1 Variational Autoencoders :

Let $\mathbf{D}$ be a data set that contains observations $\mathbf{x}$, such that each $\mathbf{x}$ is the realization of random variables $\mathbf{X} = (X_1, .., X_m) \in \mathbb{R}^m$, where $(X_i)_{1 \leq i \leq m}$ are independent.
The observations x are generated based on hidden latent variables that we will denote by $Z = (Z_1, Z_2, \ .. Z_n)$.
We can write this :

$$Generative\ Story : \begin{cases} z \sim p_\theta(z) \quad (Hidden\ and\ not\ observed) \\ x \sim p_\theta(x|z) \quad (Observed) \end{cases} \tag{1}$$

To acquire more knowledge about the latent variables we have to compute the posterior distribution by using the Bayes theorem and we can write the posterior distribution as :

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} = \frac{p_\theta(x|z)p_\theta(z)}{\sum_z p_\theta(x|z)p_\theta(z)} \tag{2}$$

It's hard to compute the posterior because the denominator is **intractable**. (if we have 100 dimensions in the latent space, the denominator will contain 100 sums over all the realizable values which is a gigantic operation to compute).
A solution has been proposed, and it consists on approximating the posterior $p_\theta(z|x)$ by another distribution $q_\phi(z|x)$ which must be tractable and easy to compute.
To do this, we must minimize the Kullback–Leibler divergence that measures a similarity between 2 distributions.

$$\begin{aligned} KL\Big[\ q_\phi(z|x)||p_\theta(z|x)\ \Big] &= \sum_z q_\phi(z|x) log\Big[\frac{q_\phi(z|x)}{p_\theta(z|x)}\Big] \\ &= \sum_z q_\phi(z|x) log\Big[q_\phi(z|x)\frac{p_\theta(x)}{p_\theta(x|z)p_\theta(z)}\Big] \\ &= \sum_z q_\phi(z|x) log\Big[\frac{q_\phi(z|x)}{p_\theta(z)}\Big] + \sum_z q_\phi(z|x) log\Big[p_\theta(x)\Big] - \sum_z q_\phi(z|x) log\Big[p_\theta(x|z)\Big] \\ &= log\Big[p_\theta(x)\Big] + \sum_z q_\phi(z|x) log\Big[\frac{q_\phi(z|x)}{p_\theta(z)}\Big] - \sum_z q_\phi(z|x) log\Big[p_\theta(x|z)\Big] \\ &= log\Big[p_\theta(x)\Big] + KL\Big[\ q_\phi(z|x)||p_\theta(z)\ \Big] - \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] \end{aligned}$$

We denote :

$$L(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] - KL\Big[\ q_\phi(z|x)||p_\theta(z)\ \Big] \tag{3}$$

Hence :

$$KL\Big[\ q_\phi(z|x)||p_\theta(z|x)\ \Big] = log\Big[p_\theta(x)\Big] - L(\theta, \phi) \tag{4}$$

Knowing that $KL\Big[\ q_\phi(z|x)||p_\theta(z|x)\ \Big] \geq 0$, we get :

$$log\Big[p_\theta(x)\Big] \geq L(\theta, \phi) \tag{5}$$

L in known as the Evidence Lower Bound (ELBO).
By going back to equation (4), $log(p_\theta(x))$ does not depend on ϕ so minimizing $KL\Big[\ q_\phi(z|x)||p_\theta(z|x)\ \Big]$ w.r.t ϕ is equivalent to maximizing L w.r.t ϕ.
We also want to maximize the log-likelihood of the data (the evidence) $\frac{1}{|D|}\sum_{x \in D} log(p_\theta(x))$ w.r.t θ, by maximizing the evidence we acquire more information about the distribution from which the observations are drawn . So by going back to the in-equation (5), we have the ELBO as a lower bound for the evidence which means that maximizing this lower bound w.r.t θ, we maximize the evidence too.

So the goal is to maximize the ELBO w.r.t to θ and ϕ. And since we are going to work with neural networks it's better to transform the problem into a minimization problem. So we are going to minimize -ELBO instead, Hence we will minimize :

$$Loss(\theta, \phi) = KL\Big[\ \ q_\phi(z|x)||p_\theta(z)\ \ \Big] - \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] \tag{6}$$

The first term $KL\Big[q_\phi(z|x)||p_\theta(z)\Big]$ is the Kullback–Leibler divergence from the prior, and the second term $\mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big]$ represents a reconstruction term. When we minimize the $Loss$ we minimize the first term (KL) and we maximize the second term $\mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big]$ or we minimize also $-\mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big]$.
To do this we can imagine that $q_\phi(z|x)$ and $p_\theta(x|z)$ are both neural networks, where $q_\phi(z|x)$ takes the observation as input and projects it into the latent space by minimizing the divergence between the posterior and the prior, and $p_\theta(x|z)$ tries to reconstruct the observation knowing its hidden variable. This looks like an encoding and decoding process, which is where the term Autoencoder comes from, and since we are approximating the latent space to a well-defined distribution the term becomes Variational Autoencoder (VAE).

2 VAE with Normal latent space and Bernoulli observed space :

In this section we make the assumption that the prior distribution $p(z)$ is a multivariate Gaussian where each coordinate is independent and follows $\mathcal{N}(0, 1)$, we also assume that X are independent Bernoulli random variables.
We want to approximate the prior $p(z)$ by $q_\phi(z|x)$, and we do that by assuming that :

$$\hat{Z} = (\hat{Z}_1, .., \hat{Z}_n) \sim q_\phi(z|x) \Rightarrow \forall i \in \{1, .., n\}\ \hat{Z}_i \sim \mathcal{N}(\mu_i, \sigma_i)$$

We denote $\mu = (\mu_1, .., \mu_n)$ and $\sigma = (\sigma_1, .., \sigma_n)$.
We want our reconstructed observation to be as close as possible to the distribution of the observations. Hence we must have :

$$\hat{X} = (\hat{X}_1, .., \hat{X}_m) \sim p_\theta(x|z) \Rightarrow \forall i \in \{1, .., m\}\ \hat{X}_i \sim \mathcal{B}(\omega_i)$$

We denote $\omega = (\omega_1, .., \omega_m)$.
So here we are first searching for our parameters μ, σ and ω.
To do so we will train 2 neural networks, by minimizing the loss in equation (6). Hence we must find the gradient of our loss w.r.t θ and w.r.t ϕ.
Let's start by finding an analytical expression for the loss:

- We can find The KL divergence between 2 normal distributions in the paper https://arxiv.org/pdf/1312.6114.pdf provided in the notebook:

$$KL\Big[\ \ q_\phi(z|x)||p_\theta(z)\ \ \Big] = -\frac{1}{2}\sum_j \Big[1 + log[\sigma_j^2] - \mu_j^2 - \sigma_j^2\Big]$$

- For one sampling $\hat{x} = (\hat{x}_1, \hat{x}_2, .., \hat{x}_m)$ and knowing that $\hat{x}_j$ are independent, the reconstruction term becomes :

$$log(p_\theta(\hat{x}|z)) = log(\prod_{i=1}^{m} p_\theta(\hat{x}_i|z)) = log(\prod_{i=1}^{m} \omega_i^{\hat{x}_i}(1-\omega_i)^{1-\hat{x}_i})$$

  Hence :

$$log(p_\theta(\hat{x}|z)) = \sum_{i=1}^{m} \hat{x}_i log(\omega_i) + (1-\hat{x}_i) log(1-\omega_i) \tag{7}$$

Now let's compute the gradients:

- Gradients w.r.t θ :

$$\begin{aligned}\nabla_\theta Loss(\theta,\phi) &= \nabla_\theta KL\Big[q_\phi(z|x)||p_\theta(z)\Big] - \nabla_\theta \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] \\ &= \nabla_\theta KL\Big[q_\phi(z|x)||p_\theta(z)\Big] - \mathbb{E}_{q_\phi(z|x)}\Big[\nabla_\theta log(p_\theta(x|z))\Big]\end{aligned}$$

  This expression is easily computed. We can use Monte-Carlo by sampling multiple values of z_k from $q_\phi(z|x)$ to estimate the expectation, i.e :

$$\mathbb{E}_{q_\phi(z|x)}\Big[\nabla_\theta log(p_\theta(x|z))\Big] \simeq \frac{1}{N}\sum_{k=1}^{N} \nabla_\theta log(p_\theta(x|z_k))$$

- Gradients w.r.t ϕ:

$$\nabla_\phi Loss(\theta,\phi) = \nabla_\phi KL\Big[\ \ q_\phi(z|x)||p_\theta(z)\ \ \Big] - \nabla_\phi \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big]$$

  Here, we are taking the derivative w.r.t ϕ, and the reconstruction term has an expectation w.r.t q_ϕ. To solve this problem we use the reparameterization trick and we will introduce a new variable ϵ where $\epsilon \sim \mathcal{N}(0,1)$.
  When we sample z from q_ϕ, we will have $z = (z_1, .., z_n)$ and each z_i is sampled from $\mathcal{N}(\mu_i, \sigma_i)$. This is equivalent to sampling $\epsilon = (\epsilon_1, ..\epsilon_n)$ such that each ϵ_i is sampled from $\mathcal{N}(0,1)$ and by doing this operation : $\sigma \bullet \epsilon + \mu$ we get a sample from q_ϕ.($\bullet$ indicates multiplication entry one by entry) Hence z can be rewritten as $z = h(\epsilon)$ and we get :

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] &= \nabla_\phi \mathbb{E}_{p(\epsilon)}\Big[log(p_\theta(x|h(\epsilon)))\Big] \\ &= \mathbb{E}_{p(\epsilon)}\Big[\nabla_\phi log(p_\theta(x|h(\epsilon)))\Big]\end{aligned}$$

  Now the derivation w.r.t ϕ is possible and by using the Monte-Carlo sampling we can estimate the expectation.

So the computation of the gradients is possible at this stage and we can train our networks.

The dataset used in training is a binarized MNIST which are hand-written digits and we reshaped them to get vectors to use to train the networks.

In the following figures, we represents some results we obtained :

Figure 1: Results when sampling a new observation using the ω parameters.

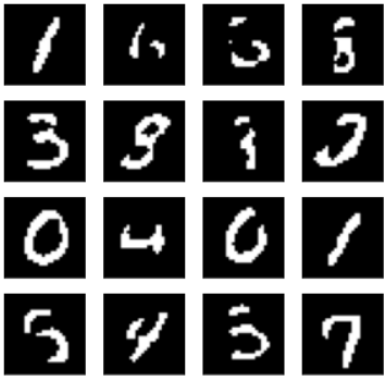

Figure 2: Results using a threshold on the ω parameters (without sampling)

After training we use the network to generate values that represent parameters of a Bernoullis distribution with parameters $\omega = (\omega_1, .., \omega_m)$ and we sample from the Bernoullis to obtain this the images in figure 1.
As we can see the results seem to be not accurate and very blurry. So we also tried to generate images by using a kind of threshold on the output of the decoder without sampling from it, i.e if the ω_k value is greater than 0.5 then we consider a value of 1 and if it's lower we consider a value of 0 which resulted in a better results (figure 2).

3 VAE with binary latent space and binary observed space :

In this section, we assume that the hidden latent variables Z are independent Bernoulli random variables and X also are independent Bernoulli random variables.
We assume also that the prior follow all Bernoulli with parameter $\frac{1}{2}$,i.e $\forall i \in \{1,..,n\} Z_i \sim \mathcal{B}(y = \frac{1}{2})$.
In the equation (6), we want to approximate the prior with $q_\phi(z|x)$, and we are going to do so, first, by assuming that :

$$\hat{Z} = (\hat{Z}_1,..,\hat{Z}_n) \sim q_\phi(z|x) \Rightarrow \forall i \in \{1,..,n\}\ \hat{Z}_i \sim \mathcal{B}(\mu_i) \text{ where } \mu_i \in \mathbb{R}$$

Since X are also Bernoulli random variables, we want the distribution of the reconstructed observation to be as close as possible to the distribution of the observations. Hence we must have :

$$\hat{X} = (\hat{X}_1,..,\hat{X}_m) \sim p_\theta(x|z) \Rightarrow \forall i \in \{1,..,m\}\ \hat{X}_i \sim \mathcal{B}(\omega_i) \text{ where } \omega_i \in \mathbb{R},\ \omega_i \text{ depends on } z.$$

We start by computing an analytical expression of the loss in equation (6).

- The KL divergence :

$$\begin{aligned}
KL\Big[\ q_\phi(z|x)||p_\theta(z)\ \Big] &= \sum_z q_\phi(z|x) log[q_\phi(z|x)] - \sum_z q_\phi(z|x) log[p_\theta(z)] \\
&= -H[q_\phi(z|x)] - \sum_z q_\phi(z|x) log[\prod_{i=1}^{n} p_\theta(z_i)] \\
&= -H[q_\phi(z|x)] - \sum_z q_\phi(z|x) \sum_{i=1}^{n} log[p_\theta(z_i)] \\
&= -H[q_\phi(z|x)] - \sum_z q_\phi(z|x) \sum_{i=1}^{n} log[y^{z_i}(1-y)^{1-z_i}] \quad (y = 1/2) \\
&= -H[q_\phi(z|x)] - \sum_z q_\phi(z|x) \sum_{i=1}^{n} log[y] \\
&= -H[q_\phi(z|x)] - \Big[\sum_z q_\phi(z|x)\Big] n * log[y]
\end{aligned}$$

  We know that $\sum_z q_\phi(z|x) = 1$, then :

$$KL\Big[q_\phi(z|x)||p_\theta(z)\Big] = -H[q_\phi(z|x)] - n * log[y] \tag{8}$$

- The reconstruction term remains the same as before, and we use the equation (7) for one sampling.

Let's see how to compute the gradients :

- Gradients w.r.t θ have the same form as the previous section :

$$\nabla_\theta Loss(\theta,\phi) = \nabla_\theta KL\Big[\ q_\phi(z|x)||p_\theta(z)\ \Big] - \mathbb{E}_{q_\phi(z|x)}\Big[\nabla_\theta log(p_\theta(x|z))\Big]$$

  This expression is easily computed. We can use Monte-Carlo by sampling multiple values of z_k from $q_\phi(z|x)$ to estimate the expectation, i.e :

$$\mathbb{E}_{q_\phi(z|x)}\Big[\nabla_\theta log(p_\theta(x|z))\Big] \simeq \frac{1}{N}\sum_{k=1}^{N} \nabla_\theta log(p_\theta(x|z_k))$$

- Gradients w.r.t ϕ :

$$\nabla_\phi Loss(\theta,\phi) = \nabla_\phi KL\Big[\ q_\phi(z|x)||p_\theta(z)\ \Big] - \nabla_\phi \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big]$$

In this gradients, we are taking the derivative w.r.t to ϕ, and the reconstruction term is having an expectation w.r.t to q_ϕ, before we used the reparameterization-trick to solve it, in this case we must use the Score Function Estimator :

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] &= \nabla_\phi \sum_z q_\phi(z|x) log(p_\theta(x|z)) \\ &= \sum_z log(p_\theta(x|z)) \nabla_\phi q_\phi(z|x) \\ &= \sum_z q_\phi(z|x) log(p_\theta(x|z)) \nabla_\phi log(q_\phi(z|x))\end{aligned}$$

Hence :

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z))\Big] = \mathbb{E}_{q_\phi(z|x)}\Big[log(p_\theta(x|z)) \nabla_\phi log(q_\phi(z|x))\Big] \tag{9}$$

Now that we managed to get the derivative inside the expectation, we can always use Monte-Carlo sampling to estimate it.
The equation (9) is called the score function estimator, it is unbiased but this estimator has high variance, and to minimize it, we introduce the control-variates method, and instead of using the last estimator we use this one :

$$\mathbb{E}_{q_\phi(z|x)}\Big[(log(p_\theta(x|z)) - c(x)) \nabla_\phi log(q_\phi(z|x))\Big] \tag{10}$$

Where $c(x)$ is a deterministic quantity to control the variance, and in our case we take $c(x)$ to be the running average of all previous values of $log(p_\theta(x|z)$.

At this stage, we know how to compute the gradients w.r.t θ, ϕ, but we must code this in Pytorch, and in Pytorch we usually code losses in the form of one sampling only, and in our case if we code it in that form we won't represent correctly the gradients w.r.t ϕ, so we must add a term that's going to solve this problem.
In practice : For one sampling, we code the loss like this :

$$LOSS = KL\Big[q_\phi(z|x)||p_\theta(z)\Big] - log[p_\theta(x|z)] - log[p_\theta(x|z)] log[q_\phi(z|x)] \tag{11}$$

We put the term in red, to signify that it should not be derived w.r.t θ during back-propagation. So if we re-take the loss in equation (11) and compute its gradients w.r.t θ, ϕ we will get the previous gradients which are correct.
In the following figure, we show how we code the SFE and the control variates method.

```
reconstruction_logits = decoder(z)

# We compute the BCE loss and summing over the second dimension
# Mathematically the BCE is - reconstruction_loss actualy.
# but we named the variable reconstruction loss anyway.
reconstruction_loss = F.binary_cross_entropy_with_logits(
    reconstruction_logits,
    batch,
    reduction="none"
).sum(-1)

kl_div_bern = KL_with_bernoulli_prior(probs)
# We detach the reconstruction loss in the score function estimator
# runing average is the mean of all previous reconstruction losses
SFE  = - log_probs_sum * (reconstruction_loss.detach() - running_avg)

# The two first terms are actually what we want to minimize
# we added another term SFE to have correct gradients during training
# and we added another SFE.detach() to recover the values of the loss we really
# want to minimize
loss = reconstruction_loss + kl_div_bern - SFE + SFE.detach()
```

Figure 3: The loss coded in Pytorch.

The method **detach()** in Pytorch allows us to not compute derivation for a certain variable.
The reason why we added another **SFE.detach()** is to recover instantly the loss we truly aim to minimize, and the running average quantity is updated on each iteration.

We train the network this time with a different loss but the same dataset of binarized MNIST.
In the following figures we generate images by using the same methods used for the normal latent space in the previous VAE.

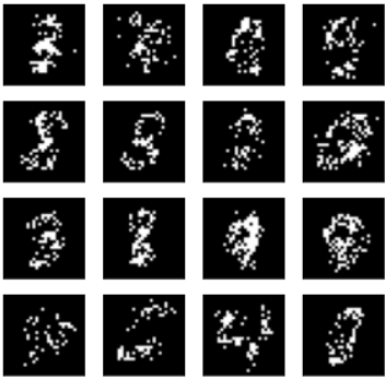

Figure 4: Results when sampling a new observation using the ω parameters.

Figure 5: Results using a threshold on the ω parameters (without sampling)

Here the results are worse than before, they seem a bit better using the threshold in some examples, but we can say that in general this model does not generate good samples.

4 Turning a Deterministic Auto-Encoder into a generative model

The difference between Auto-Encoder and Variational Auto-Encoder, is that Auto-Encoder reconstruct the observations by projecting it into a latent space that does not necessarily approximate the prior $p_\theta(z)$.
The difference also appears in the loss function of the Auto-Encoder because it does not involve a Kullback-Leibler divergence term in it.
Another difference is that in VAE we can use the decoder as a generative model by feeding it samples from the approximate of the prior, but in the Auto-Encoder we can't do this because we don't know the parameters of the approximate of the prior. So we are going to use GMMs to solve this problem and turn the Auto-Encoder into a generative model too.

We start by training the auto-encoder to reconstruct the training set using the loss function $L_{AE}(\theta) = \mathbb{E}\Big[log(p_\theta(x)\Big]$. So for one sampling the loss will look like :

$$log(p_\theta(x^j)) = \sum_{i=1}^{m} x_i^j log(\omega_i) + (1 - x_i^j) log(1 - \omega_i) \tag{12}$$

Where ω are the parameters of the Bernoulli followed by the training set realization.
So after training the AE, we take all observations of the training set $D = \{(x)_{1 \leq i \leq N}\}$, and passed them through the encoder only. In our case, the encoder only return 2 values that we denote $Z = (Z_1, Z_2)$, so we end up with a data-set $D_z = \{Z_i = (Z_1^i, Z_2^i)_{1 \leq i \leq N}\}$.
We denote $p(z)$ the marginal distribution of the dataset D_z.
Now we use Gaussian mixture model to model the dataset D_z, i.e we assume that :

$$p(z) = \sum_{r=1}^{K} \pi_r \mathcal{N}(z|\mu_r, \Sigma_r) \quad s.t \quad \sum_{r=1}^{K} \pi_r = 1 \tag{13}$$

We estimate the parameters $(\pi_r, \mu_r, \Sigma_r)_{1 \leq r \leq K}$ by using the Expectation Maximization algorithm.

The following figure represents the data points scatter of D_z.

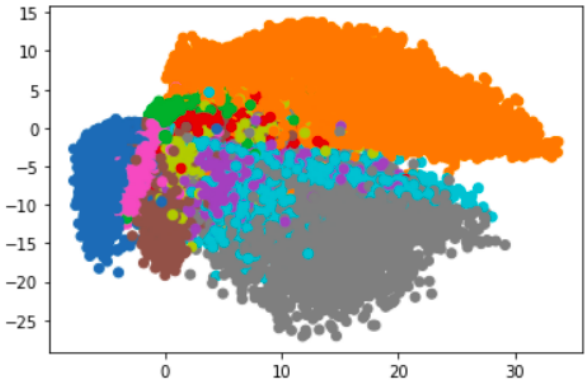

Figure 6: The scatter of data points of D_z.

The data points are colored according to which digit they represent of the MNIST data.
The orange and gray represent the data points in an elliptical form which looks like a normal distribution, also the blue one looks like an ellipse rotated. Also the density of points gets bigger as we move to the center of the ellipses. So We believe we can approximate this with a GMM.

When we finish training the GMM on D_z using 20 components, we randomly sample (according the probabilities π_r) a component and then use the GMM distribution to sample an observation of this components.
We do this step 20 000 times and we plot it in order to see if the distribution looks similar to the one of the latent space of the AE.

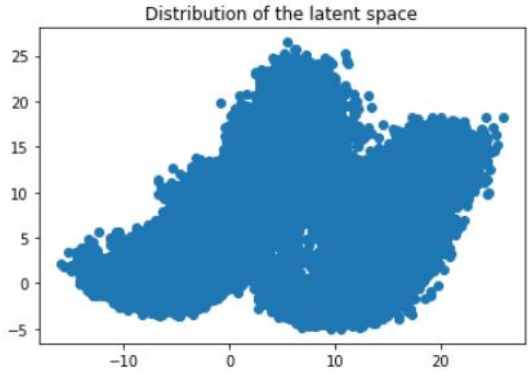

Figure 7: Distribution of the latent space.

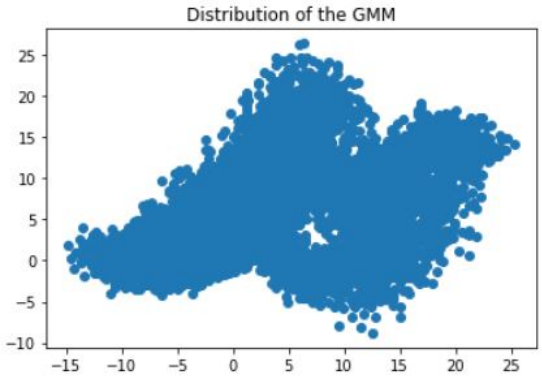

Figure 8: Distribution of the GMM.

We can see that the distributions are really close to one another. It seems that the GMM can reproduce the distribution of the latent space (produced by the encoder).
We can now generate values sampled from one the Gaussians of the GMM and then pass it to the decoder.

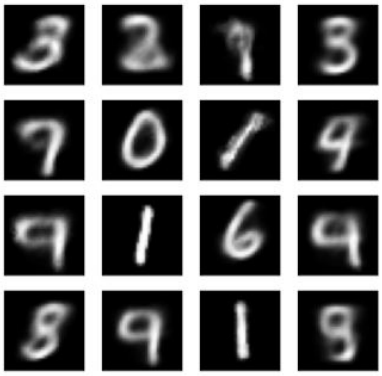

Figure 9: Results of the decoder using new GMM samples in the input.

As we could think after seeing the distribution of the GMM, the results are good. Since the GMM can approximate well the distribution of latent space and the decoder can generate quite good output. Even if the images are a bit blurry, it seems to be one of the best results we had.

5 Conclusions and drawbacks :

In this work, we trained 3 models. The first 2 models are VAEs with different distributions for the latent space, and the last one is just an AE that we managed to transform into a generative model by modeling the latent space with a GMM.

The last model performed better in terms of the quality of images generated.
In the following figures, we show the evolution of the Kullback-Leibler divergence and the reconstruction terms (-Binary-Cross-Entropy) for the first 2 VAEs.

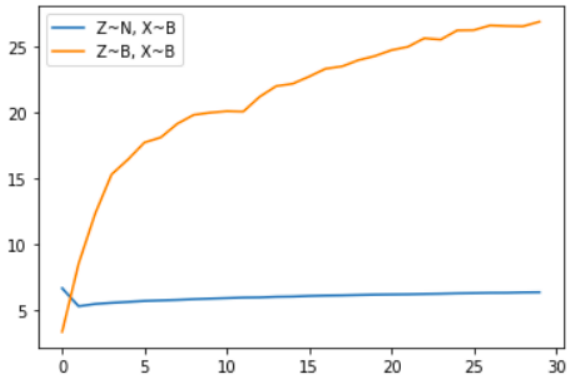

Figure 10: The evolution of the KL in the 2 models with epochs.

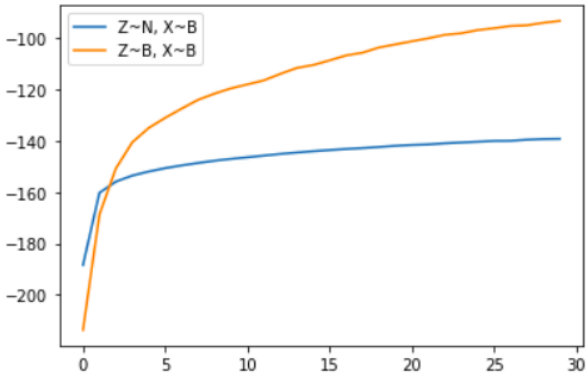

Figure 11: The evolution of the reconstruction terms (-BCE) with epochs.

- In Figure 10, we can see that the reconstruction term increases with epochs, which is a good sign that the decoder improves in reconstructing the observation on the input.
- In Figure 11, we can see that the first model where the latent space was modeled with multivariate normal distribution, the KL divergence is stable during the training, but for the second model the KL divergence is increasing, which indicates that the encoder can't approximate the distribution of the latent space.

In the second model the encoder can't approximate the latent space distribution, hence the encoder can't generate good parameters to sample from $q_\phi(z|x)$, but the decoder manages improves in reconstructing the observation on the input regardless of this bad samples from the encoder.
To solve the previous problem, and to push the encoder to approximate the prior $p(z)$, we could try some solutions like :

- Different choice of the family of the posterior approximation q_ϕ.
- We can also try to add weights to the loss, since the KL divergence does not improve it means that the weights are not penalized in that direction, so we could try to add weights to the loss, especially in the KL term to make gradients more significant in this direction.
- We can also try to use convolutions instead of fully-connected layers in the networks.

In this work of training VAEs, we believe that the things that need to be taken into consideration when doing futur works on VAEs are :

- We need to be careful on how to backpropagate the gradients (reparametrization trick or SFE).