Jingzhuo Hui
Damoun Ayman
VAE github repository
Master M2 IA

TCP4
Deep Probabilistice Generative Models
Caio Filippo Corro
November 4, 2020

Lab 1: Deep Probabilistice Generative Models: Variational Auto-Encoders

0.1 Preliminaries

Generative modelling is an active area of deep learning research that centers around realistic data generation. In this lab we will discover the principles of a generative model and learn how to build such models in neural networks. We will mainly focus on Variational Auto-Encoders.

0.2 Generative modeling

A generative model take as input training samples from some distribution and learn a model that represents that distribution. The figure 1 list the deep generative models that can learn via the principle of maximim likeli-hood differ with respect to how they represent or approximate the likelihood.

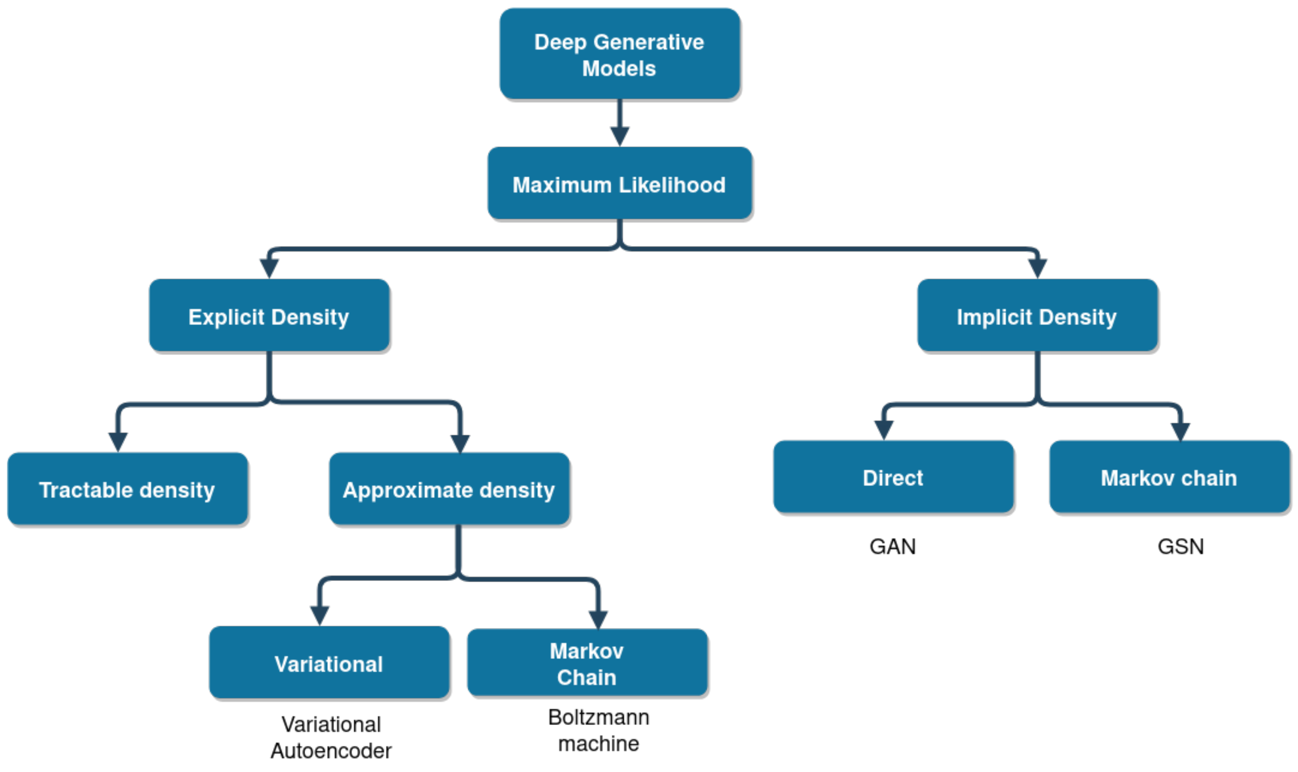


Figure 1: Taxonomy of Generative models. Reproduced from Goodfellow [1]

Maximum likelihood is identified as a central task around which generative models can be compared. Not all generative models use maximum likelihood.

$$\theta^* = \arg\max \mathbb{E}_{x\sim p_{\text{data}}} \log P_{\text{model}}\left(x/\theta\right) \tag{1}$$

In Eq.1, x is a vector describing the input, $P_{\text{model}}\left(x/\theta\right)$ is a density function controlled by the parameter θ. Maximum likelihood consists of measuring the log probability that the density function assigns to all

data points and adjusting the parameter θ to increase that probability.
A model is generative if it places a joint distribution over all observed dimensions of the data.
Consider a learning task with features X and labels Y:

- Generative models want to learn $P(X, Y)$.
- Discriminative models want to learn $P(Y|X)$.

How to design a rich family of probability distributions?
Three basic recipes for using a flexible function $f_\theta()$:

1. Apply a richly parameterized transformation to a simple random variable.

$$Z \sim \mathcal{N}(0, \mathbf{I}) \qquad X = f_\theta(Z)$$

2. Use a rich mixing distribution for a simple parametric family.

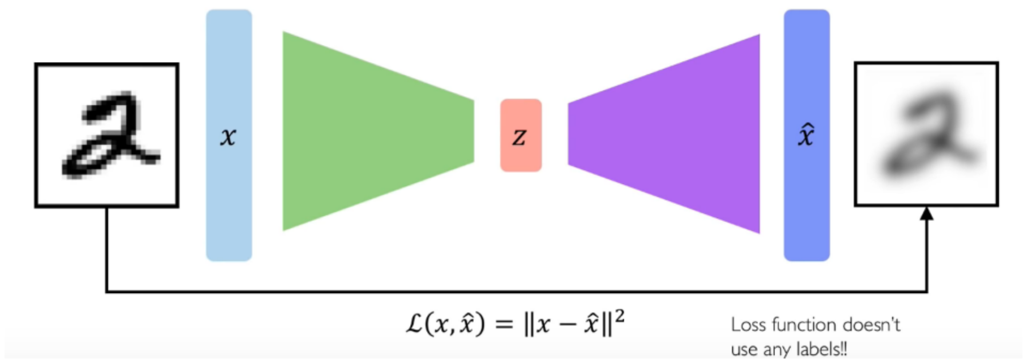
$$Z \sim \mathcal{N}(0, \mathbf{I}) \qquad X = \mathcal{N}(f_\theta(Z), \Sigma)$$

3. Specify a complicated distribution via its log density:

$$X \sim \frac{1}{\mathcal{Z}_\theta} \exp\{f_\theta(x)\} \quad \mathcal{Z}_\theta = \int \exp\{f_\theta(x)\}\, dx$$

0.3 The variational autoencoder

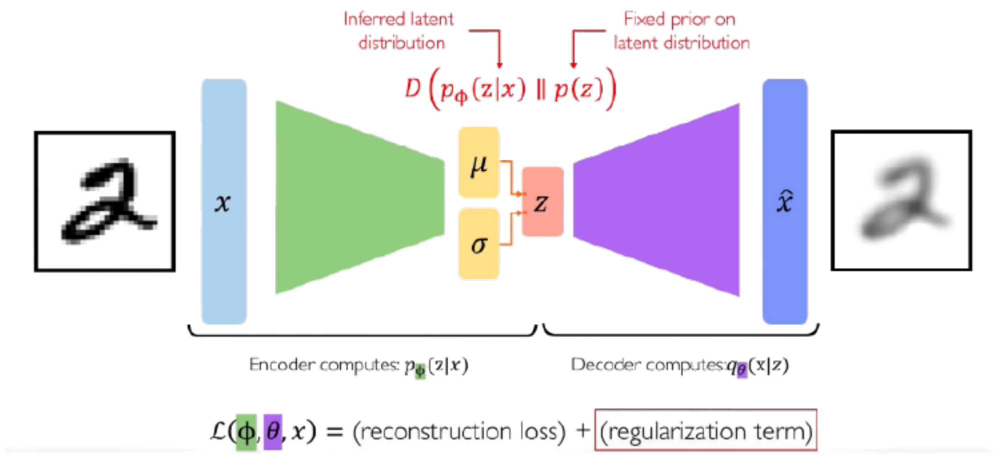
A standard autoencoder consists of an encoder and a decoder. Let the input data be X. The encoder produces the latent space vector z from X. Then the decoder tries to reconstruct the input data X from the latent vector z. the figure 2 below shows the shape of an autoencoder. the autoencoder has **battleneck hidden layer** that forces network to learn a compressed latent representation.



x
z
$\hat{x}$
$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$
Loss function doesn't use any labels!!

Figure 2: The working of a simple deep learning autoencoder model (Traditional autoencoders).

The idea of Variational Autoencoder [2], short for VAE, is based on the same principle of autoencoder models above, but deeply rooted in the methods of variational bayesian and graphical model. During training, VAEs force this normal distribution to be as close as possible to the standard normal distribution by including the Kullback–Leibler divergence in the loss function.
To summarize, variational autoencoders (VAEs) are autoencoders that tackle the problem of the latent space irregularity by making the encoder return a distribution over the latent space instead of a single point and by adding in the loss function a regularisation term over that returned distribution in order to ensure a better organisation of the latent space and avoid overfitting.



Inferred latent distribution
Fixed prior on latent distribution
$D\left(p_\phi(z|x) \parallel p(z)\right)$
x
μ
σ
z
$\hat{x}$
Encoder computes: $p_\phi(z|x)$
Decoder computes: $q_\theta(x|z)$
$\mathcal{L}(\phi, \theta, x)$ = (reconstruction loss) + (regularization term)

Figure 3: Variational autoencoder .

0.4 MNIST classification with Pytorch

The MNIST database is a database of handwritten digits that is commonly used for training and testing in the field of machine learning and computer vision. The MNIST database contains 60,000 training images and 10,000 testing images. Each image is of size 28×28.

Question 1
Can you write mathematically the function computed by the classifier network? what are the dimensions of each parameter?

solution: The architecture of the neural network is presented below

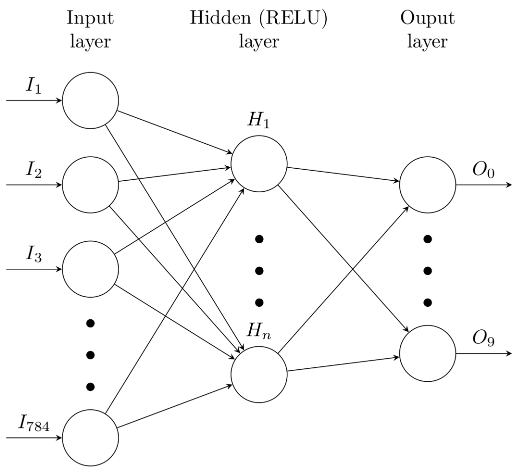


Figure 4: the classifier network

```python
class Classifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_ratio=0.5):
        super().__init__()
        self.proj1 = nn.Linear(input_dim, hidden_dim)
        self.proj2 = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(0.5)
    def forward(self, batch):
        hid = self.proj1(batch)
        hid = F.relu(hid)
        hid = self.dropout(hid)
        return self.proj2(hid)
```

Where:

- input dim: in our case it will the size of a single input image (28*28=784)
- hidden dim: the hidden dim is define when when calling the model
- output dim: the number of class, in our case there are 10 digits

As activation function the rectified linear units (ReLUs in short) is chosen. In PyTorch a nice way to build a network is by creating a new class for the network we wish to build.

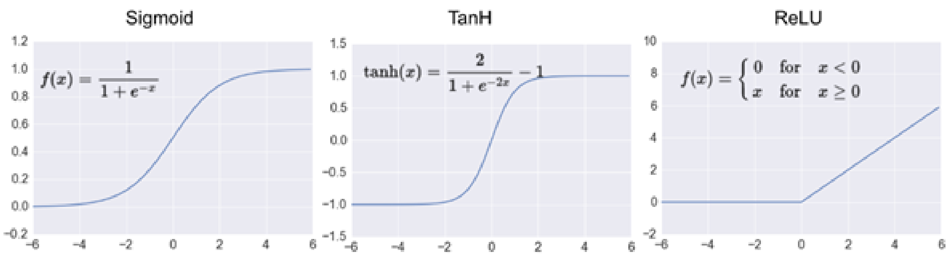


Figure 5: activation

0.5 VAE with continuous latent space and binary observed space

Generative Process

Let us consider a dataset $X = \{x_i\}_{i=1}^{N}$ i.i.d. consisting of N samples of some continuous or discrete variable x. We assume that the data are generated by some random process, involving an unobserved continuous randmo variable z. The process consists of two steps:

1. A value $\boldsymbol{z}_i$ is generated from some prior distribution $p_{\theta^*}(\boldsymbol{z})$.
2. A value $\boldsymbol{x}_i$ is generated from some conditional distribution $p_{\theta^*}(\boldsymbol{x}|\boldsymbol{z})$.

We assumed that the prior $p_{\theta^*}(\boldsymbol{z})$ and likelihood $p_{\theta^*}(\boldsymbol{x}|\boldsymbol{z})$ come from parametric families of distributions $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, and that their PDFs are differentiable almost everywehree w.r.t. both θ and $\boldsymbol{z}$. Unfortunatey, a lot of this process is hidden from our view, i.e. both the true parameters θ^* and the values of the latent variables $\boldsymbol{z}_i$ are unknown. To summarise:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}\ ; \mathbf{0}, \mathbf{I}) \tag{2}$$

$$\boldsymbol{x} \sim, \mathcal{N}(f_\theta(z), \Sigma) \tag{3}$$

Mean Field Approximation

In this type of variational inference, we assume the variational distribution over the latent variables factorises as

$$q(z_1, ..., z_m) = \prod_{j=1}^{m} q(z_j) \tag{4}$$

We refer to $q(z_j)$, the variational approximation for a single latent variable, as a ”local variational approximation”

Inference

The objective is to approximate the true posterior distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)$ which in this case is intractable. While there is much freedom in the form of $q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)$, we will assume that the true but intractable posterior takes on an approximate Gaussian form with an approximately diagonal covariance. In order to approximate $p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)$, we use the aforementioned mean field variational approximation. In the setting of the variational autoencoder, the variational posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)$ is also parameterised by a neural network f_ϕ which takes input $\boldsymbol{x}_i$ and outputs the mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\sigma}_i^2$ of the approximate posterior Normal distribution:

$$\boldsymbol{\mu}(\boldsymbol{x}_i;\phi),\ \log\boldsymbol{\sigma}^2(\boldsymbol{x}_i;\phi) = f_\phi(\boldsymbol{x}_i) \tag{5}$$

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) = \mathcal{N}(\boldsymbol{z}\ ;\ \boldsymbol{\mu}(\boldsymbol{x}_i;\phi),\ \boldsymbol{\sigma}^2(\boldsymbol{x}_i;\phi)\mathbf{I}) \tag{6}$$

Let the prior over the latent variables be the centered isotropic multivariate Gaussian $p_\theta(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}\ ;\mathbf{0},\mathbf{I})$.

ELBO (Evidence Lower Bound)

The true posterior has the following form:

$$\begin{aligned} p_\theta(\boldsymbol{z}|\boldsymbol{x}_i) &= \frac{p_\theta(\boldsymbol{z},\boldsymbol{x}_i)}{p_\theta(x_i)} && (7)\\ &= \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)p_\theta(\boldsymbol{z})}{p_\theta(x_i)} && (8)\end{aligned}$$

Because directly optimising $\log(p_\theta(\boldsymbol{x}_i))$ is infeasible, we choose to optimise a lower bound $\mathcal{L}$ of it. The lower bound on the marginal likelihood of datapoint $\boldsymbol{x}_i$ can be written as:

$$\begin{aligned} \log(p_\theta(\boldsymbol{x}_i)) \geq \mathcal{L}(x_i;\theta,\phi) &:= E_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\left[\log p_\theta(\boldsymbol{x}_i,\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)\right] && (9)\\ &= E_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\left[\log p_\theta(\boldsymbol{x}_i|\boldsymbol{z})\right] - D_{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)\parallel p(\boldsymbol{z})\right) && (10)\end{aligned}$$

The KL term which acts as aregulariser can be integrated analytically in our case. For a general covariance matrix $\Sigma = (\sigma_{ij})_{i,j=1}^N$ and a fixed $i,j \in \{1,...,N\}$ we obtain the following analytical solution for the KL term:

$$\begin{aligned} D_{KL}(q_\phi(z_j|x_i)\parallel p_\theta(z_j)) &= E_{q_\phi(z_j|x_i)}\left[\log\left(\frac{q_\phi(z_j|x_i)}{p_\theta(z_j)}\right)\right] && (11)\\ &= E_{q_\phi(z_j|x_i)}\left[\log\left(q_\phi(z_j|x_i)\right)\right] - E_{q_\phi(z_j|x_i)}\left[\log\left(p_\theta(z_j)\right)\right] && (12)\\ &= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(\sigma_{ij}^2\right) - \frac{1}{2} - \left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\left(\sigma_{ij}^2+\mu_{ij}^2\right)\right) && (13)\\ &= \frac{1}{2}\left(\sigma_{ij}^2+\mu_{ij}^2\right) - \frac{1}{2}\log\left(\sigma_{ij}^2\right) - \frac{1}{2} && (14)\end{aligned}$$

Therefore the ELBO simplifies for a fixed $i,j \in \{1,...,N\}$ to:

$$\begin{aligned} \log(p_\theta(x_i)) &\geq E_{q_\phi(z_j|x_i)}\left[\log\left(p_\theta(x_i|z_j)\right)\right] - D_{KL}(q_\phi(z_j|x_i)\parallel p_\theta(z_j)) && (15)\\ &= E_{q_\phi(z_j|x_i)}\left[\log\left(p_\theta(x_i|z_j)\right)\right] - \left(\frac{1}{2}\left(\sigma_{ij}^2+\mu_{ij}^2\right) - \frac{1}{2}\log\left(\sigma_{ij}^2\right) - \frac{1}{2}\right) && (16)\\ &= E_{q_\phi(z_j|x_i)}\left[\log\left(p_\theta(x_i|z_j)\right)\right] - \frac{1}{2}\left(\sigma_{ij}^2+\mu_{ij}^2\right) + \frac{1}{2}\log\left(\sigma_{ij}^2\right) + \frac{1}{2} && (17)\end{aligned}$$

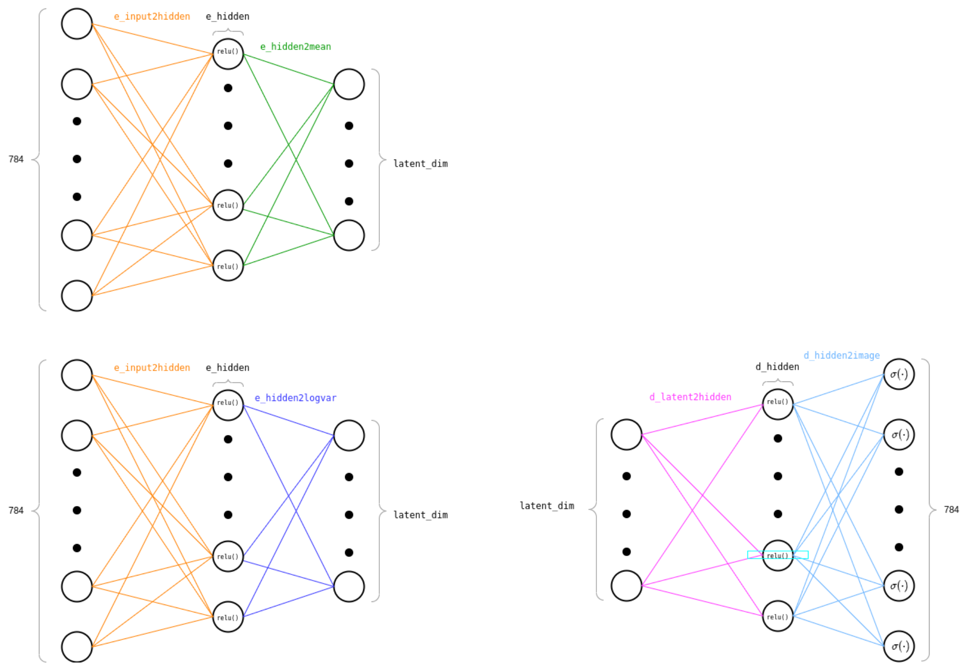
The expected reconstruction error $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ requires estimation using Monte Carlo by sampling from $q_\phi(z_i|x_i)$:

$$E_{q_\phi(z_i|x_i)}\left[\log\left(p_\theta(x_i|z_i)\right)\right] \simeq \frac{1}{L}\sum_{\ell=1}^{L}\log\left(p_\theta\left(x_i|z_i^\ell\right)\right), \qquad z_i^\ell \sim q_\phi(z_i|x_i) \tag{18}$$

According to the original paper [2], L can be set to 1 if the minibatch size M is large enough ($M \geq 100$). The ELBO over one batch can be calculated with:

$$\mathcal{L}(X;\theta,\phi) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{1}{L}\sum_{\ell=1}^{L}\log\left(p_\theta\left(x_i|z_i^\ell\right)\right) - D_{KL}(q_\phi(z_i|x_i) \parallel p(z_i))\right) \tag{19}$$

$$\overset{(ii)}{=} \frac{1}{M}\sum_{i=1}^{M}\left(\frac{1}{L}\sum_{\ell=1}^{L}\log\left(p_\theta\left(x_i|z_i^\ell\right)\right) - \frac{1}{2}\left(\sigma_{ii}^2 + \mu_{ii}^2\right) + \frac{1}{2}\log\left(\sigma_{ii}^2\right) + \frac{1}{2}\right) \tag{20}$$



(a) The architecture of the encoder (b) The architecture of the decoder

Figure 6: Variational autoencoder architecture

0.6 VAE with binary latent space and binary observed space

In this case, the corresponding prior distribution over the latent space is reduce to Bernoulli distributions. The prior $p(z)$ and the recognition model $q(z|y)$ are modeled as:

$$p(z) = \prod_{i=1}^{Q}\text{Ber}(z_i; 0.5); \qquad p(z|y) = \prod_{i=1}^{Q}\text{Ber}(z_i; \theta_i(y)) \tag{21}$$

where $\theta_i(y)$ are predicted using the encoder, given the input y. As before, the likelihood is to be optimized during training. As this may be infeasible, the evidence lower bound is optimized isntead. This means for a training sample

$$y_m : \mathcal{L}_{\text{VAE}}(w) = \text{KL}(q(z|y_m)|p(z)) - \frac{1}{L}\sum_{l=1}^{L}\ln p(y_m|z_{l,m}) \tag{22}$$

where $z_{l,m} \sim q(z|y)$. And the KL-divergence can be computed analytically, however, differentiation through the sampling process $z_{l,m} \sim q(z|y)$ is problematic- unfortunately.

0.7 Turning a Deterministic Auto-Encoder into a generative model

We get the point for encoder from a GMM, and we plot these point, found out that GMM not exactly fit with the latent space , cause the weight of GMM is not pre-defined.

After sampling data from the model , we got the pretty much similaier result with VAE.So we can see the turning of Deterministic Auto-Encoder into a generative model is well performed.

0.8 Conclusion

VAE is an amazing tool that can solve some challenging problems with the help of neural networks: generative models. Compared with traditional methods, VAEs solve two main problems:

1. How to extract the most relevant hidden variables from the hidden space to give the output.
2. How to map the data distribution in the hidden space to the real data distribution.

However, VAE also has some shortcomings: because it uses the root mean square error, it makes the generator converge to the average optimal, resulting in a little blur in the generated image.

Bibliography

[1] Ian J. Goodfellow. “NIPS 2016 Tutorial: Generative Adversarial Networks”. In: *CoRR* abs/1701.00160 (2017). arXiv: `1701.00160`. URL: `http://arxiv.org/abs/1701.00160`.

[2] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: `1312.6114 [stat.ML]`.