VAE- a basic introduction

In this article I'll cover the basis of VAE — its intuition and a simple formulation. Its full name is **Variational Auto-Encoder**, which comprises of two parts: **variational** and **auto-encoder**. In this article, the two terms will be talked about respectively and then be combined together as the probablistic model VAE.

What is autoencoder

The origin of autoencoder comes from a thought — if you can't generate it, you don't understand it. An autoencoder is such a generative model which takes datapoints $\{x_i\}_{i=1}^N$ as input, encodes it as a real-value vector z following a specific conditional distribution $q_\phi(z|x)$, decodes the hidden state z to regenerate a sample $\hat{x}$ using another distribution $p_\theta(x|z)$. It is a generative model, which can be used to both *represent* and *generate* a sample.

What is variational inference

For most probablistic models the true distribution is very complex and intractable. Variational inference is such a tool to make an approximation of the true distribution with regards to new parameters.

I haven't read enough materials about it. This part will be completed after I read more papers about it.

Variational Auto-Encoder

VAE, by its name, is an auto-encoder that applies variational inference. Why do we need variational method to help construct an auto-encoder? Let's first look at the objectives of an auto-encoder.

As an auto-encoder, it is expected to carry out two missions:

1. efficient approximation of posterior inference $p(z|x)$. This is for the recognition of a given sample x.
2. efficient approximation of marginal inference of the sample $p(x)$. This is for the purpose of generation.

For the first mission, we first assign a distribution to the posterior, usually a Gaussian distribution $z|x \sim N(\mu_x, \sigma_x I)$. For z itself, we also have to assign a distribution, similarly $z \sim N(0, I)$. Finally for the reconstruction a distribution is also needed. Let's make it as $x|z \sim N(\mu_z, \sigma_z I)$. Notice that the parameters like μ_x, σ_z can be a complex composition of other parameters, so we can parameterize them with a complex neural model and a set of parameters. Formally we denote

$q_\phi(z|x)$ as the encoder and $p_\theta(x|z)$ as the decoder.

We cannot give an analytical solution to the parameters as we have to integrate over a complex distribution. So what we can do is to set an objective function and update the parameters iteratively. Usually we use Maximum Likelihood Estimation(MLE) or Maximum A Posteriori(MAP). As the posteriori is not tractable here, we use MLE to carry out parameter estimation.

MLE formulation

We want to maximize the likelihood of sample datapoints seen in the dataset, namely $p(X)$, where $X = \{x_i\}_{i=1}^N$. For each datapoint x ,

$$\begin{aligned}
\log p(x) &= \int q_\phi(z|x) \log p(x) dz \\
&= \int q_\phi(z|x) \log \frac{p(x,z)}{p(z|x)} dz \\
&= \int q_\phi(z|x) \log [\frac{p(x,z)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p(z|x)}] dz \\
&= KL(q_\phi(z|x) \,||\, p(z|x)) \; + \; \int q_\phi(z|x) \log \frac{p(x,z)}{q_\phi(z|x)} dz
\end{aligned}$$

It is now seperated as two terms. The first term refers to the *quality* of the approximation of the posterior inference. As the KL divergence is always greater than or equal to 0, the rightmost term can be seen as a **lower bound** of the likelihood:

$$\log p(x) \geq \int q_\phi(z|x) \log \frac{p(x,z)}{q_\phi(z|x)} dz$$

We denote it as L_b. L_b can be further decomposed as:

$$\begin{aligned}
L_b &= \int q_\phi(z|x) \log \frac{p(x|z)p(z)}{q_\phi(z|x)} dz \\
&= KL(q_\phi(z|x) \,||\, z) \; + \; E_{z \sim q_\phi(z|x)}[\log p(x|z)]
\end{aligned}$$

The first term measures the deviance between our approximation and the distribution of z (as hypothesized, $N(0, I)$). This can be used to control the **variance** of our approximation, in order to avoid simply "memorizing" all the datapoints. It could also be viewed as a regularizer to keep away from overfitting. Take the Gaussian distribution we hypothesized before, we can directly compute this term with basic integral knowledge:

$$\begin{aligned}
KL(q_\phi(z|x) \,||\, z) &= \int q_\phi(z|x) \log p(z) dz - \int q_\phi(z|x) \log q_\phi(z|x) dz \\
&= \int N(z; \mu_x, \sigma_x I) \log N(z; 0, I) dz - \int N(z; \mu_x, \sigma_x I) \log N(z; \mu_x, \sigma_x I) \\
&= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{J} ((\sigma_x^i)^2 + (\mu_x^i)^2) + \frac{J}{2} \log(2\pi) + \frac{1}{2} \sum_{i=1}^{J} (1 + \log(\sigma_x^i)^2) \\
&= \frac{1}{2} \sum_{i=1}^{J} (1 + \log((\sigma_x^i)^2) - (\mu_x^i)^2 - (\sigma_x^i)^2)
\end{aligned}$$

The second term is not directly computable. We can use **Monte Carlo** mothod to estimate this term. First we have to sample z for l times according to its distribution $q_\phi(z|x)$. For this step we can use a differentiable transformation $g_\phi(\epsilon, x)$ to represent this sampling process. Recall that $z|x \sim N(\mu_x, \sigma_x I)$, we sample z as $z = \mu_x + \sigma_x \circ \epsilon$, where $\epsilon \sim N(0, I)$. For the computation of $p(x|z)$ we parameterize it with a model called *decoder* and a set of parameters θ. Thus the second term is computed as:

$$E_{z\sim q_\phi(z|x)}[p(x|z)] = \frac{1}{L}\sum_{i=1}^{L} \log p_\theta(x_i|\mu_x + \sigma_x \circ \epsilon_i),\ \epsilon_i \sim N(0, I)$$

Now we can iteratively update our parameters and increase L_b as it is differentiable under our approximation. In a commonsense, if we continuously increase the lower bound L_b, we'll get a better estimation of $p(z|x)$, because $p(x)$ is a constant and $KL(q_\phi(z|x)||p(z|x))$ will be decreasing.

However, it is not the most accurate depiction of the object. Remind this formulation:

$$\log p(x) = KL(q_\phi(z|x)\ ||\ p(z|x)) + L_b$$

When we compute $p(x|z)$ above, we actually utilize an approximation of this distribution, and it is factorized from $p(x)$. So, the $\log p(x)$ is actually $\log p_\theta(x)$, and it changes as we update θ.

In my understanding, the purpose of jointly update θ and ϕ with respect to L_b is to provide a greater **lower bound** of the likelihood $\log p(x)$. More specificly, $\log p_\theta(x)$ is an approximation of $\log p(x)$, so there must be a gap between them. To update θ is to minimize the gap between $\log p_\theta(x)$ and $\log p(x)$. To update ϕ is to provide a greater lower bound **under the approximation** $\log p_\theta(x)$. They two jointly lift up the global lower bound of $\log p(x)$.