Variational Autoencoder (VAE)

Introduction:

In the past few years, deep learning based generative models which rely on huge amount of data, smart training methods and well-designed architectures, have shown their extraordinary ability to generate highly realistic data including images, text and speech. Among them, the most familiar and popular models Generative Adversarial Network (GANs) and Variational Autoencoders (VAE)

GANs consists of two networks, a Generator and a Discriminator and performs adversarial training to push both of them to improve iteration after iteration. However, VAE is an autoencoder whose encodings distribution is regularized during the training to ensure that the latent space has good properties. This regularized latent distribution helps the VAE to generate the input. Let's dive into VAE.

What is VAE?

VAE stands for Variational Auto Encoder. It is a generative model which produces new unseen data. Unlike the normal Autoencoder, VAE focuses on understanding the distribution of a smaller representation of the data. This lower-dimensional representation of the data is known as **latent vector z**. There arise two questions. What is the dimension of vector z, i.e., is it too small or too large? What should be the values of this vector?

The dimension of z is a hyperparameter which the developer should choose while defining the architecture of Network. We want this to be too small to create the information bottleneck. Another reason for small z dimension is to able to sample new vectors without considering many features. Now, which values of z will produce new data points from the original data distribution? This is the beauty of VAE, which learns the distribution of z. For every element of z, we will learn mean and standard deviation.

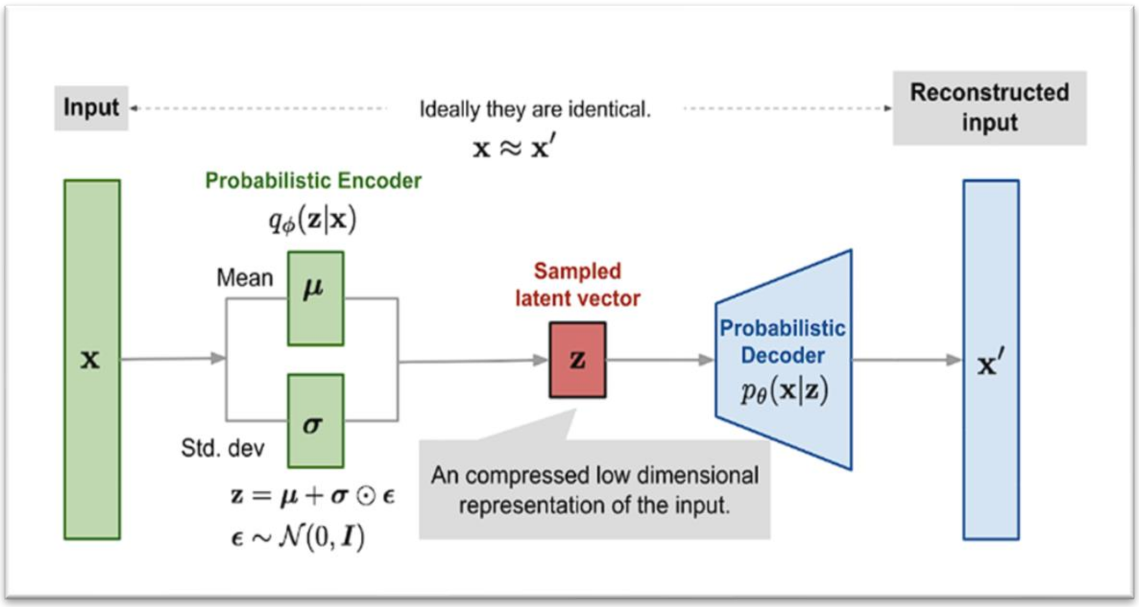

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
$\mathbf{x}$
$\boldsymbol{\sigma}$
Std. dev
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon$
$\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure 1(Architecture of VAE)

Architecture of VAE

The architecture of VAE consists of following components:

- **Probabilistic Encoder:** Given some parameters $\boldsymbol{\varphi}$ (which are parameters of the model), $\boldsymbol{q_{\varphi}(z|x)}$ models the probability of obtaining the latent vector $\boldsymbol{z}$, given the input feature vector x which connects to the μ and $\boldsymbol{\sigma}$ layers.
- **Latent vector(z):** A compressed low dimensional representation of input.
- **Probabilistic Decoder**: Given some parameters $\boldsymbol{\theta}$ (which are parameters of the model) $\boldsymbol{p_{\theta}(x|z)}$ is the probability of obtaining a data point x given the latent space z.
- The **reconstructed input** denoted as $\hat{\boldsymbol{x}}$.

Figure 1 demonstrates the architecture of a Variational Autoencoder. The input x gets fed in a Probabilistic Encoder which in turns connects with the mean and std layers. Note that usually there is a encoder Network before the mean and std layers, but here in the figure it is omitted. Then, they sample z which in turn is fed to the Probabilistic Decoder. The result is then fed to an output layer which represents the reconstructed input data.

Goal of VAE

Suppose z has k components, $z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_k \end{bmatrix}$, then the mean and standard deviation vectors are defined as $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_k \end{bmatrix}$ and $\sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_k \end{bmatrix}$, we need to learn these mean and standard deviation vectors to sample z as:

$$z = \mu + \epsilon \odot \sigma$$

where $\epsilon \sim N(0,1)$ is a gaussian with mean zero and standard deviation 1.

In other words, find a distribution $q_\varphi(z|x)$ of some latent variable z which we can sample from $z \sim q_\varphi(z|x)$ to generate new samples $x' \sim p_\theta(x|z)$.

Autoencoder vs VAE

The need to VAE comes after autoencoder faces a problem. The decoder assumes the encoded representation responds to some unknown distribution and hence, doesn't account for the encoding outside of it, i.e., it results in random pixels if data is not from source dataset as illustrated in figure 2.

This problem is solved by VAE which attempts to make sure that these encoding comes from the known distribution and reconstructs the image even data isn't from source dataset. This is done by forcing the encoder to produce a probability distribution over encoding instead of a single encoding. Decoder then sample z from the probability distribution and hence constructs the image as explained in the figure 3.

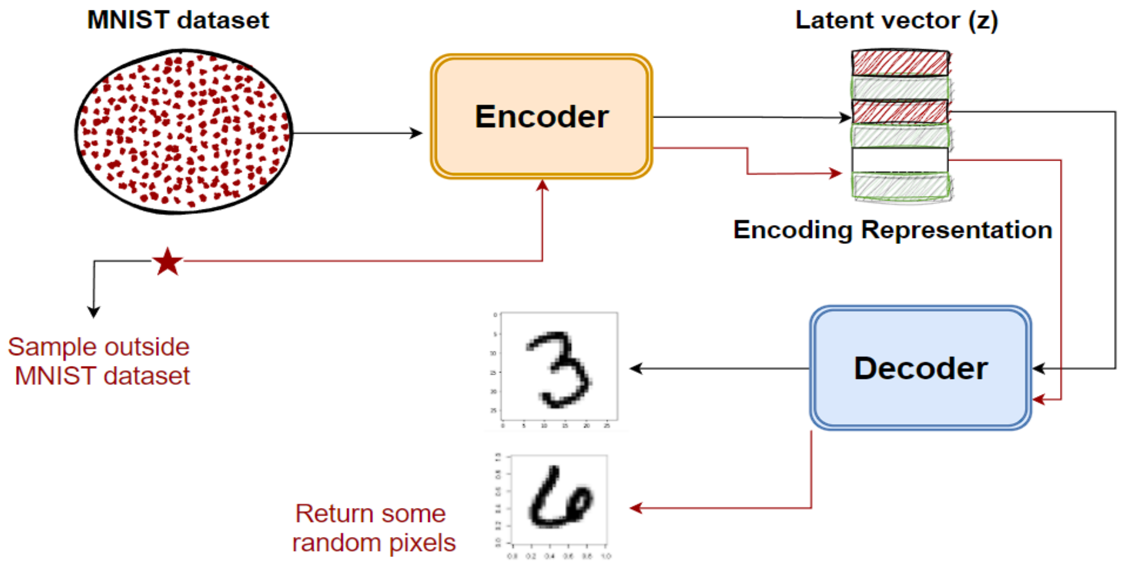


Figure 2(Problem of Autoencoder)

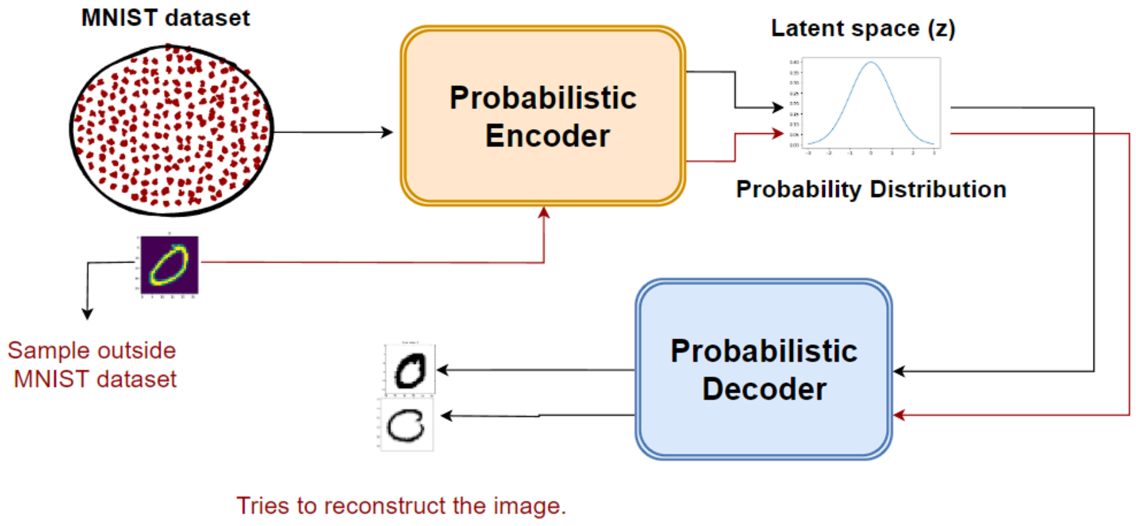


Figure 3 (Variational Autoencoder)

Latent Variables: Z is called the latent or hidden variable. In other words, it corresponds to the real feature of the object that have not been measured. For example, if we have a large datasets of face and encoder converts these images into the attributes such as smile, skintone, glasses, gender etc. These are called latent attributes which are present in z. Decoder then uses the latent attributes or variable and will try to reconstruct the image x'.

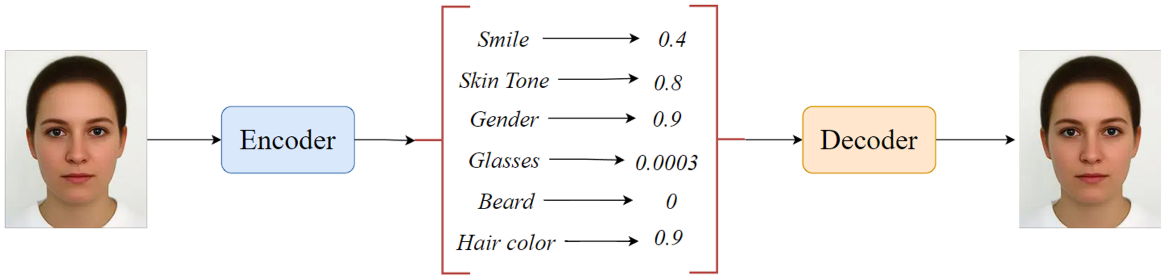
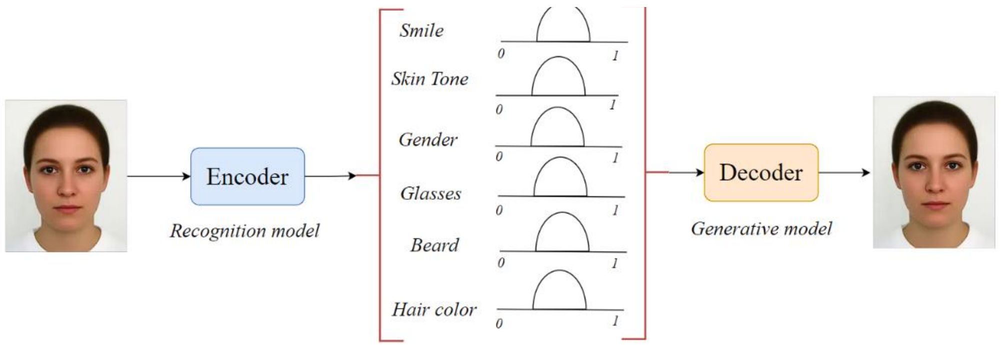


Figure 4(latent variable in Simple autoencoder)

In vaniila autoencoder which consist of an encoder and decoder architecture, the intermediate bottleneck information consists of feature that has single fixed value for an image. Consider the above example, the latent variable has six different latent attributes as illustrated in the figure 4. However, VAE defines the range of values in the form of probability distribution corresponds to each attribute within a latent variable as illustrated in the figure 5.

Figure 5(Latent variable in Variational Autoencoder)



Loss Function of VAE

Before diving into the loss function and its derivation, first understand some of the concepts as explained below:

❑ Probability concepts:

- p(X) = probability of random variable X.
- P(X|Y) = probability of random variable X provided Y has happened.

- **Bayes Theorem**, p(y|x) = p(x|y) p(y) /p(x) where, P(x|y)/p(x) is called likelihood ratio, p(y) is called prior probability and p(y|x) is called posterior probability.
- P(y|x) = $\frac{p(\boldsymbol{x}|\boldsymbol{y})p(y)}{p(x)}$ ***is also written as joint probability*** $\frac{p(x,y)}{p(x)}$
- **Theorem of total probability**: Let y_1, y_2, …., y_N be the set of mutually exclusive events (no overlap) and event X is the union of the mutually exclusive events, then, $\sum_{i=0}^{N} p(X|y_i)p(y_i)$.
- **Expectation of Random Variable:** E(x) is the weighted average of all the values that X can take where each value is being weighted according to the probability of that event. $\sum_{i=0}^{n} x_i p(X = x_i) = E_p(X)$
- **Multivariant normal distribution** when both distributions have k dimensions, is defined as:

$$N(x;\ \mu,\ \Sigma) = \frac{1}{\sqrt{(2\pi)^k\,|\Sigma|}} e^{(\frac{-1}{2}(x-\mu)^T\ \Sigma^{-1}(x-\mu))} \qquad \text{-----(A)}$$

where x is a vector of length kx1.

- **KL Divergence Concept:** Measures how one probability distribution is different from the other. For discrete probability distribution P and Q , KL-Divergence between these two distributions, denoted as D_{KL}(P || Q) , is defined as:

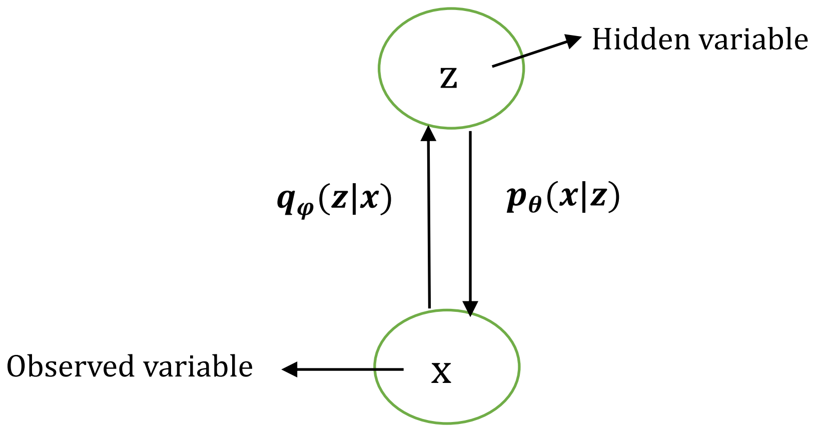
$$D_{KL}(P\,||\,Q) = \sum_x p(x) \log \frac{P(x)}{Q(x)}$$

- **Properties of KL Divergence:**
    1. $D_{KL}(P\,||\,Q) \geq 0$ **OR** $D_{KL}(Q\,||\,P) \geq 0$
    2. $D_{KL}(P\,||\,Q) \neq D_{KL}(Q\,||\,P)$ {Not symmetric}
- If P and Q are Multivariate Normal Distributions as mentioned in (A), then after putting the values of $P(x) = N(x;\ \mu_1,\ \Sigma_1\ )$ and $Q(x) = N(x;\ \mu_2,\ \Sigma_2\ )$, the simplified version is:

$$D_{KL}(P\,||\,Q) = \frac{1}{2}\left\{ log \frac{|\Sigma_1|}{|\Sigma_2|}\right\} - \mathrm{E_p}\left[(\mathrm{x} - \mu_1)^T\ \Sigma_1^{-1}(x-\mu_1)\right] + \mathrm{E_p}\left[(\mathrm{x}-\mu_2)^T\ \Sigma_2^{-1}(x - \mu_2)\right]$$

Mathematical Derivation:

As we discussed that the goal of VAE is to find the distribution $\boldsymbol{q_{\varphi}(z|x)}$ of some latent variables z, which are sampled from $\boldsymbol{z \sim q_{\varphi}(z|x)}$ and generate new samples x' from $\boldsymbol{p_{\theta}(x|z)}$. Graphical model of the goal is represented below.. The direction of the arrow or edge of the line determine the action, if it's going from z to x then the action is to generate new images using generator model ($\boldsymbol{p_{\theta}(x|z)}$), while, if it's going from x to z, then action is to learn the latent distribution of the data using recognition model ($\boldsymbol{q_{\varphi}(z|x\,)}$).



The problem of approximate inference

Let's derive the loss function. Let x be the set of observed variables and z be the set of latent variables with joint probability distribution $\boldsymbol{P(z,x)}$. Then, the inference problem is to compute the conditional probability distribution of latent variables given observations which can be written as:

$$\boldsymbol{P(z|x) = \frac{P(X|Z)\,P(z)}{P(x)}} \qquad \text{---------EQ(1)}$$

The above relation is difficult to compute because p(x) can't be solved. According to law of total probability:

$$\boldsymbol{P(x) = \int_{z} P(x|z)P(z)dz = \int_{z} P(x,z)\,dz}$$

The reason is that neither the solution is not in closed form nor the solution is tractable (i.e., it requires the exponential time to compute). It requires multiple integrals to compute correspond to each latent variable. The **alternative** is to approximate $\boldsymbol{P(z|x)}$ by another distribution $\boldsymbol{q(z|x)}$ which is defined in such a way that it has tractable solution and is in closed form. This is done using **Variational Inference (VI).**

The main idea of VI is to pose the inference problem as an optimization problem by modelling $\boldsymbol{P_\theta(z|x)}$ using $\boldsymbol{Q_\varphi(z|x)}$ where $\boldsymbol{Q_\varphi(z|x)}$ has a Gaussian distribution ($\boldsymbol{N(0,1)}$). So, the KL-divergence between $\boldsymbol{P_\theta(z|x)}$ and $\boldsymbol{Q_\varphi(z|x)}$ is:

$$\boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x))} = \sum \boldsymbol{Q_\varphi(z|x)} \left\{ \boldsymbol{log \frac{Q_\varphi(z|x)}{P_\theta(z|x)}} \right\}$$

$$= \boldsymbol{E_{z\sim Q_\varphi(z|x)}} \left\{ \boldsymbol{log \frac{Q_\varphi(z|x)}{P_\theta(z|x)}} \right\}$$

$$= \boldsymbol{E_{z\sim Q_\varphi(z|x)}} \left\{ \mathbf{log}\left(\boldsymbol{Q_\varphi(z|x)}\right) - \mathbf{log}\left(\boldsymbol{P_\theta(z|x)}\right) \right\}$$

Substituting EQ(1) and replace $z' = \boldsymbol{z\sim Q_\varphi(z|x)}$ results in :

$$\boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x))} = \boldsymbol{E_{z'}} \left\{ \mathbf{log}\left(\boldsymbol{Q_\varphi(z|x)}\right) - \mathbf{log}\left(\frac{\boldsymbol{P_\theta}(x|z)P_\theta(z)}{P_\theta(x)}\right) \right\}$$

$$= \boldsymbol{E_{z'}} \left\{ \mathbf{log}\left(\boldsymbol{Q_\varphi(z|x)}\right) - \mathbf{log}(\boldsymbol{P_\theta}(x|z)) - \log\left(P_\theta(z)\right) + \log\left(P_\theta(x)\right) \right\}$$

Since, the expectation is over z and $P_\theta(x)$ doesn't involve z, it can be moved out,

$$\boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x))} - \log\left(P_\theta(x)\right)$$
$$= \boldsymbol{E_{z'}} \left\{ \mathbf{log}\left(\boldsymbol{Q_\varphi(z|x)}\right) - \mathbf{log}(\boldsymbol{P_\theta}(x|z)) - \log\left(P_\theta(z)\right) \right\}$$

Multiplying with minus both sides and rearranging the equation, we get,

$\log\left(P_\theta(x)\right) - \boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x))}\ \boldsymbol{=} E_z'\{\ \mathbf{log}(\boldsymbol{P_\theta}(x|z))\} - E_z'\{\ \mathbf{log}\left(\boldsymbol{Q_\varphi(z|x)}\right) - \log\left(P_\theta(z)\right)\}$

$$\boxed{\log\left(P_\theta(x)\right) - \boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x))}\ \boldsymbol{=} E_{z\sim \boldsymbol{Q_\varphi(z|x)}}\{\ \mathbf{log}(\boldsymbol{P_\theta}(x|z))\} - D_{KL}((\boldsymbol{Q_\varphi(z|x)}\ ||\ \boldsymbol{P_\theta}(z))}$$

The above relation is the **objective function of VAE** in which the first term represents the reconstruction likelihood while second term ensures that our learned distribution Q is similar to the prior distribution P.

Loss function = - objective function

$$\boxed{L(\theta,\varphi) = -E_{z\sim \boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\{\mathbf{log}(\boldsymbol{P}_{\boldsymbol{\theta}}(x|z))\} + D_{KL}((\boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})\ ||\ \boldsymbol{P}_{\boldsymbol{\theta}}(z))}$$

The KL-Divergence term can also be reduced to:

$$D_{KL}((Q_{\varphi}(z|x)\ ||\ P_{\theta}(z)) = D_{KL}\big(N(\mu_{\theta}(x), \textstyle\sum_{\theta}(x)||\ N(0,1)\big)$$

$$= \frac{1}{2}\sum_{k} e^{(\ \Sigma_{\varphi}(x) + \ \mu_{\varphi}^{2}(x) - 1 - \ \Sigma_{\varphi}(x))}$$

$$\boxed{L(\theta,\varphi) = -E_{z\sim \boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\{\mathbf{log}(\boldsymbol{P}_{\boldsymbol{\theta}}(x|z))\} + \frac{1}{2}\sum e^{(\ \Sigma_{\varphi}(x) + \ \mu_{\varphi}^{2}(x) - 1 - \ \Sigma_{\varphi}(x))}}$$

More intuition about Loss Function:

As mentioned, the loss function is a function of two type of parameters, i.e., $\theta\ and\ \varphi$. These two are the network parameters of VAE that need to be learned. Loss function consist of two terms which is explained below:

1) First term $E_{z\sim \boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\{\mathbf{log}(\boldsymbol{P}_{\boldsymbol{\theta}}(x|z))\}$ consist of the expectation carried out over log when z is sampled from $\boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$. This $\boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ is a neural network known as recognition model because it learns how to map x to z. This is also called **log likelihood.** Let's discuss why? If you take $\boldsymbol{P}_{\boldsymbol{\theta}}(x|z)$ as a Gaussian distribution with mean $\mu_{\theta}(z)$ and co-variance $\sum_{\theta}(z)$, then log of $\boldsymbol{P}_{\boldsymbol{\theta}}(x|z)$ is:

$$\boldsymbol{P}_{\boldsymbol{\theta}}(x|z) = \boldsymbol{N}\left(\boldsymbol{x};\ \mu_{\theta}(z)\ ,\ \sum\nolimits_{\theta}(z)\right)$$

$$= \frac{\mathbf{1}}{\sqrt{(\mathbf{2}\boldsymbol{\pi})^{\boldsymbol{k}}\ |\Sigma_{\theta}(z)\,|}}\boldsymbol{e}^{\left(\frac{-\mathbf{1}}{\mathbf{2}}(x-\mu_{\theta}(z))^{\boldsymbol{T}}\ \Sigma_{\theta}(z)^{-\mathbf{1}}(x-\mu_{\theta}(z))\right)}$$

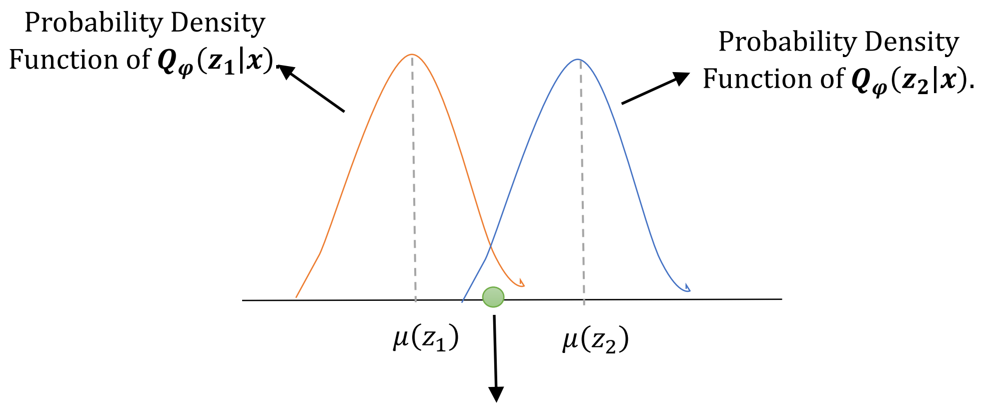
$$\log(\boldsymbol{P}_{\boldsymbol{\theta}}(x|z)) \ \propto\ (\boldsymbol{x} - \mu_{\theta}(z))^{\boldsymbol{T}} \sum\nolimits_{\theta}(z)^{-\mathbf{1}} (x - \mu_{\theta}(z))$$

The proportional term after taking log is known as **square reconstruction error** between the data x and the mean of the latent distribution $\mu_{\theta}(z)$. It is also called **data fidelity** term. This term tells how far the reconstructed data is from the mean of z. Therefore, $\boldsymbol{P}_{\boldsymbol{\theta}}(x|z)$ is also known as generator model because it takes z and maps z to x' which is the reconstructed image.

2) Second term is the KL divergence between the $\boldsymbol{Q}_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ and $\boldsymbol{P}_{\boldsymbol{\theta}}(\boldsymbol{z})$**.** Here we are assuming that we know $P_{\boldsymbol{\theta}}(z)$ which is a Gaussian of $N(0,1)$. The target is to make these distributions as close as possible, i.e., the recognition model

should be close to $N(0,1)$. In other words, it acts as a **Regularizer** because it won't allow the latent distribution to be away from $N(0,1)$. Thus, KL divergence doesn't allow the Probability density function (PDF) of the latent variable, not collapse with zero variance but penalizes if deviates from $N(0,1)$.

The role of regularizer can also be seen in the graph presented below. Consider the latent variable space of z_1 and z_2.


Probability Density Function of $\boldsymbol{Q_\varphi(z_1|x)}$
Probability Density Function of $\boldsymbol{Q_\varphi(z_2|x)}$.
$\mu(z_1)$
$\mu(z_2)$

Data sampled at this point doesn't belong to the dataset of z_1 or z_2. The generative model $\boldsymbol{P_\theta}(x|z)$ will reconstruct the data that doesn't belong to z_1 or z_2. Instead, it'll generate the data that is the weighted average of both z_1 and z_2, i.e., $x' = \frac{z_1+z_2}{2}$.

Let's discuss three different cases of loss function:

Case1: Only Data fidelity term is present in loss function.

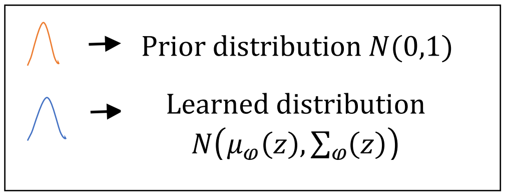
Without regularization, network cheats by learning the narrow distribution. Here the case become similar to the vanilla autoencoder where the learned encoded distribution represents just single value instead of range of values to help generator model to produce data outside from dataset as represented in the graph 1.

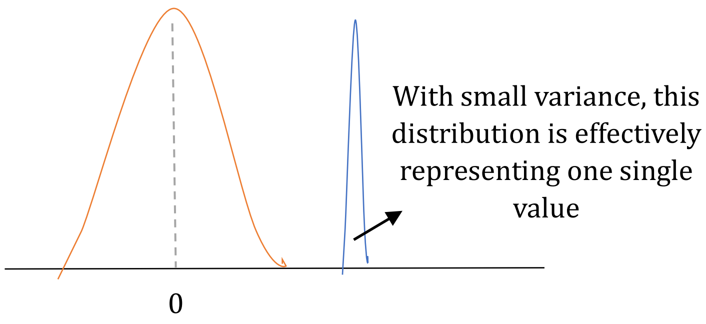
Case2: Only Regularizer term is present in loss function.

When there exists no reconstruction error, the learned distribution will deviate from the prior distribution $N(0,1)$. It actually penalizes the reconstruction loss and encourage the distribution to generate the input using some characteristics of data as represented in the graph 2.

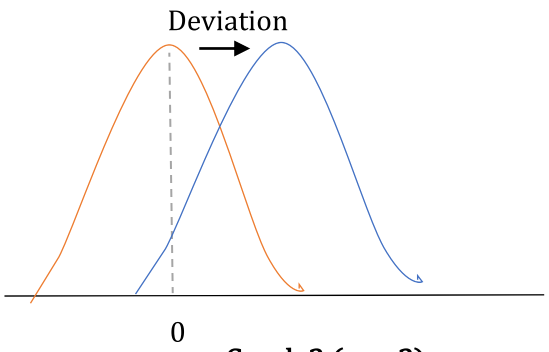
Case3: Both data fidelity and Regularizer is present in loss function.

When both terms are present, there exists the attraction between the learned and the prior distribution. The reason of this attraction is basically due to data fidelity term. The data fidelity term will force the distribution $Q_{\varphi}(z|x)$ to move towards $N(0,1)$ as represented in the graph 3.

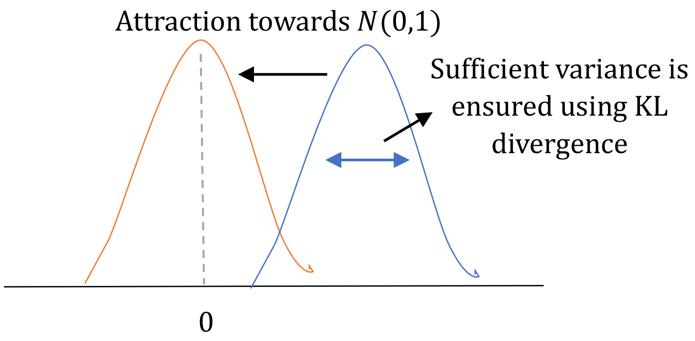

Prior distribution $N(0,1)$
Learned distribution $N\left(\mu_{\varphi}(z), \Sigma_{\varphi}(z)\right)$


With small variance, this distribution is effectively representing one single value
0

Graph 1 (case1)



Graph 2 (case2)


Attraction towards $N(0,1)$
Sufficient variance is ensured using KL divergence
0

Graph 3 (case3)

Optimization:

Optimization of loss function involves the finding of the optimal values of θ and φ. The loss function is also called as **Variational Lower Bound (ELBO).** The lower bound comes from the fact that the KL-Divergence is always non-negative and therefore, loss function is the lower bound of $\log\big(P_\theta(x)\big)$.

The ELBO expression is:

$\log\big(P_\theta(x)\big) - D_{KL}\big(Q_\varphi(z|x)\ ||\ P_\theta(z|x)\big) = -L(\theta,\varphi)$According to the properties of the KL-Divergence,

$$\boldsymbol{D_{KL}(Q_\varphi(z|x)\ ||\ P_\theta(z|x) \geq 0}$$

This implies that:

$$L(\theta,\varphi) \leq \log\big(P_\theta(x)\big)$$

Minimizing loss function is actually maximizing the lower bound of the probability of generating real data samples. The simplified loss function is :

$$\min_{\theta,\varphi}(L(\theta,\varphi)) = \min_{\theta,\varphi}(= -E_{z\sim Q_\varphi(Z|x)}\{\log\big(P_\theta(x|z)\big)\} + \tfrac{1}{2}\textstyle\sum_k e^{(\Sigma_\varphi(x) + \mu_\varphi^2(x) - 1 - \Sigma_\varphi(x))})$$

$$\begin{tikzcd}
\boxed{\theta^* = \nabla_\theta\big(L(\theta,\varphi)\big)\ ;\ \varphi = constant} \arrow[d, shift right=3, "\theta^*"'] \arrow[d, shift left=3, "\theta^{**}"] \\
\boxed{\varphi^* = \nabla_\theta\big(L(\theta^*,\varphi)\big)} \arrow[u, bend left=60, "\varphi = \varphi^*"]
\end{tikzcd}$$

The optimization of loss function actually involves the optimization principle where you find the optimization of θ while making φ constant and produce θ^*. Use this θ^* to another optimization technique which produces φ^*. This φ^* is again used to produce θ^{**} which is further use to produce φ^{**}. This process continues for a fixed number of iterations as represented in the graph above.

However, while taking the derivative of the loss function w.r.t φ. Let's discuss it in detail.

Algorithm for calculating the optimal values of $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}$:

Optimization of loss function is carried out w.r.t θ and φ to learn the optimal values of both for fixed number of iterations. The loss function is:

$$L(\theta,\varphi) = -E_{z\sim \boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x})}\{\mathbf{log}(\boldsymbol{P_\theta}(x|z))\} + \frac{1}{2}\sum_k e^{(\Sigma_\varphi(x)+\mu_\varphi^2(x)-1-\Sigma_\varphi(x))}$$

It is done during back propagation to find the derivative of loss function w.r.t θ and φ at the same time. Thus, performs following steps for i=1,2,.....,n:

1) Calculate the derivative of the loss function w.r.t θ while making φ constant.

$$\theta_i^{\wedge} = \nabla_\theta(L(\theta,\varphi))$$

The second term doesn't contain any θ term and derivative of constant is zero. Hence, only first term contributes in derivative calculation. According to Monte Carlo estimation of expectation, the derivative becomes:

$$\boldsymbol{\theta_i^{\wedge}} = \frac{\mathbf{1}}{\boldsymbol{L}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{l}} \boldsymbol{\nabla_\theta}\left(\mathbf{log}\left(\boldsymbol{P_\theta}(\boldsymbol{x}|\boldsymbol{z}^{(l)})\right)\right)$$

Where $z^{(l)}$ are the samples coming from the learned distribution, i.e., $\boldsymbol{z}^{(l)} \sim \boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x})$. The above relation will result in first optimal value of θ.

2) Calculate the derivative of loss function w.r.t φ using the above calculated $\theta_i^{\wedge}$.

$$\varphi_i^{\wedge} = \nabla_\varphi\left(L(\theta_i^{\wedge},\varphi)\right)$$

$$\varphi_i^{\wedge} = \nabla_\varphi[-E_{z\sim \boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x})}\{\mathbf{log}(\boldsymbol{P_\theta}(x|z))\} + \tfrac{1}{2}\textstyle\sum_k e^{(\Sigma_\varphi(x)+\mu_\varphi^2(x)-1-\Sigma_\varphi(x))}] \text{ -----(B)}$$

Examining the above relation, the derivative of second term is calculated easily because it involves φ directly. However, the first term causes problem while taking derivative. In fact, the derivative of first term is harder to estimate because φ appears in the distribution w.r.t which expectation is taken. So, we can't really move the derivative inside the expectation as:

$$\nabla_\varphi E_{Q_\varphi(Z|x)}[f(z)] \neq E_{Q_\varphi(Z|x)}[\nabla_\varphi f(z)]$$

Solution to the issue while back propagating $\boldsymbol{\varphi}$:

If the above expression is written in such a way that φ appears just inside the expectation, then the gradient is computed. The above problem is resolved by pushing the gradient inside the expectation which is possible only when φ occurs inside expectation. This is done by performing linear transformation dependent on ε and x where $\varepsilon \sim N(0,1)$. Thus, the expectation now looks like:

$$E_{Q_{\varphi}(z|x)}\,[f(z)] = E_{p(\varepsilon)}\,\left[f\left(g_{\varphi}(\varepsilon, x)\right)\right]$$

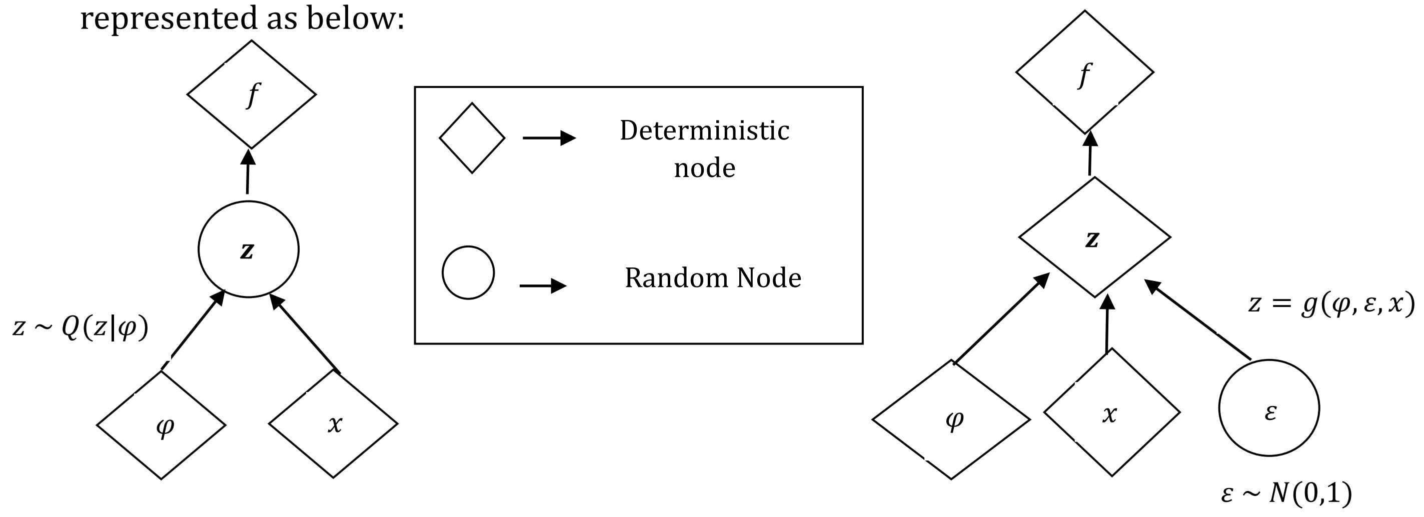
Now, the sophisticated variable z is pushed out by introducing another variable that samples from normal distribution ($N(0,1)$) using the following transformation:

$$z = g_{\varphi}(\varepsilon, x) = \mu_{\varphi}(x) + \varepsilon \bigodot \sum_{\varphi}^{\frac{1}{2}}(x)$$

The above transformation means that $N\left(\mu_{\varphi}(x), \sum_{\varphi}(x)\right)$ can be obtained from $N(0,1)$ when z is samples from ε which is a normal distribution. This is called **Reparameterization trick.**

Reparameterization Trick:

The reparameterization trick is a mathematical operation used in the training process of VAEs. It's a technique that helps bypass a significant problem in training VAEs: the backpropagation algorithm cannot be applied directly through random nodes. The VAE architecture includes a sampling operation where we sample latent variables from a distribution parameterized by the outputs of the encoder. The direct application of backpropagation here is problematic because of the inherent randomness of the sampling operation. In other words, since computing the latent involve sampling from a (multivariate normal) distribution. This sampling operation introduce stochasticity and therefore cannot be differentiated. Graphically it is represented as below:



The left graph is the original distribution in which samples are drawn from the probability distribution of $Q_{\varphi}(z|x)$. The z act as a stochastic node since backpropagation along the path is not possible due to the presence of φ in the distribution over which expectation is happened.

However, the right-hand side graph represents the reparameterization trick where we push the stochastic node outside. That means we are now sampling from $N(0,1)$ and apply linear transformation to make it behave like the same one as original. Now, we can easily back propagate through the network.

Back to optimization, now consider EQ(B) which is now converted to:

$$\hat{\varphi_i} = \nabla_{\varphi}[- E_{z^{(l)} \sim p(\varepsilon)}\{ \log\left(P_{\theta}\left(x|z^{(l)}\right)\right)\} + \frac{1}{2}\sum_k e^{(\Sigma_{\varphi}(x) + \mu_{\varphi}^2(x) - 1 - \Sigma_{\varphi}(x))}]$$

Now, derivative can be computed as:

$$\hat{\varphi_i} = \left[- E_{z^{(l)} \sim p(\varepsilon)}\{ \nabla_{\varphi}(\log\left(P_{\theta}\left(x|z^{(l)}\right)\right))\} + \nabla_{\varphi}\left(\frac{1}{2}\sum_k e^{(\Sigma_{\varphi}(x) + \mu_{\varphi}^2(x) - 1 - \Sigma_{\varphi}(x))}\right)\right]$$

According to Monte-Carlo estimation:

$$\hat{\varphi_i} = -\frac{1}{S}\sum_{S=1}^{S}\left(\log\left(P_{\theta}\left(x|z^{(l)}\right)\right)\right) + \nabla_{\varphi}\left(\frac{1}{2}\sum_k e^{(\Sigma_{\varphi}(x) + \mu_{\varphi}^2(x) - 1 - \Sigma_{\varphi}(x))}\right)]$$

Where $z^{(l)}$ comes after applying the transformation i.e., $z^{(l)} = \mu_{\varphi}(x) + \varepsilon \odot \sum_{\varphi}(x)$ where $\varepsilon \sim N(0,1)$.

Conditional VAE

Recall, the objective function of VAE is:

$$\log(P_{\theta}(x)) - \boldsymbol{D_{KL}}(\boldsymbol{Q_{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) \;||\; \boldsymbol{P_{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = E_{z\sim \boldsymbol{Q_{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\{ \mathbf{log}(\boldsymbol{P_{\theta}}(x|z))\} - D_{KL}((\boldsymbol{Q_{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) \;||\; \boldsymbol{P_{\theta}}(z))$$

Here, we want to optimize the log likelihood of our data $P(X)$ under some "encoding" error. The original VAE model has two parts: the encoder $Q(z|X)$ and the decoder $P(x|z)$. Looking closely, we could see why can't VAE generate specific data. For example, suppose the user inputted character '2', how do we generate handwriting image that is a character '2'? It's because the encoder models the latent variable z directly based on X, it doesn't care about the different type of X. For example, it doesn't take any account on the label of X. Similarly, in the decoder part, it only models X directly based on the latent variable z.

We could improve VAE by **conditioning the encoder and decoder** to another thing. Let's say that other thing is c, so the encoder is now conditioned to two variables X and c and is represented as $Q(z|X,c)$ The same with the decoder, it's now conditioned to two variables z and c and is represented as $P(X|z,c)$**.**

The objective function is now changed to:

$$\log\big(P_\theta(x|c)\big) - \boldsymbol{D_{KL}}\big(\boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{c}) \ || \ \boldsymbol{P_\theta}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{c})\big) = E_{z\sim \boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{c})}\{\,\mathbf{log}(\boldsymbol{P_\theta}(x|z,c))\} - D_{KL}\big((\boldsymbol{Q_\varphi}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{c}) \ || \ \boldsymbol{P_\theta}(z|c)\big)$$

Where c refers to the labels of the input given to both encoder and decoder. The latent distribution $P(z)$ is now become a conditional probability distribution $P(z|c)$. The image represents the conditional VAE:

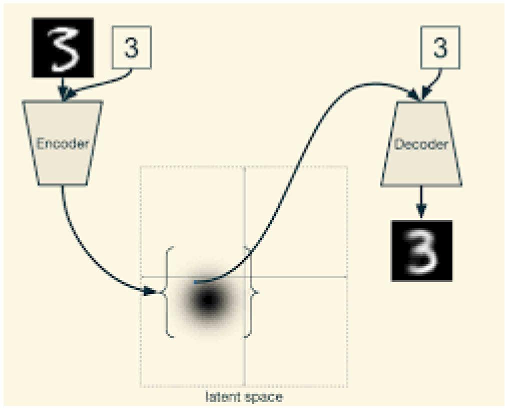


Figure (C_VAE on MNIST dataset)