# Deep Probabilistice Generative Models - Variational Auto-Encoders

Jiangnan Huang, You Zuo

October 30, 2020

Instructor: Caio Corro

**Abstract**

This paper aims to make a more in-depth discussion about the variational auto-encoder model and demonstrate how to implement it in practice. We first introduce the main ideas and logics behind the Variational Auto-Encoders, then compare it with another generative model Sigmoid Belief Network, and give some points to note when implementing the model in practice.
Finally, we demonstrate and analyze the results obtained by the 3 different models implemented from our experiments.

**Keywords**: VAE; SBN; mean field theory; variational methods; Score function Estimator

## 1 Main ideas of Variational Auto-Encoders

### 1.1 Definition of auto-encoder

To begin with, VAE is a kind of auto-encoders. "Auto" literally means self-driven, while the encoder is a tool or technique to compress information by extracting the most representative features. So implementing an Auto-Encoder to a dataset means basically encoding this dataset itself. And later with those extracted features, we can reconstruct the original data via a decoder.

More precisely, it is an unsupervised approach for learning a lower-dimensional feature representation $z$ from unlabeled training data $x$. The size of $z$ is usually smaller than $x$ because $z$ is supposed to represent the most important features of our input data $x$. For instance, if our task is to generate human faces, ideally, the feature variables could be something like the degree of smile or the posture of the head etc.

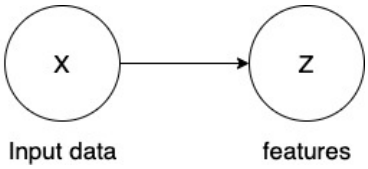


Figure 1: the encoder process

After having these representative features $z$, we can then use them to reconstruct original data (new human faces for the previous example). We represent the output of the decoder by $\hat{x}$ here:

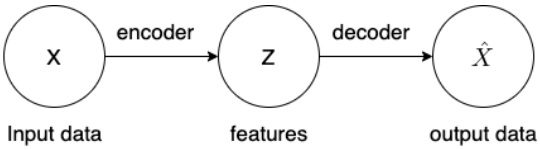


Figure 2: whole process of encoder and decoder

As we can see, both the encoder and decoder process are actually estimating some conditional distributions, which could be very complicated in practice. So in general, people would simply represent them with neural networks.

## 1.2 Introduction of Variational Method

### 1.2.1 Intractability

The probabilistic spin on autoencoders will let us sample from the model to generate data. So we might think about sampling from the true prior $p(z)$ first, then sampling from the conditional distribution $p(x|z^{(i)};\theta)$.

In order to represent this distribution, it requires us to make assumptions about the family of the prior $p(z)$. Normally people choose $p(z)$ to be simple like Gaussian or Bernoulli. If we assume that $z$ is a Gaussian vector, then the output of the encoder network would be a mean vector $\mu_{z|x}$ and a covariance matrix $\Sigma_{z|x}$. However, if we assume that z follows a Bernoulli distribution then the output of the encoder would only be the mean $\mu_{z|x}$.

To train this decoder network, we have to learn parameters $\theta$ to maximize the

log-likelihood of training data as:

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \sum_{d} \log p(x^{(d)};\theta) \\ &= \arg\max_{\theta} \sum_{d} \log \int p(x^{(d)}, z;\theta) dz \\ &= \arg\max_{\theta} \sum_{d} \log \int p(x^{(d)}|z;\theta) p(z) dz\end{aligned} \tag{1}$$

From the equation above, we can see that in order to compute the marginal distribution we have to compute the integral for z, for which we have to compute $p(x|z)$ for each z.

However, in practice in order to express more information, we usually assume $z$ to be a continuous variable (eg. Gaussian). But it is intractible for a computer to compute an infinite sum! Moreover, we do not have any information about the form of $p(x|z)$ since it is just a distribution parameterized by neural network.

### 1.2.2 Monte Carlo Sampling Method

A common way to solve the intractability problem is implementing the Monte Carlo Sampling Method. It helps to replace the intergral term by the expectation of latent variable $z$:

$$\int p(x^{(d)}|z;\theta) p(z) dz = \mathbb{E}_{z\sim p(z)}[p(x^{(d)}|z;\theta)] \tag{2}$$

We can do this because according to the MC sampling method, expectation of a function $f(z)$ can be considered as an average of multiple sampling for $z \sim p(z)$ i.e. $\mathbb{E}_{z\sim p(z)}[f(z)] = \frac{1}{m}\sum_{i=1}^{m} f(z_i)$. By applying this to (2) we get:

$$\int p(x^{(d)}|z;\theta) p(z) dz = \frac{1}{m}\sum_{i=1}^{m} p(x^{(d)}|z_i) \tag{3}$$

### 1.2.3 Approximation of Posterior

But here comes another question: how to choose the number of sampling $m$ to achieve a good approximation? To answer this question, we introduce here another function of z:

$$f_x(z) = p(x = x_i|z) \tag{4}$$

For each given $x$, it is a function of the latent variable $z$. Intuitively, if the model is capable of reconstructing our data $x_i$ with fewer times of sampling $m$, this function $f_x(z)$ of z should be closer to the posterior distribution $p(z|x;\theta)$ where

$$p(z|x;\theta) = \frac{p(x|z;\theta)p(z)}{p(x;\theta)} \tag{5}$$

However, the denominator term is our final goal of the learning problem, so it is impossible to compute the posterior directly. Instead, we propose another distribution $q(z|x;\Phi)$ to approximate $p(z|x;\theta)$. Besides, we have a well-defined metric to measure the discrepancy of two distribution named KL-divergence:

$$\begin{aligned} & KL[q(z|x;\Phi)||p(z|x;\theta)] \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log q(z|x;\Phi) - \log p(z|x;\theta)] \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log q(z|x;\Phi) - \log \frac{p(x|z;\theta)p(z)}{p(x;\theta)}] \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log q(z|x;\Phi) - \log p(x|z;\theta) - \log p(z)] + \log p(x;\theta) \end{aligned} \tag{6}$$

On top of only maximizing our log-likelihood, we also want to minimizing the KL-divergence above, so we reorganize our objective function as :

$$\begin{aligned} & \log p(x;\theta) - KL[q(z|x;\Phi)||p(z|x;\theta)] \\ &= \log p(x;\theta) - \mathbb{E}_{z\sim q(z|x;\Phi)}[\log q(z|x;\Phi) - \log p(x|z;\theta) - \log p(z)] - \log p(x;\theta) \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log p(x|z;\theta) - (\log q(z|x;\Phi) - \log p(z))] \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log p(x|z;\theta)] - \mathbb{E}_{z\sim q(z|x;\Phi)}[\log q(z|x;\Phi) - \log p(z)] \\ &= \mathbb{E}_{z\sim q(z|x;\Phi)}[\log p(x|z;\theta)] - KL[q(z|x;\Phi)||p(z)] \end{aligned} \tag{7}$$

The KL divergence subtracted from the log-likelihood is always greater than or equal to zero (it equals to zero only when the two distributions $q(z|x;\Phi)$ and $p(z|x;\theta)$ are identical in the sense of probability).

Consequently, we can define the expression (7) as a variational lower bound of log-likelihood (ELBO), and the problem of maximizing log-likelihood can then be converted to maximizing the ELBO .

## 1.3 Learning Problem

Equipped with our encoder and decoder networks, we now transform our learning problem of maximizing log-likelihood into maximizing the ELBO $\mathcal{E}$:

$$\hat{\theta}, \hat{\Phi} = \arg\max_{\theta,\Phi} \sum_{d} \mathcal{E}(\theta, \Phi, x^{(d)}) \tag{8}$$

To give further explanation, for each input data $x^{(d)}$ we have the expression of ELBO as:

$$\mathcal{E}(\theta, \Phi, x^{(d)}) = \mathbb{E}_{z\sim q(z|x^{(d)};\Phi)}[\log p(x^{(d)}|z;\theta)] - KL[q(z|x^{(d)};\Phi)||p(z)] \tag{9}$$

The first term of ELBO is called the **reconstruction term**. It is actually our decoder network which gives $p(x|z;\theta)$, and we can compute this through sampling. The second term is a **KL-divergence term**, it simply tells that we should make the approximate posterior distribution $q(z|x;\Phi)$ close to the prior $p(z)$.

### 1.3.1 Optimization

Unlike other models with latent variables like GMM and SBN which optimize $\theta$ and $\Phi$ alternatively, VAE applies a joint optimization to upgrade all parameters at each step. For step $t+1$ of mini-batch Gradient Descent, we will have:

$$\begin{aligned}\theta^{(t+1)} &= \theta^{(t)} + s\nabla_{\theta^{(t)}}\mathcal{E}(\theta^{(t)}, \Phi^{(t)}, x) \\ \Phi^{(t+1)} &= \Phi^{(t)} + s\nabla_{\Phi^{(t)}}\mathcal{E}(\theta^{(t+1)}, \Phi^{(t)}, x)\end{aligned} \tag{10}$$

where $s$ is the learning rate that can be chose by ourselves.

# 2 Comparisons of VAE and SBN

## 2.1 Brief Introduction of SBN

SBN, which is abbreviated for Sigmoid Belief Network, is literally a Belief Network implementing a sigmoid function for each node. Its joint distribution can be written as follows:

$$\begin{aligned}&p(V^{(1)} = v_1, \ldots, V^{(m+n)} = v_{m+n}) \\ =\ &p(v_1, \ldots, v_{m+n}) \\ =\ &\prod_{i=1}^{m+n} p(v_i | \{v_j; j \in Pa(i)\}) \\ =\ &\prod_{i=1}^{m+n} \sigma(b_i + \sum_{j \in Pa(i)} w_{ij} v_j)\end{aligned} \tag{11}$$

To train a SBN, we would also think about maximazing the log-likelihood by learning parameters. Let's simply rewrite the likelihood as below:

$$p(\{x_i\}_i) = \sum_{\{z_i\}_i} p(\{x_i\}_i; \{z_i\}_i) \tag{12}$$

Unlike VAE, here we have a sum of a finite number of PMFs since they are all Bernoulli variables. Nevertheless, as the size of latent variables gets larger there would still be intractability problem: the amount of calculation would be $2^{|z|}$. Similarly, the same problem for the Expectation Step when computing the denominator of posterior:

$$p(\{z_i\}_i | \{x_i\}_i) = \frac{p(\{x_i\}_i; \{z_i\}_i)}{\sum_{\{z_i\}_i} p(\{x_i\}_i; \{z_i\}_i)} \tag{13}$$

Therefore, we introduce again the variational method to propose another distribution $q(\cdot|\{x_i\}_i;\Phi)$ to approxiamte the true prior $p(\{z_i\}_i|\{x_i\}_i;\theta)$ .

$$\hat{\Phi} = \arg\min_{\Phi} KL[q(\cdot|\{x_i\}_i;\Phi)||p(\{z_i\}_i|\{x_i\}_i;\theta)] \tag{14}$$

Instead of maximizing the log-likelihood we can thus minimize the KL-divergence or maximize the ELBO. And it implements a iterative procedure to train the model:

$$\begin{aligned} \Phi^{(t+1)} &= \arg\min_{\Phi} KL(q|p) \\ \theta^{(t+1)} &= \theta^{(t)} + s\nabla_{\theta^{(t)}} \sum_d \mathcal{E}^{(d)}(\theta^{(t+1)}, \Phi^{(t)}, x^{(d)}) \end{aligned} \tag{15}$$

### 2.2 Differences between VAE and SBN

As we can see, both of VAE method and SBN propose distributions to approximate the posterior distribution and optimize the lower bound of log-likelihood (ELBO). But what makes them different are the following:

- VAE assumes that all the variables follow the same distribution, thus proposes a single parameterized distribution $q(\{z_i\}_i|\{x_i\}_i;\Phi)$ for the encoder network. Yet for SBN, it proposes a unique distribution $q^{(i)}(\cdot|x_i;\Phi)$ for each given $x_i$.
- For SBN, we have to train the network separately for parameters $\Phi$ and $\theta$. But for VAE it is an end-to-end learning process. All of the parameters will be updated at the same time.

## 3 Distribution Family of Variables

### 3.1 Latent Variables

Before constructing our networks, we need to make assumptions for latent variables about which distribution family comes from the proposed distribution $q(\{z_i\}_i|\{x_i\}_i;\Phi)$. Two common distributions are Gaussian and Bernoulli.

#### 3.1.1 Gaussian Distribution

In most of the cases, we suppose that $z$ follows a Gaussian distribution (but different latent variables $z$ are independent, which means the correlation coefficients equal to 0 and the covariance matrix is diagonal). So from the output of the encoder network we obtain the mean vector $\mu_{z|x}$ and the variance $\sigma^2_{z|x}$ for each $z$. Then when doing sampling we will sample z such that

$z|x \sim \mathcal{N}(\mu_{z|x}, \sigma^2_{z|x})$, from which $z$ should be then passed on to the input of the decoder network.

However, when training the model we have to compute the gradient of $\Phi$. But if we sample directly from $\mathcal{N}(\mu_{z|x}, \sigma^2_{z|x})$, we will only get the value of the sample thus lose the information of gradients.

So in order to remove the stochastic node from the path of gradient, we implement the **reparameterization** trick: we sample from another random variable $\epsilon \sim \mathcal{N}(0, I)$ each time. Finally we can combine the output of encoder with a sampling $\epsilon$ to represent a sample of $z$ as:

$$z = \epsilon \cdot \sigma_{z|x} + \mu_{z|x} \tag{16}$$

Hence the gradients $\frac{\partial z}{\partial \sigma}$ and $\frac{\partial z}{\partial \mu}$ can be computed via the reparameterization expression above.

### 3.1.2 Bernoulli Distribution

Another case is assuming that latent variables follow a distribution of Bernoulli ($z \sim \mathcal{B}(\mu)$). But Bernoulli variables have only PMFs, we can no longer apply the reparameterization trick. So in order to have differentiable sampling, we utilize another tool called the **Score Function Estimator**. It works when doing back-propagation for the encoder network:

$$\nabla_\Phi \mathcal{E}(\theta, \Phi, x) = \nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)] - \nabla_\Phi KL[q(\cdot|x;\Phi)||p(z)] \tag{17}$$

Expand the first term:

$$\begin{aligned}
\nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)] &= \nabla_\Phi q(z|x;\Phi) \log p(x|z;\theta) \\
&= \log p(x|z;\theta) \nabla_\Phi q(z|x;\Phi) \\
&= \log p(x|z;\theta) \frac{q(z|x;\Phi)}{q(z|x;\Phi)} \nabla_\Phi q(z|x;\Phi) \\
&= \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta) \frac{1}{q(z|x;\Phi)} \nabla_\Phi q(z|x;\Phi)] \\
&= \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta) \nabla_\Phi \log q(z|x;\Phi)]
\end{aligned} \tag{18}$$

In practice, we use MC sampling method to approximate the expectation operator:

$$\begin{aligned}
&\mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta) \nabla_\Phi \log q(z|x;\Phi)] \\
&\simeq \frac{1}{K} \sum_{i=1}^{K} \log p(x|z^{(i)};\theta) \cdot \nabla_\Phi \log q(z|x;\Phi)
\end{aligned} \tag{19}$$

The term after the multiplication sign is what we call the *Score Function Estimator*:

$$\nabla_\Phi \log q(z|x;\Phi) \tag{20}$$

which is basically the gradient of the log function of our proposed distribution.

Therefore, instead of the original loss function, now we have the loss function containing the log term of score function:

$$Loss = -logp(x|z;\theta) + KL[q(z|x;\Phi)||p(z)] - \log q(z|x;\Phi) \tag{21}$$

**Variance Reduction**

The one last problem of Score Function Estimator is that it has a large variance even though it is an estimation without bias. So we have to balance the term to reduce the variation. We introduce a function here:

$$f(x,z) = c(x)\log q(z|x;\theta)$$

where c(x) can be any function of x, and for $f(x,z)$ the gradient of $\Phi$ of its expectation should always be zero because:

$$\begin{aligned}
\nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[f(x,z)] &= \mathbb{E}_{q(z|x;\Phi)}[\nabla_\Phi (c(x)\log q(z|x;\Phi] \\
&= c(x)\sum_z q(z|x;\Phi)\frac{1}{q(z|x;\Phi)}\nabla_\Phi q(z|x;\Phi) \\
&= c(x)\nabla_\Phi \sum_z q(z|x;\Phi) \\
&= c(x)\nabla_\Phi 1 \\
&= 0
\end{aligned} \tag{22}$$

Thus we can rewrite the gradient of the reconstruction term based on (19) (22):

$$\begin{aligned}
&\nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)] \\
&= \nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)] - \nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[f(x,z)] \\
&= \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)\nabla_\Phi \log q(z|x;\Phi)] - \mathbb{E}_{q(z|x;\Phi)}[c(x)\nabla_\Phi \log q(z|x;\Phi)] \\
&= \mathbb{E}_{q(z|x;\Phi)}[(\log p(x|z;\theta) - c(x))\nabla_\Phi \log q(z|x;\Phi)]
\end{aligned} \tag{23}$$

Since $c(x)$ can be any function of $x$, we can choose it to be the average/moving average of $\log p(x|z;\theta)$. It works for reducing the variance because it can be considered as a kind of normalization.

## 3.2 Observed Variables

For the observed variables $\{x_i\}_i$, mostly, we assume that they follow a Gaussian distribution. Because most of the time, the tasks applying VAE are tasks like regenerating images. The input and output are supposed to be multiple-level pixels rather than pixels with only binary values because a broader range of pixels provides more information.

But of course, we may have observed variables assumed to be Bernoulli in many other applications. Under this circumstance, we have to apply an activation function like sigmoid or tanh for the decoder network to simulate the probability to be activated ($x = 1$). The only parameter to learn will be the $\mu_{x|z}$ of Bernoulli distribution.

Reminding us of SBN, all the variables in its network, including latent variables and observed variables, are Bernoulli. And to reduce the complexity, it is trained with Mean Field theory, which assumes that all latent variables are conditionally independent given observed variables. On the other hand, SBN with different family distributions than Bernoulli would no longer implement a sigmoid function for each node.

# 4 Implementation of VAE

To implement a Variational Auto-Encoder, there are several points to note:

1) Given the assumption of distribution family of latent features, we can calculate the analytic expression of KL-divergence. It should be a function of parameters $\theta$.

2) For a proposed Gaussian distribution $q(z|x;\Phi)$, we assume that $z_i$ are independent variables. Hence for the covariance matrix $\Sigma_{z|x}$, the correlation coefficients should be 0. But for those $\sigma$, they should be strictly greater than 0. To remove the constraint of being positive we can compute $\log\sigma$ instead of $\sigma$, because $e^{\log\sigma} = \sigma > 0$.

3) When constructing the networks, the return values should be unconstrained. i.e., it does not directly return the $\mu$ of the Bernoulli distribution, but values in $\mathbb{R}$. And we have to call the sigmoid function $\sigma(\mu)$ to transform the unconstrained values to the $\mu$ parameters.

4) Utilize the reparameterization trick for continuous latent space (Gaussian) and a score function estimator for binary latent space (Bernoulli).

5) For a binary latent space ($z \sim \mathcal{B}(\mu)$), when doing the back-propagation for ELBO, we want to utilize the score function and thus transform the gradient of the reconstruction term as:

$$\nabla_\Phi \mathbb{E}_{q(z|x;\Phi)}[\log p(x|z;\theta)] \simeq \frac{1}{K}\sum_{i=1}^{K} \log p(x|z^{(i)};\theta)\nabla_\Phi \log q(z^{(i)}|x;\Phi)$$

here we want to keep the value of $\log p$ and compute only the gradient of $\log q$. In PyTorch, we can implement the function **detach()** to the log_p variable in order to not compute its gradient later.

# 5 Results of Experiments

In this work, we used the **MNIST** dataset. We then binarized the original dataset because we wanted our observed random variables to follow a **Bernoulli distribution**, i.e., each pixel can either be black or white. There are 50000 images in our training set and each image is of size 28x28.

We first trained two VAE models: one with continuous latent space and binary observed space, the other with binary latent space and binary observed space. Both of them have the following generative story:

1. $z \sim p(z)$
2. $x \sim p(x|z;\theta)$

Also, we trained a deterministic Auto-Encoder with the same training set and turned it into a generative model by applying the following step:

1) train a deterministic Auto-Encoder with a 2 dimension latent space
2) train a GMM on the latent space
3) sample a cluster from the GMM: $y \sim p(y)$
4) sample a point from the bivariate Gaussian associated with this cluster: $z \sim p(z|y)$
5) use the decoder to sample an image: $x \sim p(x|z)$

## 5.1 VAE with continuous latent space

In this part, we assume that the latent random variable $Z$ takes value in $\mathbb{R}^n$. The prior distribution $p(z)$ is a multivariate Gaussian where each coordinate is independent. We fix the mean and variance of each coordinate to 0 and 1, respectively. The conditional distribution $p(x|z;\theta)$ is parameterized by a neural network. The random variables $X$ are $m$ independent Bernoulli random variables.

### 5.1.1 Neural Architecture

Our neural networks are constructed in the following way.

**Encoder:**

From input $x$ to the hidden layer $h_1$, we have a full linear connection and a non-linear relu connection:

$$h1 = \max(W_1 \cdot x + b_1, 0) \tag{24}$$

From hidden layer to the output of encoder network:

$$y1 = \begin{pmatrix} \mu_x \\ \sigma_x \end{pmatrix} = \begin{pmatrix} W_{21} \cdot h1 + b_{21} \\ W_{22} \cdot h1 + b_{22} \end{pmatrix} \tag{25}$$

**PriorDecoder:**

We have a full linear connection with a relu to the hidden later, then a full linear connection to the output:

$$\begin{aligned} h2 &= \max(W_3 \cdot y_1 + b_3, 0) \\ \hat{x} &= W_4 \cdot h_2 + b_4 \end{aligned} \tag{26}$$

Since we assumed that our observed variable $x$ is a Bernoulli, the PriorDecoder network's output should be its parameter $\mu_{x|z}$. To achieve this, we normalize the output to a certain probabilistic value between 0 and 1 by performing a *sigmoid()* function.

### 5.1.2 Results

First we visualize the latent space by displaying the mean values:

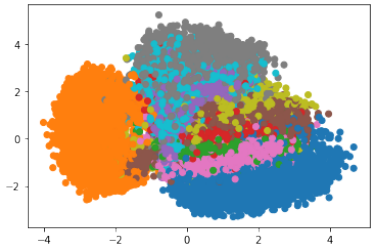


Figure 3: Latent space

From the result we can see that almost all the points are close to 0 and there are several clusters of the same color, which means some of the classes are well delimited, but there also exist some classes with mean values really close to 0 are mixed together.

Then we generate some new distributions of random variable $X$ by applying the trained decoder on some new samples from the latent space. The new images are then generated by sampling from the output distribution or directly using the most probable value for each random variable. In this work, we display the output image with the most probable value.

From the results we can see that the VAE with continuous Gaussian latent space works pretty well on this task. Even though the generated distributions are a bit blurry, we can still observe the shape of the numbers. And the binary images

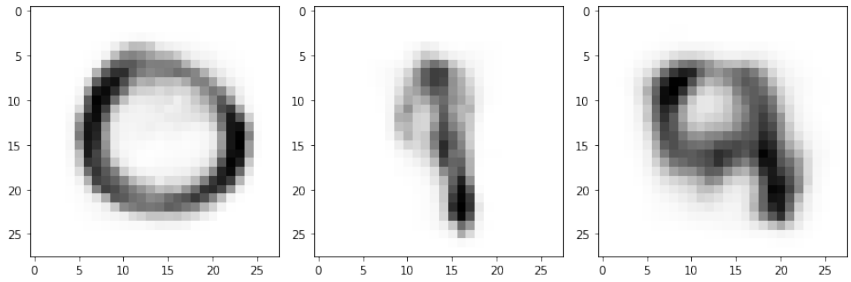

Figure 4: Distributions generated by VAE with Gaussian

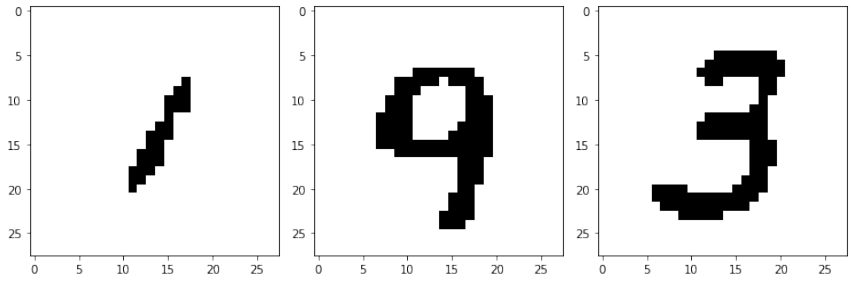

Figure 5: Image generated with most probable value

generated with most probable value (which are actually the final output of the algorithm) looks very close to actual handwritten numbers.

## 5.2 VAE with binary latent space

In this part, we assume that the prior distribution $p(z)$ is a multivariate Bernoulli where each coordinate is independent, the mean of each coordinate is fixed to 0.5, so we have $z_i \sim \mathcal{B}(0.5)$. The latent random variable $Z$ takes value in $[0, 1]$ which is sampled from the corresponding Bernoulli coordinate. The conditional distribution $p(x|z;\theta)$ is parameterized by a neural network. The random variables $X$ are $m$ independent Bernoulli random variables.

### 5.2.1 Neural Architecture

**Encoder:**

From input $x$ to the hidden layer $h_1$, we have a full linear connection and a non-linear relu connection:

$$h1 = \max(W_1 \cdot x + b_1, 0) \tag{27}$$

From hidden layer to the output of encoder network:

$$y1 = p_z = W_2 \cdot h1 + b_2 \tag{28}$$

**PriorDecoder:**

We have a full linear connection with a relu to the hidden later, then a full linear connection to the output:

$$\begin{aligned} h2 &= \max(W_3 \cdot y_1 + b_3, 0) \\ \hat{x} &= W_4 \cdot h_2 + b_4 \end{aligned} \tag{29}$$

Also, the outputs of the network (encoder & decoder) should be normalized to certain probabilistic values between 0 and 1 by performing a *sigmoid()* function.

### 5.2.2 Results

First we visualize the latent space by displaying the mean values of the Bernoulli distribution (which represent $p_x = 1$):

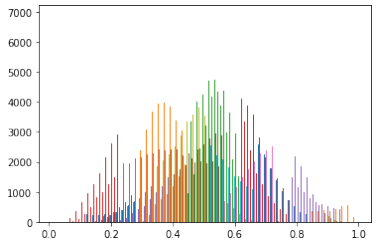


Figure 6: Latent space

As the latent space was constructed by 64 latent Bernoulli distribution, we simply display all the mean values with a 1d histogram. From the result we can see that almost all the points are close to 0.5. And we still can observe some 'clusters' of the same color, which represent the well delimited classes, but in general most of the classes are mixed together.

Then we generate some new distributions of random variable $X$, and finally some new images with the most probable value.

From the results we can see that the VAE with binary Bernoulli latent space is not very suitable for solving this task. We can still observe the shape of the numbers from the generated distributions, but the contours are very unclear. the algorithm is almost failed when we try to generated some new binary images with most probable value. The experimental results are in line with our inference that the binary latent space can not carry as much information as the continuous latent space.

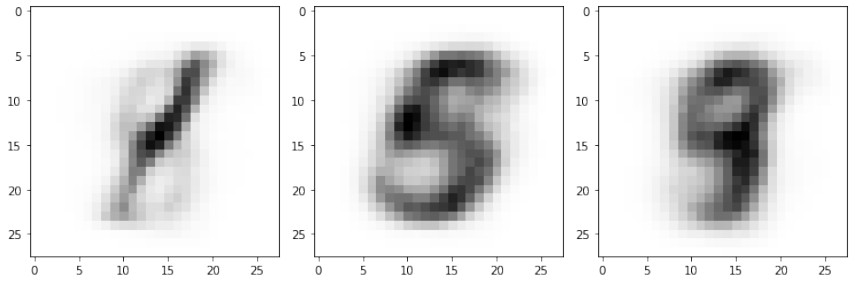

Figure 7: Distributions generated by VAE with Bernoulli

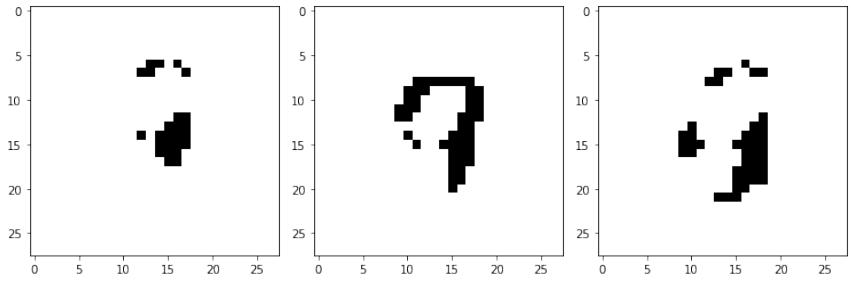

Figure 8: Image generated with most probable value

## 5.3 Deterministic Auto-Encoder + GMM

With our deterministic Auto-Encoder, an image is projected in a 2 dimension latent space. In other words, each image with $28 \times 28$ pixels is represented by a point in the $\mathbb{R}^2$ space (this is actually a task of dimension reduction). Then a GMM model is trained base on these points in latent space, by sampling new points from the GMM model and expand the dimension by applying decoder, we could generate new images.

### 5.3.1 Neural Architecture

The deterministic Auto-encoder network was constructed in the following way.

**Encoder:**

From input $x$ to the hidden layer $h_1$, we have a full linear connection and a non-linear relu connection:

$$h1 = \max(W_1 \cdot x + b_1, 0) \tag{30}$$

From hidden layer to the output of encoder network:

$$y1 = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} W_{21} \cdot h1 + b_{21} \\ W_{22} \cdot h1 + b_{22} \end{pmatrix} \tag{31}$$

Where $(\alpha, \beta)$ represent a point in the $\mathbb{R}^2$ space.

**Decoder:**

We have a full linear connection with a relu to the hidden later, then a full linear connection to the output:

$$\begin{aligned} h2 &= \max(W_3 \cdot y_1 + b_3, 0) \\ \hat{x} &= W_4 \cdot h_2 + b_4 \end{aligned} \tag{32}$$

### 5.3.2 Results

As above, we first visualize the latent space, which contains the 2d points projected from the image by the encoder network. We also visualize the GMM model trained on this latent space. The number of components is set to be 30 after several experiments.

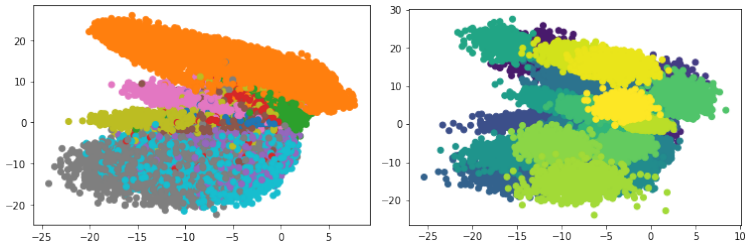


Figure 9: 2d points in latent space / GMM model

As we can see from the plot of latent space, the 2d points also form several clusters. And the resulting points cloud of GMM looks very similar to the 2d points cloud, which means our GMM model can well represent the latent space. This GMM model can then be used to generate new points.

Finally, we pass the points sampled by GMM model through the decoder network to generate some new distributions of random variable $X$, then create some new images with the most probable value.

From the results we can obseve that Deterministic Auto-encoder + GMM model works even better than VAE with continuous Gaussian latent space. Both the generated distributions and the binary images are very similar to the real handwritten numbers. This result may tell us that for this specific dataset, the dimension reduction based latent space may carry more useful information than the latent space which contains the proposed posterior distribution.

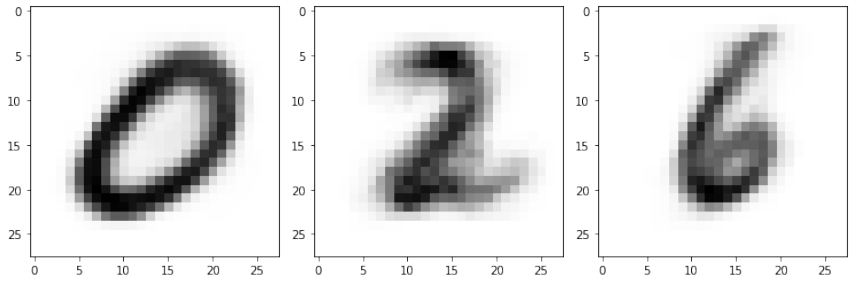

Figure 10: Distributions generated by Det. Auto-encoder + GMM model

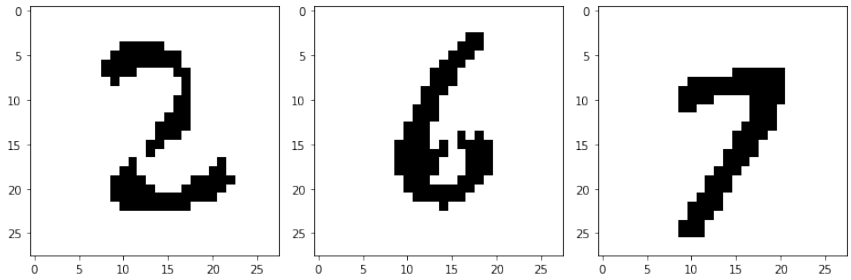

Figure 11: Image generated with most probable value

# 6 Conclusion

By doing this lab work, we've better understood the principle of the VAE model and the difference between VAE and SBN. We've also implemented (part of) 3 different models in practice, VAE with continuous Gaussian latent space, VAE with binary Bernoulli latent space, model based on deterministic Auto-encoder + GMM, which forced us to think more in-depth with some details in the algorithm. For example, the reparameterization trick, the score function estimator, etc. By doing the experiments, we also realized that there exist some trade off in the loss function of VAE between the log-likelihood and the KL term, different weights for these two terms can also lead to very different results.