Variational auto-encoders and GANs

Yiangos Georgiou
11807288
University of Amsterdam
ygeorgiou12@gmail.com

1 Variational Auto Encoders

1.1 Question 1.1

variational autoencoders learn a latent variable model. In essence, the algorithm is trying to learn a probability distribution, conditioned on a few latent variables. These latent variables can be seen as a generalization of the first few principle components of PCA.Basically, VAE uses encoder and decoder to maximized marginal likelihood as PPCA.

1.2 Question 1.2

To sample from this model we need, firstly to sample latent variable z from $\mathcal{N}(0, 1_D)$. Then, by passing forward the decoder network, for each m in x_n parameter μ_n^m is obtained. Last step, is to use this parameter μ_n^m to sample every pixel in the image using bernoulli distribution in order to get values for each pixel. Moreover, another option is to use the means(outputs of the decoder) to generate X instead of sampling Bernoulli by using those means.

1.3 Question 1.3

The key idea is that by using a prior $p(z)$ which is a set of variables that are normally distributed can generate any kind of distribution that have the same number of dimensions and can be mapped through complicated function.

Thus, the intuition is that, given independent normally distributed values we can learn a function to map these values to whatever latent Z_d space we need to estimate our final output.

1.4 Question 1.4

(a) Monte Carlo Integration *is a technique for numerical integration using random numbers. It is a particular Monte Carlo method that numerically computes a definite integral.*

$$I = \int f(x)dx$$

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$\begin{aligned}
\log p(x_n) &= \log \mathbb{E}_{p(z_n)} p(x_n|z_n) \\
&\Rightarrow \log \int p(z)p(x_n|z)dz \\
&\Rightarrow \log \sum_{i=1}^{N} p(x_n|z_i)
\end{aligned}$$

Where z_i is sampled from p(z)

(b) This sampling is inefficient because not all z contribute to estimate the likelihood of x, most of z contribute by giving probabilities near to 0. That fact does not make us able to create good representations for X, in the optimal scenario we want to sample z that contribute most.

However, as the dimensionality increases the latent useful regions reduce exponentially, which makes it even more difficult to generate good sample of X in high dimensional space.

1.5 Question 1.5

KL-Divergence between two Gaussian distributions:

$$D_{KL}(\mathcal{N}(\mu_1, \sigma_1^2)||\mathcal{N}(\mu_2, \sigma_2^2)) = log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

a) The intuition behind KL-divergence is to investigate the difference between two distributions, thus if μs and σs is similar then we will get a very small divergence while if distribution's parameters are very different then we have large divergence value.

$$KL_D(\mathcal{N}(0.2, 1.2)||\mathcal{N}(0, 1)) = 0.0288$$
$$KL_D(\mathcal{N}(4, 3)||\mathcal{N}(0, 1)) = 8.4506$$

b) The formula for KL-Divergence is :

$$D_{KL}(\mathcal{N}(\mu_1, \sigma_1^2)||\mathcal{N}(\mu_2, \sigma_2^2)) = log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$
$$\Rightarrow D_{KL}(\mathcal{N}(\mu_1, \sigma_1^2)||\mathcal{N}(0, 1) = \log(\frac{1}{\sigma_1} + \frac{\sigma_1^2 + \mu_1^2}{2} - \frac{1}{2}$$

1.6 Question 1.6

Given the equation :

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

we can say that the right hand side is the lower bound of the log-probability. By definition KL divergence between two distributions is a positive number($D_{KL} \geq 0$). Thus, if we transform the above equation to:

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

we get the full equation of log-probability. So, given that KL divergence is positive we can derive this relation:

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

Thus, is called lower bound because is the lowest possible value for the log-probability.

1.7 Question 1.7

This objective is not computable because it requires computing the evidence of $p(z|x)$. Given that $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$ then, $p(x) = \int p(z)p(x|z)dz$ is intractable because we cannot compute $p(x|z)$ for every z. However, $ELBO$ is intractable and differentiable and we can train our model by using that term. The intuition is that, maximizing the ELBO is equivalent to minimizing the KL divergence. Thus, we expecting KL divergence term to become very small and then to be able to optimize $logp(x)$ directly.

1.8 Question 1.8

Regarding $\log p(x_n)$ and $D_{KL}(q(z|x_n)||p(Z|x_n))$ terms, what it happens when the $ELBO$ is getting larger $\log p(x)$ increases while $D_{KL}(q(z|x_n)||p(Z|x_n))$ deceases. The divergence decreases when the decoder $q(z|x_n)$ improve in approximating the true posterior $p(Z|x_n)$. As the divergence decreases the log-probability increase while $ELBO$ is also increasing.

1.9 Question 1.9

In the first term, $\log p(x|z)$ is the log-likelihood of the observed x given the code z that we have sampled. This term is maximized when $p(x|z)$ assigns high probability to the original x. It is trying to reconstruct x given the code z; for that reason we call $p(x|z)$ the decoder network and the term is called the reconstruction error.

The second term is the divergence between $q(z|x)$ and the prior $p(z)$, which we will fix to be a unit Normal. It encourages the codes z to look Gaussian. We call it the regularization term. It prevents $q(z|x)$ from simply encoding an identity mapping, and instead forces it to learn some more interesting representation.

1.10 Question 1.10

$$\begin{aligned}
\mathcal{L}_n^{recon} &= -\mathbb{E}_{q_\phi(z|x_n)}[\log p_\theta(x_n|Z)] \\
&= -\frac{1}{L}\sum_{l=1}^{L}[\log p_\theta(x_n|z_l)] \\
\log p_\theta(x_n|z_l) &= \log \prod_{i=1}^{D_x} Bern(X_{ni}; y_i) \\
&= \log \prod_{i=1}^{D_x} y_i^{x_{ni}} * (1-y_i)^{(1-x_{ni})} \\
&= \sum_{i=1}^{D_x} x_{ni} * \log y_i + (1-x_{ni}) * (1-y_i)
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}_{nreg} &= D(\mathcal{N}(\mu_1,\sigma_1)||\mathcal{N}(\mu_2,\sigma_2)) \\
&= \int p(x)\log\frac{p(x)}{q(x)}dx \\
&= \int p(x)\log\frac{\frac{1}{\sqrt{2\pi\sigma_1^2}}\exp(-\frac{(x-\mu_1)^2}{2\sigma_1^2})}{\frac{1}{\sqrt{2\pi\sigma_2^2}}\exp(-\frac{(x-\mu_2)^2}{2\sigma_2^2})} \\
&= \frac{1}{2}\log(\frac{\sigma_2^2}{\sigma_1^2}) + \frac{1}{2\sigma_1^2}[\int(x-\mu_1)^2p(x)dx] + \frac{1}{2\sigma_2^2}[\int(x-\mu_2)^2p(x)dx] \\
&= \frac{1}{2}\log(\frac{\sigma_2^2}{\sigma_1^2}) - \frac{\sigma_1^2}{2\sigma_1^2} + \frac{1}{2\sigma_2^2}[\int(x-\mu_1+\mu_1-\mu_2)^2]p(x)dx \\
&= \frac{1}{2}\log(\frac{\sigma_2^2}{\sigma_1^2}) - \frac{1}{2} + \frac{1}{2\sigma_2^2}[\int(x-\mu_1)^2p(x)dx + (\mu_1-\mu_2)^2 \\
&\int p(x)dx + 2(\mu_1-\mu_2)\int(x-\mu_1)p(x)dx] \\
&= \frac{1}{2}\log\frac{\sigma_2^2}{\sigma_1^2} - \frac{1}{2} + \frac{1}{2\sigma_2^2}[\sigma_1^2 + (\mu_1-\mu_2)] \\
\Rightarrow D(\mathcal{N}(\mu_1,\sigma_1)||\mathcal{N}(\mu_2,\sigma_2)) &= \frac{1}{2}\log\frac{\sigma_2^2}{\sigma_1^2} - \frac{1}{2} + \frac{\sigma_1^2(\mu_1-\mu_2)^2}{\sigma_2^2} \\
\Rightarrow D(\mathcal{N}(\mu_1,\sigma_1)||\mathcal{N}(0,1)) &= \frac{1}{2}\sum_i^D \log\frac{1}{\sigma_1^2} - 1 + \sigma_1^2 + (\mu_1)^2 \\
&= \frac{1}{2}\sum_i^D -\log\sigma_1^2 - 1 + \sigma_1^2 + (\mu_1)^2
\end{aligned}$$

1.11 Question 1.11

1.11.1 we need $\nabla_{\phi} L$

Encoder's goal is to learn how to represent x into the latent space z, thus is essential to update encoder's weights. Otherwise, the encoder will encode(output) x in random number, without ti extract any valuable information from it, and as a result the decoder to try to decode useless information and generate outputs without any meaning or context.

1.11.2 the act of sampling prevents us from computing $\nabla_{\phi} L$

The problem with sampling is that it is not continuous deterministic function, so has no gradients to back-propagate through the network and update the weights.

1.11.3 What the re-parametrization trick is, and how it solves this problem.

Introducing a new parameter e as an input layer or outside parameter allows us to re-parameterize z in a way that allows back-propagation to flow through the network. The fact that e doesn't depend on the output of encoder makes it insignificant for the back-propagation procedure. Thus, by using this noise e we an produce different sample for the decoder following the equation $z = \mu + \sigma * e$. Thus, using this trick we achieve both sampling and back-propagation.

1.12 Question 1.12

For VAE implementation, first we initialize the encoder and decoder networks. The encoder consist of 3 Weight matrices(input, μ,σ), for the input linear module we used tahn activation and then this output is fitted to the linear modules of μ and σ. The encoder provides as output an array of μ and σ in the same dimensions of latent space. Using that outputs we create the encoding z by sampling from Gaussians using μs and σs from the encoder . Decoder network takes as input the z and produce a value for every pixel.Decoder consist of 2 linear modules(input,) and we use tanh activation for the input linear and for mean linear we set sigmoid because we expecting a probability from 0-1. Then we can either sample from Bernoulli given that $mean$ as parameter the output of the decoder or just create the images using those probabilities.

1.13 Question 1.13

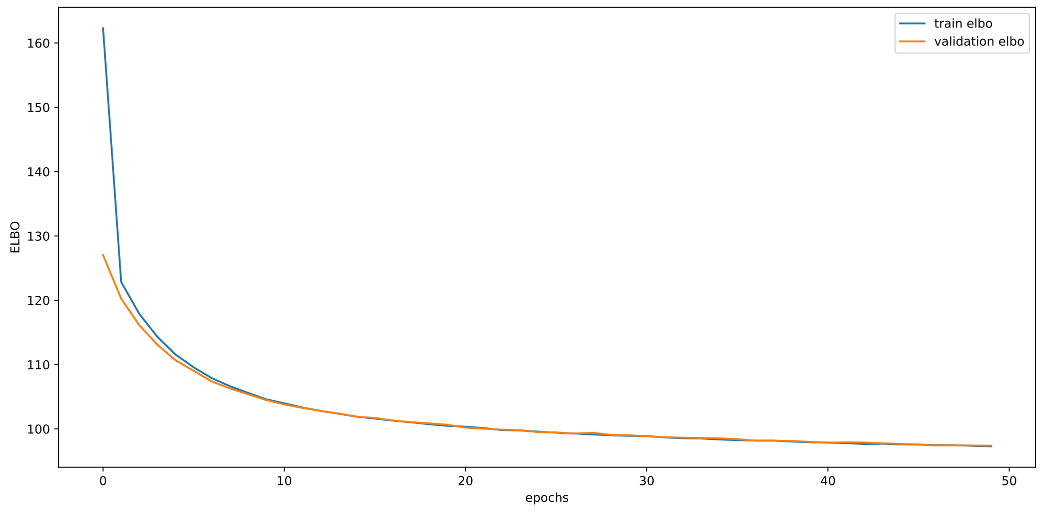

Figure 1: Average Negative ELBO values for training and validation data set through training

1.14 Question 1.14

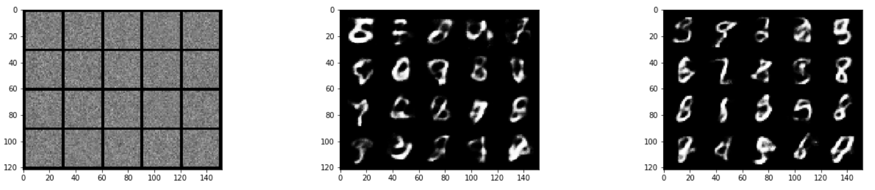

Figure 2: Performance improvement in sampling random numbers. This first image is derived from the start of training(epoch 0), the second from the middle (epoch 20) and the last after the end of the training (epoch 40)

1.15 Question 1.15

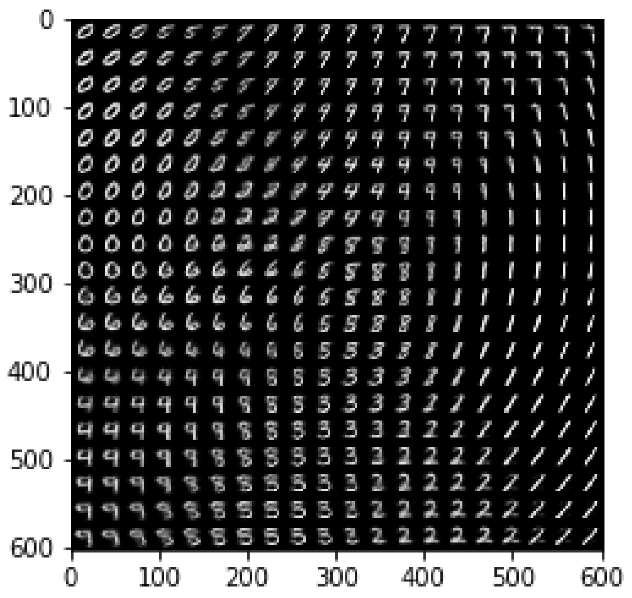

Figure 3: VAE manifold for latent space with dimension 2

2 Generative Adversarial Networks

2.1 Question 2.1

- **Generator :** The input to the generator is a series of randomly generated numbers called latent sample. Once trained, the generator can produce digit images from latent samples. Generator is a simple fully connected network that takes a latent sample and produces images or other data.
- **Discriminator :** The discriminator is a classifier trained using the supervised learning. It classifies whether an image is real or not. We train the discriminator using both the MNIST images and the images generated by the generator. The output of the discriminator is 1 for images the classified as real and 0 for images that classified as fake.

2.2 Question 2.2

The first term $E_{p_{data}(x)}[log(D(x))]$ gives the expected log likelihood of D to annotate as real a sample from our dataset that contains true data. On the other hand the second term $E_{p_z}(z)[log(1-D(G(Z)))]$ gives the expected log likelihood of the discriminator annotate that the Generator generated data as fake, if $D(G(Z))$ is small then also the loss for the second term will be large.

2.3 Question 2.3

The training of GAN is as is expressed in game theory a zero-sum game called minimax. The optimal point is GAN to reach Nash Equilibrium, that happens when one player will not change its action regardless of what the opponent may do. Thus, $V(D, G)$ is 0 when GAN converged.

2.4 Question 2.4

The problem here is that the discriminator can be trained really fast compared to generator. Moreover, based on Arjovsky's paper when we get to optimal discriminator the gradients for our GAN model will vanish.

This vanishing gradient issue can be solved by changing the objective GAN's discriminator function from $\log(1 - D(G(Z)))$ to $-\log(D(G(Z)))$. In the begining of the training the generator will produce images that is obviously fake, so the discriminator an be optimal, that makes $\log(1 - D(G(Z)))$ to become zero and as a result to vanish the gradients that we need to back-propagate through the network.

2.5 Question 2.5

For GAN implementation, first we initialize the generator and discriminator networks. The generator consist of a 4 hidden layer with leaky relu activation for the hidden and tanh activation for the output. The generator takes as input the gaussian noise and generates a fake image. The the discriminator which is a network with 2 hidden layer with leaky relu activation and an output layer that use a sigmoid activation gives a number between 0-1 which number define how real the generated image look like. The training procedure is different now due to the fact that we need to train one network at a time. So, in our implementation we start training with the generator and keep the discriminator constant and we calculate the generator loss. After that we train the discriminator and keep the generator constant. We keep doing this for 200 epochs.

2.6 Question 2.6

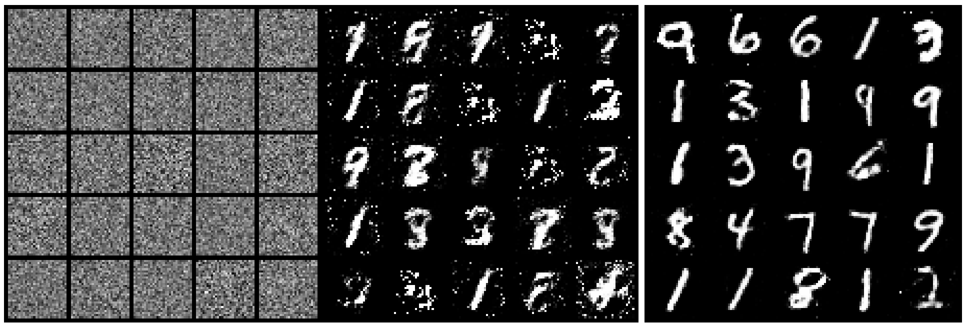

Figure 4: Performance improvement in sampling random numbers. The first image is derived from the start of training(epoch 0), the second from the middle (epoch 90) and the last after the end of the training (epoch 200)

2.7 Question 2.7

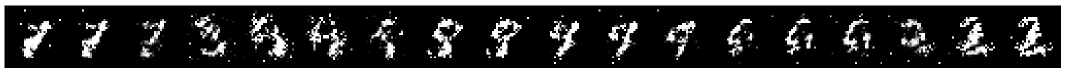

Figure 5: Interpolation samples(first:7 to 9,second:9 to 2))

3 Conclusion

VAEs are a probabilistic models whose explicit goal is latent modeling, this is achieved by comparing the encoder input to the decoder output. If the output is close to the input means that we managed to represent meaningfully the input in the latent space and compresses its input down to a vector - with much fewer dimensions than its input data. Moreover, it has the ability to set complex priors in the latent which is usefull in cases where you know that data follows a specific distribution or you have a desired latent distribution. Although, VAE is bad for generating new images basically because it learns an average representation and due to that fact results becomes blurry.

GANs follows an entirely different procedure. Encoder instead of compressing high dimensional data, it has low dimensional vectors as the inputs, high dimensional data as output. The generator tries to transform noise into realistic sample in order to "fool" the discriminator. The discriminator is another network that discriminate real and fake data. Moreover, in GANs there is no need to define mathematically the density function as it happens in VAEs.