Deep Probabilistice Generative Models - Variational Auto-Encoders

Romain MUSSARD, Salma OUARDI

12th October 2022

1 Introduction

In this lab work we focused on generating new data using deep neural network. For the 2 first parts of this project we worked with 2 different kind of Variational Autoencoder (VAE) one with a continuous latent space and one with a binary latent space. Then, on the final part we used a deterministic Autoencoder (AE) and a Gaussian Mixture Model (GMM) to generate new data.

2 VAE

2.1 Definitions

An Autoencoder is a deep neural network aiming to learn how to encode and decode data. In order to achieve that, an Autoencoder use an encoder to learn a latent representation of our data while the decoder learn how to decompress those data.

A Variational Autoencoder can be define as a regularized Autoencoder. More precisely, instead of learning a latent representation the VAE encode our data as a distribution over the latent space. We can then sample from this distribution to generate new data point. The final goal is to fit the real distribution of our data to make realistic generations. The regularization term used in the training objective is here to ensure that the distribution is close to the real one. Generally, we will put forward the hypothesis that our data follow a given distribution and make sure that the learned distribution is close to this distribution.

2.2 Encoder

The encoder takes as inputs $x \in \mathbb{R}^d$, in this project $d = 28 * 28 = 784$. The VAE output will be $z \in \mathbb{R}^n$ with $n < d$, z is the latent representation of our data. This is what we called a "bottleneck" since the encoder has to learn how to compress efficiently our data on a lower dimension.

We can then define the encoder by $q_\phi(z|x)$ since the goal of the encoder will be to find the parameters of the distribution $q_\phi(z|x)$ of the latent space z. We will then sample a vector from this distribution and pass it to the decoder to generate a new data point.

2.3 Decoder

The decoder takes as inputs $z \in \mathbb{R}^n$, which is the encoder output, and its output will be $\hat{x} \in \mathbb{R}^d$. The decoder's role is to learn how to decode the compressed information in the latent space, such that $x \approx \hat{x}$.

This time, We can denotate the decoder by $p_\theta(x|z)$ since the goal of the decoder is to return x' knowing z with this time $x' \sim p_\theta(x|z)$.

2.4 ELBO

The ultimate goal is to find the probability distribution that fits our data. Let $x \in \mathbb{R}^d$ the input of our model characterized by a probability distribution $P(x)$, the goal here is to find a good approximation of P thanks to p_θ with θ as parameter. Thus, this can be formulated by the learning objective :

$$\theta = argmax \prod_i p_\theta(x_i) = argmax \sum_i \log\left(p_\theta(x_i)\right)$$

Let $z \in \mathbb{R}^n$ and $x \in \mathbb{R}^d$, with z the encoded x on the latent space under a simple prior $p_\theta(z)$. We can then write $p_\theta(x)$ according to the joint distribution $p_\theta(x, z)$ between the two random variables :

$$p_\theta(x) = \sum_z p_\theta(x, z)$$

By applying the Bayes relation, we obtain that :

$$p_\theta(x) = \sum_z p_\theta(x|z) p_\theta(z)$$

As we saw during lectures, the computation of $p_\theta(x)$ is expensive. This is why we will use an approximation to speed up the calculations, assuming that $q_\phi(z|x) \sim p_\theta(z|x)$. These forewords will help us to simplify the calculations.

Let $L(x, \theta)$ such that:

$$\begin{aligned}
L(x,\theta) &= \log\left(p_\theta(x)\right) \\
&= \log \sum_z p_\theta(x|z)p_\theta(z) \\
&= \log \sum_z p_\theta(x|z)p_\theta(z)\frac{q_\phi(z|x)}{q_\phi(z|x)} \\
&= \log E_{z\sim q_\phi(z|x)}\left[\frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)}\right]
\end{aligned}$$

Thanks to Jensen's inequality we obtain:

$$\begin{aligned}
L(x,\theta) &\geq E_{z\sim q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)}\right)\right] \\
&\geq E_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right] + E_{z\sim q_\phi(z|x)}\left[\log\left(\frac{p_\theta(z)}{q_\phi(z|x)}\right)\right] \\
&\geq E_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right] - E_{z\sim q_\phi(z|x)}\left[\log\left(\frac{q_\phi(z|x)}{p_\theta(z)}\right)\right] \\
&\geq E_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right] - D_{KL}\left[q_\phi(z|x)|p_\theta(z)\right]
\end{aligned}$$

Thus, we obtain the evidence lower bound:

$$\epsilon(x,\theta,\phi) = \underbrace{E_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right]}_{\textbf{Reconstruction Loss}}\underbrace{-D_{KL}\left[q_\phi(z|x)|p_\theta(z)\right]}_{\textbf{KL Divergence}}$$

2.4.1 Interpretation

With the reconstruction, we want to make sure that the output of the decoder will be very similar to the input of the encoder in order to learn the latent presentation of our data. On the another hand, with the KL divergence, we make sure that the probability distribution over the latent space stays close to a given probability distribution (Generally a Gaussian or Bernoulli distribution). This insure that by sampling z over $p(z)$ we will be able to generate new data. Without this regularization, the VAE may act like an Autoencoder by affecting very different distributions to each pixel.

3 VAE with Continuous Latent Space VS Binary Latent Space

3.1 VAE with Continuous Latent Space

3.1.1 The generative story

1. $z \sim p(z) \longrightarrow$ a latent representation z is sampled from the prior distribution p(z) a multivariate Gaussian where each coordinate is independent.

2. $x \sim p(x|z,\theta) \longrightarrow$the data x is sampled from the conditional likelihood distribution $p(x|z)$ parameterized by a neural network.

3.1.2 How we computed KL divergence

We were able to calculate the KL divergence with the help of the following formula from the Auto-Encoding Variational Bayes paper:

$$KL = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2\right)$$

3.1.3 Reparametrization trick

The sampling process has to be expressed in a way that allows the error to be backpropagated through the network. So we use this trick to make the gradient descent possible even through random sampling.

The essence of the trick is to change how sampling is executed so instead of using $z \sim N(\mu, \sigma)$ we use $z = \mu + \sigma\epsilon$ where ϵ is an auxiliary noise variable $\epsilon \sim N(0, 1)$.

3.1.4 In the code :

- is calculated by: $e = torch.empty_like(mu).normal_(mean = 0., std = 1.)$

- z is sampled by: $z = mu + e * torch.exp(0.5 * log_sigma_squared)$.

Encoder

Let's define μ and σ as the Gaussian mean and variance we try to estimate. Let's define ϕ the relu function and f_1, f_2 and f_3, three linear functions.
Then we have $\mu = f_2\left(\phi(f_1(x))\right)$ and $\sigma = f_3(\phi(f_1(x)))$ $f\colon L^\infty(T) \to \mathcal{L}(H^2)$

3.2 VAE with Binary Latent Space

3.2.1 The generative story

1. $z \sim p(z) \longrightarrow$ a latent representation z is sampled from the prior distribution p(z) a Multivariate Bernoulli where each coordinate is independent.

2. $x \sim p(x|z,\theta) \longrightarrow$the data x is sampled from the conditional likelihood distribution $p(x|z)$ parameterized by a neural network.

3.2.2 How we computed KL divergence

We have $q_\phi(z|x) \sim Bern(0.5)$ and $p_\theta(x|z) \sim Bern(p_i)$. Thus the KL divergence can be computed as :

$$\begin{aligned}
D_{kl}\left(q_\phi(z|x), p_\theta(x|z)\right) &= \sum_x q_\phi(z|x) \log\left(\frac{q_\phi(z|x)}{p_\theta(x|z)}\right) \\
&= \sum_x (0.5^x \times 0.5^{(1-x)}) \log\left(\frac{(0.5^x \times 0.5^{(1-x)})}{p_i^x(1-p_i)^{(1-x)}}\right) \\
&= \sum_x \frac{1}{2} \log\left(\frac{0.5}{p_i^x(1-p_i)^{(1-x)}}\right) \\
&= \sum_x \frac{1}{2}\left(\log(2) - \log\left(p_i^x(1-p_i)^{(1-x)}\right)\right) \\
&= \sum_x \frac{1}{2}\left(\log(2) - x\log(p_i) - (1-x)\log(1-p_i)\right)) \\
&= \frac{1}{2}\sum_x \left(\log(2) - x\log(p_i) - (1-x)\log(1-p_i)\right)) \\
&= \frac{1}{2}\sum_x \left(\log(2) - H\left(p_\theta(x|z)\right)\right)
\end{aligned}$$

In conclusion we have :

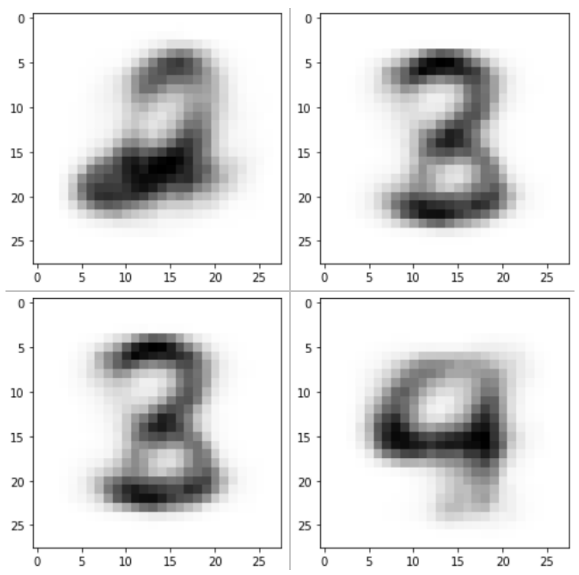
$$KL = \frac{1}{2}\sum_x \left(\log(2) - H\left(p_\theta(x|z)\right)\right)$$

3.3 Comparison between the continuous and binary latent spaces.

- We used a VAE with ***continuous latent space*** and ***binary latent space*** to generate our MNIST pictures, and as we can see in (figure 1.a) the continuous representation gave pretty good results compared to the binary latent space (figure 1.b).

  The reason behind this is that with a continuous latent space, meaning a space with several dimensions we are able to represent the data with more features compared to the binary latent space

  The ***binary latent space*** is basically bi-dimensional, which results in a compression that can be lossy, meaning that a part of the information is lost during the encoding process and cannot be recovered when decoding and we can see that clearly in (figure 1.2).


(a) Gaussian VAE

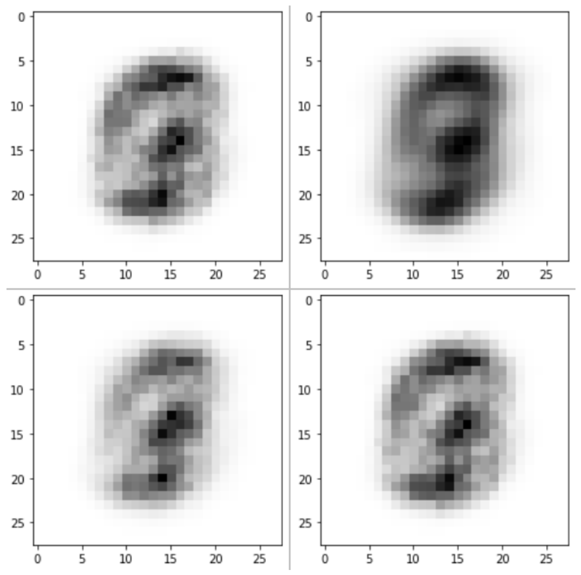
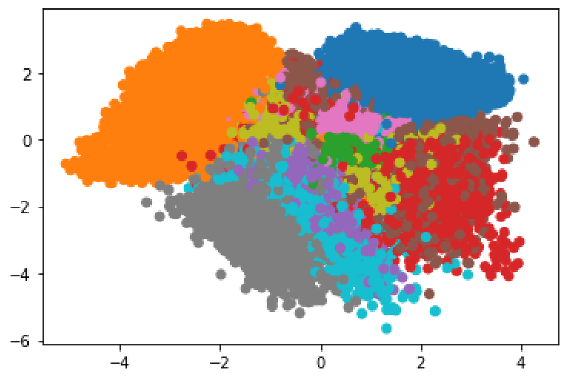

(b) Bernoulli VAE

Figure 1: Generated Samples


(a) Gaussian Latent Space

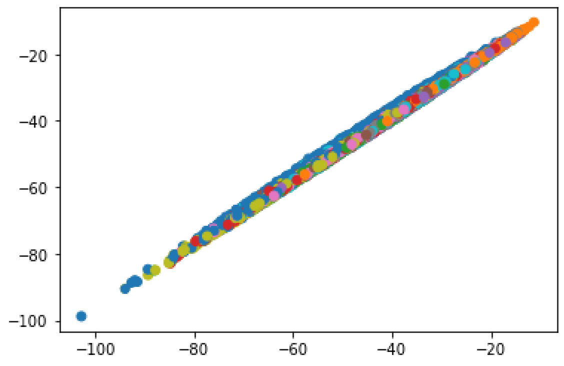
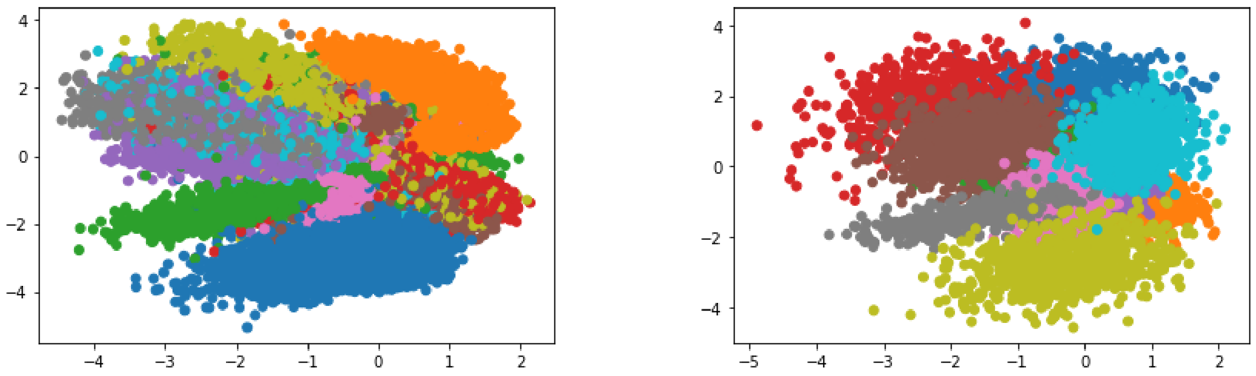

(b) Bernoulli Latent Space

Figure 2: Latent Space for dimension 2

4 Turning a Deterministic Auto-Encoder into a generative model

In the final part of this lab, we used an Autoencoder (AE) to learn the latent representation of our data and then we trained a Gaussian Mixture Model (GMM) on the latent representation of our data point. From this trained GMM, we were able to sample new data points on the latent space and by using the decoder of our Autoencoder we generated new data points.

As we can see in figure 3 the latent space is very similar between the Autoencoder and the one sampled by the GMM model. This also lead to very good generation, far better then the VAE we implemented as we can see in figure 4.



(a) Autoencoder Latent Sapce (b) GMM Latent Space VAE

Figure 3: Latent Space of dimension 2

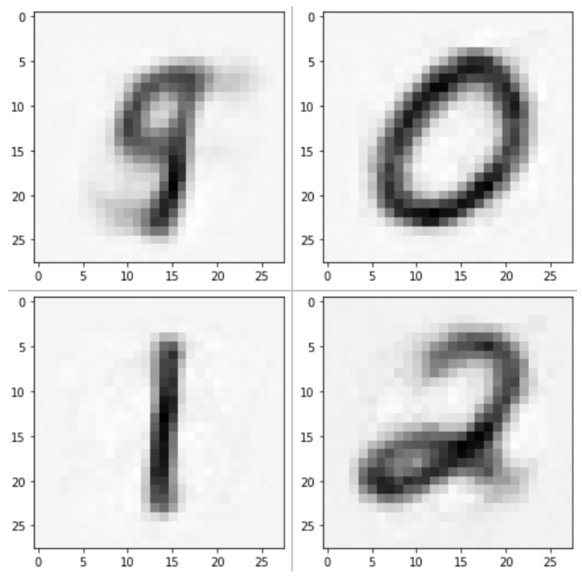


Figure 4: Generated Images by GMM