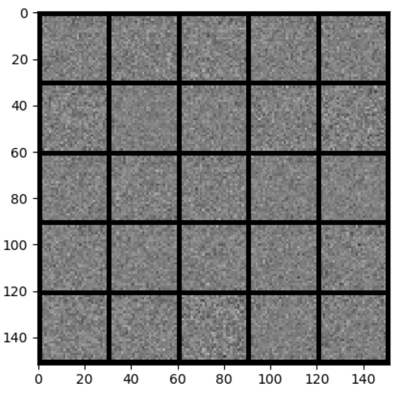

Figure 3: Generated image at 0 epoch

In the middle of the training process

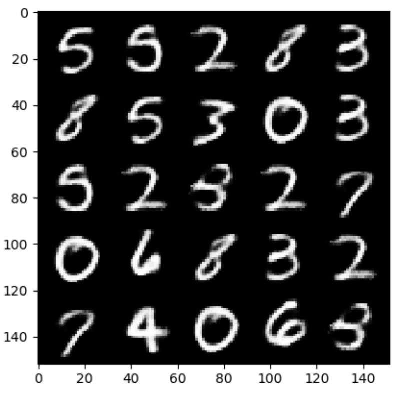

Figure 4: Generated image at 20 epoch

After the training process finishes:

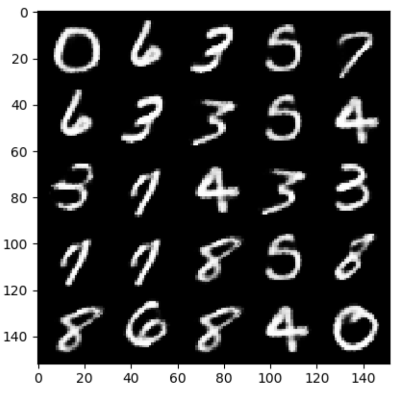

Figure 5: Generated image at 40 epoch

### 1.15 Question 1.15

20 points are sampled evenly from the distribution of z. Then I implemented percent point function and map it back to the z value in the latent space. The data manifold for $z_{dim} = 2$ could be visualized as below:

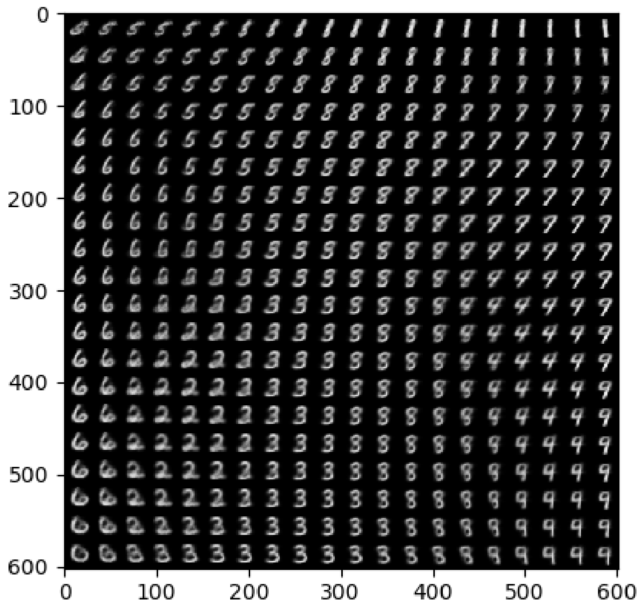

Figure 6: data manifold with 20 sampled points

## 2 Generative Adversarial Networks

### 2.1 Question 2.1

Generator: The input could be random sampled floats or noise and the output could be fake images which are generated based on trained neural networks.
Discriminator: The input should be fake and training images and the output should be a probability pair$p(p_t, p_f)$ where $p_t$ is the probability for an image to be true while $p_f$ indicates the probability for an image to be fake.

### 2.2 Question 2.2

$D(x) = 1$ represent the situation when the discriminator believes that real-world data x is a true image, while $D(G(z)) = 1$ shows that it believes that generated data $G(z)$ is a true image.
In the minimax game, the generator tries to minimize the probability that the discriminator predicts correctly.

### 2.3 Question 2.3

Under the Jensen-Shannon divergence, in the end, the optimal D(x) would take the value $\frac{p_{data}(X)}{p_{data}(X)+p_{model}(x)}$.

$$\begin{aligned} D_{JS}(p_r \| p_g) &= \frac{1}{2} D_{KL}(p_r \| \frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g \| \frac{p_r + p_g}{2}) \\ &= log(\frac{1}{2}) + log(\frac{1}{2}) \\ &= -2log(2) \end{aligned}$$

## 2.4 Question 2.4

If the discriminator performs perfect, the $D(G(Z)) = 1$ and $1 - D(G(Z)) = 0$. This further leads to problem for $log(1 - D(G(Z)))$, as the gradient vanishes during the training process. Learning and weight updates become not feasible.
This could be solved by different approaches, for instance we could define an alternative cost function $W(p_r, p_g) = \underset{\gamma \sim \prod(p_r, p_g)}{inf} \; E_{(x,y)_\gamma} |x - y|$. This ensures that the loss is seldom set to 0 and resolve the vanishing gradient problems.
Other solutions such as feature matching, which tries to match feature statistics instead of images could help us to obtain non-zero gradients.
In addition, we could try to balance the strength of generator and discriminator by separate the training process.

## 2.5 Question 2.5

The GAN follows the same structure as what is demonstrated in the instruction. For the generator, we have five hidden module. Each module contains one LeakyRelu, one batch-normalize layer and one activation layer. The discriminator has two hidden modules, each contains one linear layer and a batch-normalized layer. The total training epoch are around 18000 epochs. The quality of the images does not vary after 15000 epoch.

## 2.6 Question 2.6

Three sampled images could be visualized as below:
before the training:

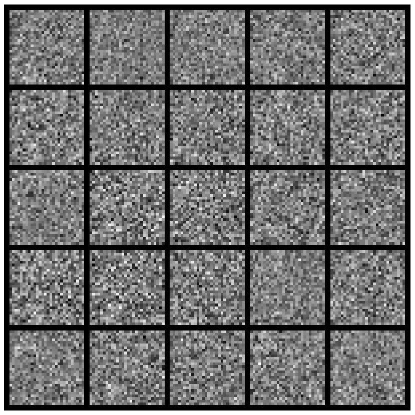

Figure 7: Image from generator at 0 epoch

In the middle of the training process:

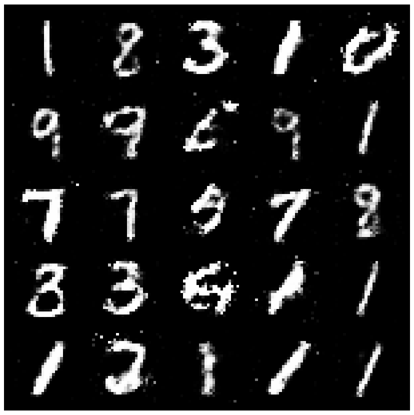
Figure 8: Image from generator at 8000 epoch

After the training process finishes:

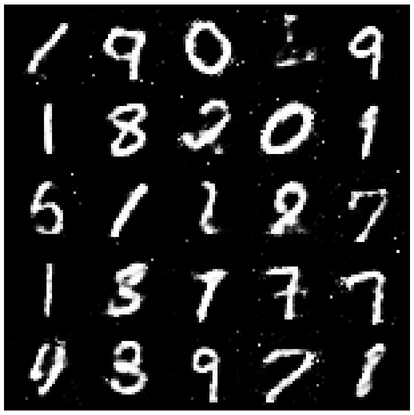
Figure 9: Image from generator at 155000 epoch

### 2.7 Question 2.7

As it is shown in the graph, the image changes from "9" to "4" in seven interpolation steps.

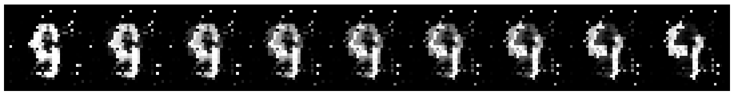
Figure 10: Gan image with 7 interpolation steps

## 3 Conclusion

### 3.1 Question 3.1

Both methods VAE and GAN are powerful generative models. In terms of the image generation performance, images generated by GAN have significantly higher quality compared with those generated by VAE. However, VAE has more meaningful representation of images in the latent space. This could be useful to do things such as imputation and completion. The idea of two models are quiet different. GAN implements a generator and discriminator in the model, for which the discriminator nudges the generator to produce realistic images. however, in VAE, the model incorporates a loss function, compare the generated images with ground truth and nudges the loss back to update models. The training time for VAE is significantly shorter than GAN. The GAN takes us around 3 hours to

train to generate decent images while it takes VAE 30 mins to converge and generate satisfactory results.
Another important fact is that GAN sometimes from model collapse. During the second time of training, I observed that in the later epoch, images are occupied with white dots and noises.