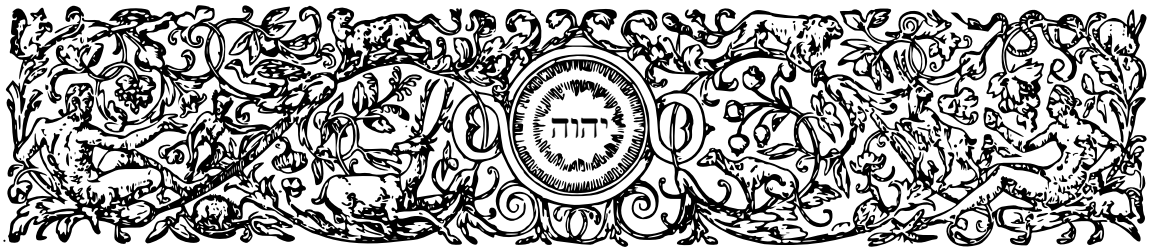


# Generative Model Learning Logs

**Author: CuiEM**

**Education: UCAS**

**Date: May 6, 2025**

*“Ywaq zuq ywag shn’ma Bo’al za qwor.”*

# Contents 0

# Introduction 1

## 1.1 First Thing to Say

This learning log of Generative Model is a digital version of my learning note during my postgraduate study. It mostly came from a seriese of video in Youtub (Stanford CS236 and Aladdin Persson) and some papers which will be discussed in detail in my website.

## 1.2 Let's Begin with Chain Rule!

**Definition 1.1 - Generative Model[1].**

A generative model is a statistical model of the joint probability distribution $P(X, Y)$ on a given variable $X$ and target variable $Y$(Ng and Jordan, 2001); A generative model can be used to "generate" random instances (outcomes) of an observation $x$.

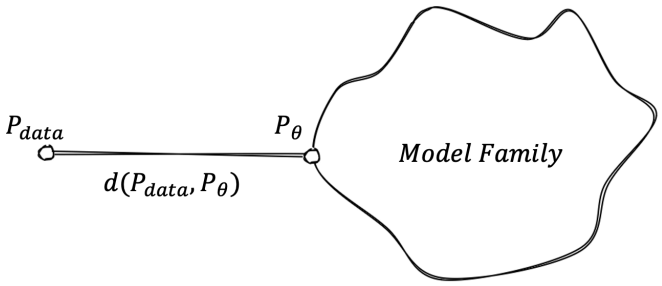


Figure 1: Generative Model

In the above figure1, we can see that the whole process is about getting the model $P_\theta$ close to the true data distribution $P_{data}$. Unfortunately, we don't know both the true data distribution $P_{data}$ and the model $P_\theta$.

There is a simpliest method to get the model $P_\theta$ which is **Chain Rule**, we can generate samples if given first $P(S_1)$, which is the basci idea behind the Autoregressive model in Section2.

**Method 1.1 - Chain Rule.**

$$P(S_1 \cap S_2 \cap \cdots \cap S_n) = P(S_1)P(S_2|S_1)\cdots P(S_n|S_1 \cap S_2 \cap \cdots \cap S_{n-1}) \tag{1.1}$$

[1] Definition Comes from Wikipedia

Still, this simple model have huge amounts of parameters. So let's make a stronger assumption:

**Assumption 1.1 - Assumption of Independence.**

$$(X_{i+1} \perp X_1, \dots, X_{i-1})|X_i \tag{1.2}$$

So we can get a simple model with $2n-1$ parameters:

$$P(X_1, \dots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_2)\cdots P(X_n|X_{n-1}) \tag{1.3}$$

## 1.3 Bayesian Network

However, above assumption is too strong, we need to make it weaker by building a **Bayesian Network** with a tool called **Conditional Independence**.

**Method 1.2 - Bayesian Rule.**

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{1.4}$$

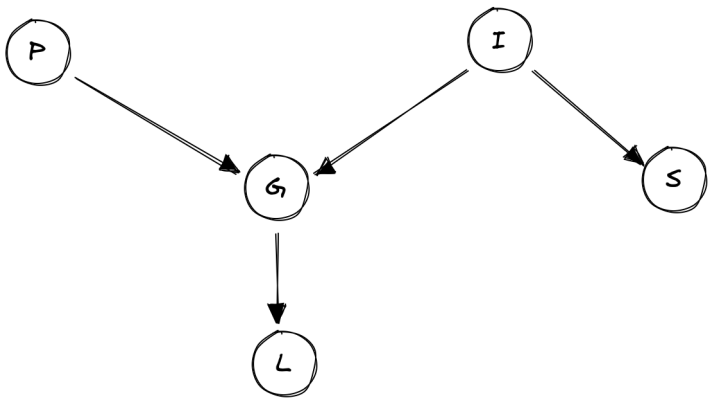


Figure 2: Bayesian Network Example

For example, we are given a Bayesian Network(which is must be a DAG) in the above Figure2, we can get below joint distribution:

$$P(d, i, g, s, l) = P(d)P(i)P(g|i, d)P(s|i)P(l|g) \tag{1.5}$$

which coulde be a verly long expression in Chain Rule:

$$P(d, i, g, s, l) = P(d)P(i|d)P(g|i, d)P(s|i, d, g)P(l|i, d, g, s) \tag{1.6}$$

## 1.4 Discriminative VS. Generative

**Example 1.1.**

The example is shown in below picture. $X_i$ are conditionally independent given $Y$:
So the joint distribution is:

$$p(y, x_1, \dots x_n) = p(y) \prod_{i=1}^{n} p(x_i \mid y) \tag{1.7}$$

Predict with Bays Rule in Equation1.4:

$$p(Y = 1 \mid x_1, \dots x_n) = \frac{p(Y = 1) \prod_{i=1}^{n} p(x_i \mid Y = 1)}{\sum_{y=\{0,1\}} p(Y = y) \prod_{i=1}^{n} p(x_i \mid Y = y)} \tag{1.8}$$

Using Chain Rule in Equation1.1, we get

$$p(Y, \mathbf{X}) = p(\mathbf{X} \mid Y)p(Y) = p(Y \mid \mathbf{X})p(\mathbf{X}) \tag{1.9}$$

Corresponding Bayesian Networks is shown in below picture. In a easiest way to say: Discriminative Model is trying to predict the class of a sample given the sample itself; whereas Generative Model is trying to predict the sample given the class.

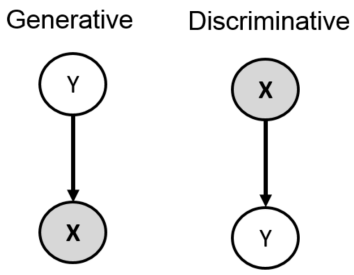


Figure 3: Discriminative VS. Generative

## 1.5 Continuous Variables

If X is a continuous random variable, we can usually represent it using its probability density function $p_X : \mathbb{R} \to \mathbb{R}^+$. However, we cannot represent this function as a table anymore. But Chain rule, Bayes rule, etc all still apply.

**Example 1.2 - Variational autoencoder (VAE).**

Bayes net Z $\longrightarrow$ X and:

- $Z \sim \mathcal{N}(0, 1)$
- $X \mid (Z = z) \sim \mathcal{N}(\mu_\theta(z), e^{\sigma_\phi(z)})$ where $\mu_\theta(z)$ and $\sigma_\phi(z)$ are neural networks with parameters (weights) $\theta$, $\phi$ respectively.

# Autoregressive Model 2

## 2.1 Fully Visible Sigmoid Belief Network (FVSBN)

**Example 2.1 - Binarized MNIST.**

A dataset of handwritten digits (binarized MNIST):

Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1). So we need to learn a probability distribution $p(x) = p(x_1, \dots, x_{784})$ over $x \in 0, 1^{784}$, $x$ looks like a digit.

---

First of all, according to Chain Rule we know that

$$p(x) = p(x_1, \dots, x_{784}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_{784}|x_1, \dots, x_{783}) \tag{2.1}$$

Now we assuem that:

**Definition 2.1 - Modeling Structure for FVSB(Frey et al., 1995).**

$$p(x_1, \dots, x_{784}) = p_{PCT}(x_1; \alpha^1)p_{logit}(x_2 \mid x_1; \alpha^2) \cdots p_{logit}(x_{784} \mid x_1, \dots, x_{783}; \alpha^{784}) \tag{2.2}$$

- $p_{PCT}(X_1 = 1; \alpha^1) = \alpha^1$
- $p_{logit}(X_2 = 1 \mid x_1; \boxtimes^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$
- $p_{logit}(X_3 = 1 \mid x_1 x_2; \boxtimes^2) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$

---

We are using parameterized functions (e.g., logistic regression above) to predict next pixel given all the previous ones. Called **Autoregressive model**.

In the Binarized MNIST, the conditional variables $X_i \mid X_1, \dots, X_{i-1}$ are Bernoulli with parameters. Thus, we let:

$$\hat{x}_i = p(X_i = 1 \mid X_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j) \tag{2.3}$$

Now with the help of probability, we know how to sample from model:

1. Sample $\bar{x}_1$ from $p(x_1)$
2. Sample $\bar{x}_2$ from $p(x_2 \mid x_1 = \bar{x}_1)$
3. Sample $\bar{x}_3$ from $p(x_3 \mid x_1 = \bar{x}_1, x_2 = \bar{x}_2)$
4. ...

## 2.2 Neural Autoregressive Density Estimation (NADE)

Still, FVSBN couldn't give us a satisfying result. So, to improve the model, we need to use neural networks instead of logistic regression.

**Definition 2.2 - Neural Networks instead of Logistic Regression.**

$$\begin{aligned} \mathbf{h}_i &= \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i) \\ \hat{x}_i &= p(x_i | x_1, \dots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \alpha_i, b_i}_{\text{parameters}}) = \sigma(\alpha_i \mathbf{h}_i + b_i) \end{aligned} \tag{2.4}$$

From above equation, we can get the original structure of NADE(Larochelle and Murray, 2011). Thus, we can visualize it as follows:

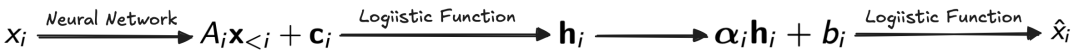


Figure 4: Structure of NADE

However, there are too much parameters in the original model. We let matrix $A$ share parameters across all the layers.

**Definition 2.3 - Modeling Structure of NADE.**

$$\begin{aligned} \mathbf{h}_i &= \sigma(W_{\cdot,<i} \mathbf{x}_{<i} + \mathbf{c}) \\ \hat{x}_i &= p(x_i | x_1, \dots, x_{i-1}) = \sigma(\alpha_i \mathbf{h}_i + b_i) \end{aligned} \tag{2.5}$$

The parameters sharing can be visualized as follows which is an example:

$$\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{W_{\cdot,<2}} x_1 + \mathbf{c}\right) \quad \mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{W_{\cdot,<3}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) \quad \mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{W_{\cdot,<4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

WLOG, we can extend model not only to non-binary discrete random variable $X_i \in \{1, \dots, K\}$, but also to continuous random variables $X_i \in \mathbb{R}$ (RNADE(Uria et al., 2014)).

## 2.3 What if Neural Network?

On the surface, FVSBN and NADE look similar to an Autoencoder:

- Encoder: $e(x) = \sigma(W^2(W^1x + b^1) + b^2)$
- Decoder: $d(e(x)) \approx x$

So how to get a generative model from an autoencoder? we can use a **single neural network** (with n inputs and outputs) to produce all the parameters $\hat{x}$ in a single pass.

- MADE(Germain et al., 2015)
- Character RNN
- Pixel RNN(Van Den Oord, Kalchbrenner, and Kavukcuoglu, 2016)
- Pixel CNN(Van den Oord et al., 2016)
- PixelDefend(Song et al., 2017)
- WaveNet(Van Den Oord, Dieleman, et al., 2016)

# 3 Variational Autoencoder

## 3.1 Latent Variable Model

Before in the Section Autoregressive Model we can see that probability distribution is only inferenced by the data themself. But in reality, lots of variability in images x due to gender, eye color, hair color, pose, etc. However, unless images are annotated, these factors of variation are not explicitly available (latent). So the key idea is to **explicitly model these factors using latent variables z**.


Figure 5: Simple Latent Variable Model

Latent variables z correspond to high level features. However, it is very difficult to specify these conditionals by hand. That's why we need **Latent Variable Model** which would finish this task.

Morever, we could choose to use neural networks to model the conditionals (deep latent variable models), or mixture simple distribution to build complex distribution (e.g. Mixture of Gaussians).

---

**Definition 3.1 - Mixture of Gaussians. Bayes net: $z \rightarrow x$.**

---

The whole modeling process actually looks like a kind of unsupervised learning: Clustering. $K$ means there are $K$ clusters and each $k \in K$: $\mu_k, \Sigma_k$ corresponds to one probability distribution $p(\mathbf{x} \mid \mathbf{z} = k)$:

- $z \sim \text{Categorical}(1, \cdots, K)$
- $p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$

---

Thus, the probability distribution of x is equal to:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) = \sum_{k=1}^{K} p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}} \tag{3.1}$$

Thinking out of box, we can extend above model into infinite number of Gaussians:

**Definition 3.2 - A mixture of an infinite number of Gaussians.**

Even though $p(x \mid z)$ is simple, the marginal $p(x)$ is very complex/flexible

- $z \sim (N(0, I))$
- $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where $\mu_\theta, \Sigma_\theta$ are **neural networks**
  - $\mu_\theta(\mathbf{z}) = \sigma(A\mathbf{z} + c) = (\sigma(a_1\mathbf{z} + c_1), \sigma(a_2\mathbf{z} + c_2)) = (\mu_1(\mathbf{z}), \mu_2(\mathbf{z}))$
  - $\Sigma_\theta(\mathbf{z}) = diag(\exp(\sigma(B\mathbf{z} + d))) = \begin{pmatrix} \exp(\sigma(b_1\mathbf{z} + d_1)) & 0 \\ 0 & \exp(\sigma(b_2\mathbf{z} + d_2)) \end{pmatrix}$
  - $\theta = (A, B, c, d)$

Bingo! we get the structure of VAE(Kingma, 2013) very quickly. Yes, it is so simple that you couldn't believe it. Actually the hard part is training beacuse even though $p(\mathbf{x} \mid \mathbf{z})$ is simple, the marginal $p(x)$ is very complex or expensive.

## 3.2 The Evidence Lower bound

Follw the last subsection's architecture, we now know the ultimate goal is to make sure the generative model give the right sample in highest probability with parameter $\theta$:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)}) = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) \tag{3.2}$$

Let's make life easier by considering discrete condition here. (you can check continuous condition here.) The Log-Likelihood function can be transformed like this:

$$\log\left(\sum_{\mathbf{z}\in\mathcal{Z}} p_\theta(\mathbf{x}, \mathbf{z})\right) = \log\left(\sum_{\mathbf{z}\in\mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z})\right) = \log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}\right]\right) \tag{3.3}$$

Above eqaution still is hard to compute. Thus, instead of getting the direct result of it, we can get the Evidence Lower Bound(ELBO) by Jensen Inequality:

$$\begin{aligned} \log p(\mathbf{x}; \theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log\left(\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}\right) \\ &= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x}, \mathbf{z}) - \underbrace{\sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z})}_{\text{Entropy } H(q) \text{ of } q} \\ &= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x}, \mathbf{z}) + H(q) \end{aligned} \tag{3.4}$$

The equality holds if $q(\mathbf{z}) = p(\mathbf{z} \mid \mathbf{x}; \theta)$. Also, there is another perspective we can examine the ELBO. In other words, we can derive the ELBO from a different and elegant way.

By the non-negative property of KL divergence, we know:

$$D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x};\theta)) = -\sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + \log p(\mathbf{x};\theta) - H(q) \geq 0 \tag{3.5}$$

Rearranging, we re-derived the Evidence lower bound (ELBO) i.e. Equation (3.4):

$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q) \tag{3.6}$$

In general,

$$\log p(\mathbf{x};\theta) = \text{ELBO} + D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x};\theta)) \tag{3.7}$$

The closer $q(\mathbf{z})$ is to $p(\mathbf{z} \mid \mathbf{x};\theta)$, the closer the ELBO is to the true log-likelihood. Thus, avoiding compute the complex and intractable log-likelihood of $p_\theta(x)$, we maxmize its ELBO i.e. the lower bound.

## 3.3 Variational Inference

- Question: What if the posterior $p(\mathbf{z} \mid \mathbf{x};\theta)$ is intractable?
- Answer: Use Variational Inference to approximate it.

---

**Definition 3.3 - Variational Inference.**

We assume distribution $q(\mathbf{z};\phi)$ is a tractable distribution over the hidden variables parameterized by $\phi$ (**variational parameters**). For example, $q(\mathbf{z};\phi)$ is a Gaussian distribution with mean and covariance specified by $\phi$:

$$q(\mathbf{z};\phi) = \mathcal{N}(\mu_\phi, \Sigma_\phi)$$

So we can pick $\phi$ to make $q(\mathbf{z};\phi)$ as close as possible to the intractable $p(\mathbf{z} \mid \mathbf{x};\theta)$.

---

Now with two parameters $\theta$ and $\phi$, we can re-write the ELBO as:

$$\begin{aligned} \log p(\mathbf{x};\theta) &\geq \textstyle\sum_{\mathbf{z}} q(\mathbf{z};\phi) \log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}} \\ &= \mathcal{L}(\mathbf{x};\theta,\phi) + D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x};\theta)) \end{aligned} \tag{3.8}$$

Here is a vivid figure to show the relationship between marginal likelihood and ELBO:

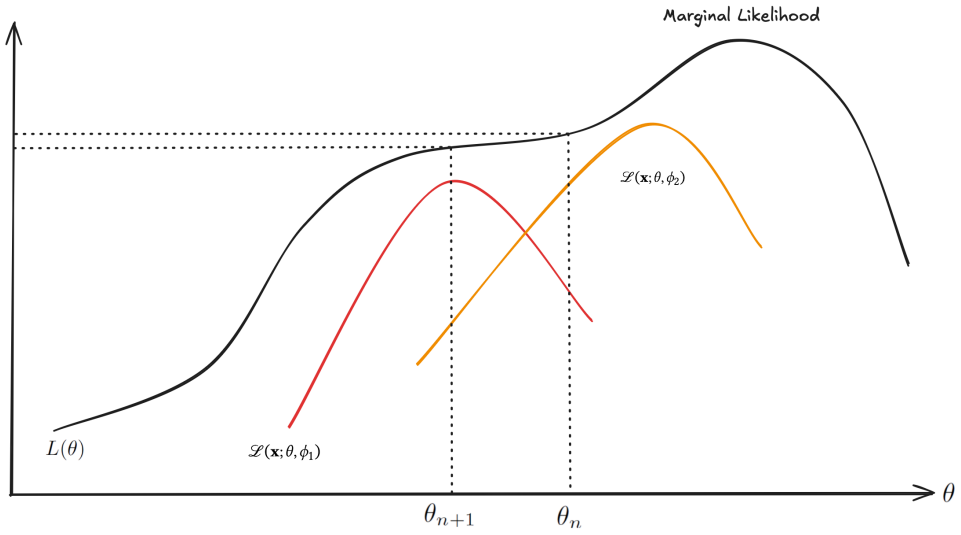


Figure 6: Relationship between marginal likelihood and ELBO

## 3.4 Optimization

We use different variational parameters $\phi^i$ for every data point $\mathbf{x}^i$ if we want to extend the maximum likelihood learning to the enitire dataset. So the maximum likelihood learning objective becomes below forms:

$$\max_{\theta} l(\theta;\mathscr{D}) = \max_{\theta} \sum_{\mathbf{x}^i\in\mathscr{D}} \log p_\theta(\mathbf{x}^i;\theta) \geq \max_{\theta,\phi^1,\cdots,\phi^M} \sum_{\mathbf{x}^i\in\mathscr{D}} \mathscr{L}(\mathbf{x}^i;\theta,\phi^i) \tag{3.9}$$

After getting the optimization objective which is the right side of above inequality, we can use (stochastic) gradient descent to optimize it. The algorithm's pseudo-code is shown below:

**Algorithm 1:** Variational Inference Optimization Algorithm

Initialize $\theta, \phi^1, \ldots, \phi^M$;
Randomly sample a data point $\mathbf{x}^i$ from dataset $\mathscr{D}$;
**while** $\phi^{i,*}$ *is not approximatly converged to* $\arg\max_\phi \mathscr{L}(\mathbf{x}^i;\theta,\phi)$ **do**
    $\phi^i \leftarrow \phi^i + \eta\nabla_{\phi^i}\mathscr{L}(\mathbf{x}^i;\theta,\phi^i)$;
Compute $\nabla_\theta\mathscr{L}(\mathbf{x}^i;\theta,\phi^{i,*})$;
Update $\theta$ in the gradient direction.

The optimization process is easy unless the gradients is hard to compute, so we use Monte Carlo sampling to lower down the difficulty. Also according to the Variational Inference, we substitute the intractable posterior $q(\mathbf{z})$ with the variational distribution $q(\mathbf{z};\phi)$ in the ELBO:

$$\begin{aligned}
\mathscr{L}(\mathbf{x};\theta,\phi) &= \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) \\
&= \mathrm{E}_{q(\mathbf{z};\phi)}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)] \\
&\approx \frac{1}{K}\sum_k \left[\log p(\mathbf{z}^k,\mathbf{x};\theta) - \log q(\mathbf{z}^k;\phi)\right]
\end{aligned} \tag{3.10}$$

For now, the gradient with respect to $\theta$ is easy to compute if we sample $\mathbf{z}^1, \cdots, \mathbf{z}^K$ from $q(\mathbf{z}; \phi)$.

$$\begin{aligned} \nabla_\theta \mathsf{E}_{q(\mathbf{z};\phi)}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q(\mathbf{z}; \phi)] = & \quad \mathsf{E}_{q(\mathbf{z};\phi)}[\nabla_\theta \log p(\mathbf{z}, \mathbf{x}; \theta)] \\ \approx & \quad \frac{1}{K} \sum_k \nabla_\theta \log p(\mathbf{z}^k, \mathbf{x}; \theta) \end{aligned} \tag{3.11}$$

However, the gradient with respect to $\phi$ is not easy to compute because the expection depends on $\phi$. There is a general technique called REINFORCE to compute the gradient with respect to $\phi$. But in here, we introduce a more efficient but only applicable to continuous $\mathbf{z}$ called **Reparameterization Trick**.

---

**Example 3.1 - Reparameterization Trick.**

---

**Objective**: we want to compute a gradient with respect to $\phi$ of

$$E_{q(\mathbf{z};\phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z} \tag{3.12}$$

**Assumptipon**: $q(\mathbf{z}; \phi) = \mathcal{N}(\mu, \sigma^2 I)$ is Gaussian with parameters $\phi = (\mu, \sigma)$.

Noticing that sample from Gaussian distribution $q(\mathbf{z}; \phi)$ is also equivalent to sample $\epsilon$ from $\mathcal{N}(0, I)$ and then take a linear transformation: $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon; \phi)$

By this equivalence we can compute the expection and gradient:

$$E_{\mathbf{z}\sim q(\mathbf{z};\phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z} = E_{\epsilon\sim\mathcal{N}(0,I)}[r(g(\epsilon; \phi))] = \int \mathcal{N}(\epsilon) r(\mu + \sigma\epsilon) d\epsilon \tag{3.13}$$

$$\begin{aligned} \nabla_\phi E_{q(\mathbf{z},\phi)}[r(\mathbf{z})] &= \nabla_\phi E_\epsilon[r(g(\epsilon; \phi))] \\ &= E_\epsilon[\nabla_\phi r(g(\epsilon; \phi))] \\ &\approx \quad \frac{1}{K} \sum_k \nabla_\theta r(g(\epsilon^k \phi)) \end{aligned} \tag{3.14}$$

where $\epsilon^1, \cdots, \epsilon^k \sim \mathcal{N}(0, I)$

---

Back to our case, we can easily find out that the ELBO have the same form except that ours is slightly more complicated because the term inside the expectation also depends on $\phi$, thus $r(\mathbf{z})$ becomes $r(\mathbf{z}, \phi)$.

$$\begin{aligned} \mathscr{L}(\mathbf{x}; \theta, \phi) = & \quad \sum_{\mathbf{z}} q(\mathbf{z}; \phi) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q(\mathbf{z}; \phi)) \\ = & \quad E_{q(\mathbf{z};\phi)}[\underbrace{\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q(\mathbf{z}; \phi)}_{r(\mathbf{z},\phi)}] \end{aligned} \tag{3.15}$$

## 3.5 Autoencoder Perspective

Now forget about variational inference for a moment, let's see what the VAE looks like from the perspective of autoencoder. Below figure is the illustration of variational autoencoder model with the multivariate Gaussian assumption[2].

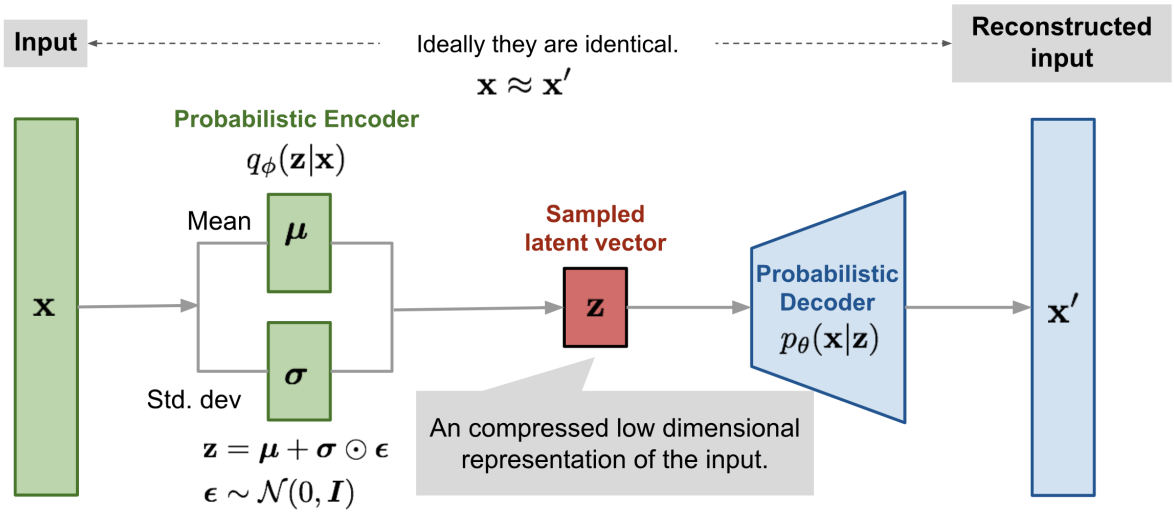


Figure 7: VAE from the perspective of autoencoder

The whole process can be treated as an autoencoder with two parts:

1. Take a data point $\mathbf{x}^i$, map it to $\hat{\mathbf{z}}$ by sampling from $q_\phi(\mathbf{z}|\mathbf{x}^i)$ (encoder). Sample from a Gaussian with parameters $(\mu, \sigma) = \text{encoder}_\phi(\mathbf{x}^i)$ (The Green part).
2. Reconstruct $\hat{\mathbf{x}}$ by sampling from $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$ (decoder). Sample from a Gaussian with parameters $\text{decoder}_\theta(\hat{\mathbf{z}})$ (The Blue part).

[2]Figure comes from "From Autoencoder to Beta-VAE" which is a very extraordinary blog post by Lilian.

# Flow Model 4

## 4.1 Quick Recap and Foreshadowing

Before in the Section Autoregressive Model and Section Variational Autoencoder we have learned two types of models:

- Autoregressive Model: $p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i|x_{<i})$
- Variational Autoencoders: $p_\theta(x) = \int p_\theta(x, z)dz$

Both of them have their own advantages and disadvantages in the evaluation of likelihood and whether the model is able to learn leatent variable or not:

| *Model* | *likelihood* | *Can it learn latent variable?* |
|---|---|---|
| Autoregressive Model | Tractable | Cannot learn |
| Variational Autoencoder | Intractable | Can learn |

Table 1: Comparison of Autoregressive Model and Variational Autoencoder

The main idea of this section is to combine the advantages of both models, which means we want to have a model that is able to learn latent variable and tractable in the evaluation of likelihood. So we want to Map simple distributions (easy to sample and evaluate densities) to complex distributions through an invertible transformation.

Key Idea: **Invertible Transformation!**

A flow model is similar to a variational autoencoder, except that it uses a invertible transformation to map the simple distribution to the complex distribution: $\mathbf{x} = f_\theta(\mathbf{z})$.

## 4.2 Change of Variables

Let's first see a simple example of Change of Variable below:

**Example 4.1 - A simple example of Change of Variable.**

**Q**: Let $Z$ be a uniform random variable $\mathcal{U}[0, 2]$ with density $p(z) = 1/2$ for $z \in [0, 2]$. Now let $X = f(Z) = 4Z$. What is the $p_X(4)$?

**A**: Clearly, $X$ is uniform in $[0, 8]$, so $p_X(4) = 1/8$

So how we can get the distribution of $X$ from the distribution of $Z$ given the transformation $f$? By using the Change of Variable Formula.

**Theorem 4.1 - Change of Variable Formula (1D case).**

If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(h(x)) \left| \frac{dh(x)}{dx} \right| \tag{4.1}$$

Extend to general case in multi-dimensional case:

**Theorem 4.2 - Change of Variable Formula (General case).**

The mapping between $Z$ and $X$, given by $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$. Then the change of variable formula is given by:

$$\begin{aligned} p_X(\mathbf{x}) &= p_Z(\mathbf{f}^{-1}(\mathbf{x})) \left| \det \left( \frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \\ &= p_Z(\mathbf{z}) \left| \det \left( \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1} \text{ (If matrix is invertible)} \end{aligned} \tag{4.2}$$

## 4.3 Normalizing Flow Models

In the Section Variational Autoencoder, we know VAE generate samples form latent variable $Z$ by a transformation of $X$ through a function $f_\theta$. Now we want to learn an invertible function $f_\theta$ to map the simple distribution to the complex distribution which can be shown in the figure below which is similar to the Figure3.1:

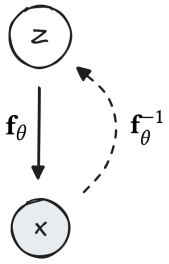


Figure 8: Normalizing Simplified Flow Model

Using Change of Variable Formula (General case) we can get the marginal likelihood $p(\mathbf{x})$ as[3]:

$$p_X(\mathbf{x}; \theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) \left| \det \left( \frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \tag{4.3}$$

[3]Noticed that here $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension

Now with the key idea of flow model and the knowledge of Change of Variable Formula, we can design a flow model to generate samples from a simple distribution to a complex distribution. Not just ONE transformation, but a sequence of transformations shown below!

$$\mathbf{z}_m = \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0) \tag{4.4}$$

By Change of Variable Formula (General case), we can get the marginal likelihood $p(\mathbf{x})$ as follows:

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) \prod_{m=1}^{M} \left| \det \left( \frac{\partial (\mathbf{f}_\theta^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m} \right) \right| \tag{4.5}$$

Extend to the entire datasets, we learning model via maximum likelihood:

$$\max_\theta \log p_X(\mathscr{D};\theta) = \sum_{\mathbf{x}\in\mathscr{D}} \log p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) + \log \left| \det \left( \frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \tag{4.6}$$

As a matter of fact, the determinant of the Jacobian is the complicated which takes $O(n^3)$ time for an $n \times n$ matrix. So the key idea is to choose tranformations so that the resulting Jacobian matrix has special structure. For example, the determinant of a triangular matrix is the product of the diagonal entries, i.e., an $O(n)$ operation.

## 4.4 Different Designs

- NICE(Nonlinear Independent Components Estimation)(Dinh et al., 2014)
- Real-NVP(2016)
- IAF(Invertible Autoregressive Flow)(Kingma et al., 2016)
- MAF(Masked Autoregressive Flow)(Papamakarios et al., 2017)
- I-resnet(Behrmann et al., 2019)
- Glow(Kingma and Dhariwal, 2018)
- Mintnet(Song et al., 2019)

The details of some models will be talked in the future but not now.

### 4.4.1 NICE

NICE or Nonlinear Independent Components Estimation(Dinh et al., 2014) composes two kinds of invertible transformations: additive coupling layers and rescaling layers.

### 4.4.2 Real-NVP

Real-NVP is an Non-volume preserving extension of NICE.

# 5 Generative Adversarial Networks

## 5.1 Likelihood-Free Learning

In many cases, there is always a situation that the test log-likelihood is non-relevant to the actual quality of samples generated by the model. For example, great test log-likelihoods could still leads to poor samples:

---

**Example 5.1.**

---

For a discrtete noise mixture model $p_\theta(\mathbf{x}) = 0.01 p_{data}(\mathbf{x}) + 0.99 p_{noise}(\mathbf{x})$. We can easily see 99% of the samples are just noise. Taking log, we get

$$\begin{aligned} \log p_\theta(\mathbf{x}) &= \log[0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})] \\ &\geq \log 0.01 p_{\text{data}}(\mathbf{x}) \\ &= \log p_{\text{data}}(\mathbf{x}) - \log 100 \end{aligned} \tag{5.1}$$

Combing with non-negativity of KL-Divergence, we get:

$$E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x}))] \geq E_{p_{\text{data}}}[\log p_\theta(\mathbf{x})] \geq E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x})] - \log 100 \tag{5.2}$$

Thus as we increase the dimension $n$ of $\mathbf{x} = (x_1, \ldots, x_n)$, likelihoods are great $E_{p_{\text{data}}}[\log p_\theta(\mathbf{x})] \approx E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x})]$ in very high dimensions.

---

Therefore, we want to consider an alternaive training objective that do not depend directly on a likelihood function which is called Likelihood-free Learning. For example, we can compare two distributions' difference in means or variances:

$$T(S_1, S_2) = \left| \frac{1}{|S_1|} \sum_{x \in S_1} x - \frac{1}{|S_2|} \sum_{x \in S_2} x \right| \tag{5.3}$$

Such test stastic is **likelihood-free** since it does not involve two distributions' densities. However, above test stastic is not a good choice beacause it's too week to distinguish between two distributions. So the **Key Idea** is to learn a statistic to automatically identify in what way the two sets of samples $S_1$ and $S_2$ differ from each other.

Key Idea: **Train a Classifer/Discriminator!**

## 5.2 Generative Adversarial Networks

Combing discriminator with generator, we can build a simplified model structure of GAN which is shown below picture:

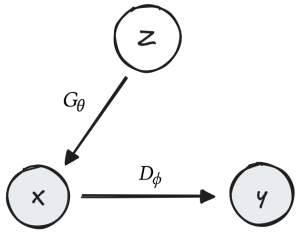


Figure 9: A simplified structure of GAN

In the generative adversarial network, $D_\phi$ is the discriminator which is trained to maximize the test stastic, or equivalently minimize calssfication loss:

$$\begin{aligned} \max_{D_\phi} V(p_\theta, D_\phi) = \quad & E_{\mathbf{x}\sim p_{\text{data}}}[\log D_\phi(\mathbf{x})] + E_{\mathbf{x}\sim p_\theta}[\log(1 - D_\phi(\mathbf{x}))] \\ \approx \quad & \sum_{\mathbf{x}\in S_1} \log D_\phi(\mathbf{x}) + \sum_{\mathbf{x}\in S_2} [\log(1 - D_\phi(\mathbf{x}))] \end{aligned} \tag{5.4}$$

It can be easily sloved that the optimal discriminator is:

$$D^*_\theta(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_\theta(\mathbf{x})} \tag{5.5}$$

$G_\theta$ is the generator is trained to minimize the test stastic which in other words is trying to let discriminator cannot distinguish between real and fake samples. Thus the training objective of GAN is a two player minmax game between a generator and a discriminator:

$$\min_G \max_D V(G, D) = E_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x}\sim p_G}[\log(1 - D(\mathbf{x}))] \tag{5.6}$$

Plug in the optimal discriminator $D^*_\theta(\cdot)$ in Equation5.5, we have[4]:

$$\begin{aligned} V(G, D^*_G(\mathbf{x})) &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}\right] \\ &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] - \log 4 \\ &= \underbrace{D_{KL}\left[p_{\text{data}}, \frac{p_{\text{data}} + p_G}{2}\right] + D_{KL}\left[p_G, \frac{p_{\text{data}} + p_G}{2}\right]}_{2\times\text{Jensen-Shannon Divergence (JSD)}} - \log 4 \\ &= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4 \end{aligned} \tag{5.7}$$

Futhermore, plug in the optimal generator, we get:

$$V(G^*, D^*_G(\mathbf{x})) = -\log 4 \tag{5.8}$$

[4]For the definition of Jensen Shannon Divergence, see Jensen-Shannon Divergence in Appendix B

**Algorithm 2:** Generative Adversarial Networks Algorithm

**for** *number of training iterations* **do**
  **for** *k steps* **do**
    Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data distribution $p_{data}(x)$.
    Update the discriminator parameters $\phi$ by ascending its stochastic gradient:

$$\nabla_\phi V(G_\theta, D_\phi) = \nabla_\phi \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_\phi(\mathbf{x}^{(i)}) + \log \left( 1 - D_\phi(G_\theta(\mathbf{z}^{(i)})) \right) \right]$$

  Update the generator parameters $\theta$ by descending its stochastic gradient:

$$\nabla_\theta V(G_\theta, D_\phi) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})))$$

However, GAN faces a very serious problem: it is very diffcult to train in practice. In fact, GAN is very unstable during optimaziation and a bigger problem whic is called "Mode Collapse" is often observed. Mode collapse refers to a phenomenon where the generator of a GAN collapses to one or few samples. Also, to deal with these diffculties, there are many tricks to train a GAN. Most of tricks are collected in this page: How to Train a GAN? by Soumith Chintala.

## 5.3 Classes

There are many classes of GANs which can almost build a *ZOO* of GANs[5]. But here we only show several classic and popular ones.

#TODO: Finish this subsection later.

### 5.3.1 f-GANs

From all above contents, we have already met two divergences: Jensen-Shannon Divergence and KL divergence. These two divergences are all special cases of one divergence: f-divergence [6].

### 5.3.2 WGAN

### 5.3.3 BiGAN

### 5.3.4 CycleGANs

[5] https://github.com/hindupuravinash/the-gan-zoo The GAN Zoo: List of all named GANs
[6] For example: KL divergence is a special case of f-divergence with $f(x) = x \log x$.

# 6 Energy-Based Generative Model

## 6.1 Parameterizing Probability Distributions

Now, forget all above knowledge and focus on the key building block of generative models: Probability distributions $p(x)$ which must satisfy the following two properties:

- $p(x)$ is sum-to-one: $\sum_x p(x) = 1$.
- $p(x)$ is non-negative: $p(x) \geq 0$.

The second property is important because it means the total "volume" of the distribution is fixed. However, we can never know the exact $p(x)$ but only get $g(x)$ through the generative model. $g_\theta(x)$ is easy to satisfy the second property, but $g_\theta(x)$ might not sum-to-one. So, a simple idea is just divide the volume to reach the onjective of normalization:

$$p_\theta(\mathbf{x}) = \frac{1}{Z(\theta)} g_\theta(\mathbf{x}) = \frac{1}{\int g_\theta(\mathbf{x}) \mathrm{d}\mathbf{x}} g_\theta(\mathbf{x}) = \frac{1}{Volume(g_\theta)} g_\theta(\mathbf{x}) \tag{6.1}$$

Typically, choose $g_\theta(x)$ so that we know the volume analytically. More complex models can be obtained by combining these building blocks. For some reasons below:

- To capture very large variations in probability.
- Exponential families. Many common distributions can be written in this form.
- These distributions arise under fairly general assumptions in statistical physics (maximum entropy, second law of thermodynamics).

we want exponential form:

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x})) d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x})) \tag{6.2}$$

Now, we can choose any function $f_\theta(x)$ we want, but it becomes hard to sample, evaluate, and optimize because computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of x.

# 7 Score-Based Generative Model

## 7.1 Denoising Score Matching

## 7.2 Sliced Score Matching

## 7.3 Noise Conditional Score Network

Denoising Score Matching in Section7.1 is the training method of Noise Conditional Score Network.

# A Maximum Likelihood Learning

# B Mathmatical Theorems

**Theorem B.1 - Jensen Inequality.**

Let $f$ be continuous and convex on an interval $I$. If $x_1, \cdots, x_n$ are in $I$ and $0 < t_1, t_2, \cdots, t_n < 1$ with $t_1 + t_2 + \cdots + t_n = 1$, then

$$f(t_1 x_1 + t_2 x_2 + \cdots + t_n x_n) \leq t_1 f(x_1) + t_2 f(x_2) + \cdots + t_n f(x_n) \tag{B.1}$$

**Definition B.1 - KL divergence.**

The Kullback-Leibler (KL) divergence (also called relative entropy and I-divergence), denoted $D_{\mathrm{KL}}(P \parallel Q)$, is a type of statistical distance: a measure of how one reference probability distribution P is different from a second probability distribution Q:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{B.2}$$

**Definition B.2 - Jensen-Shannon Divergence.**

Let $P$ and $Q$ be probability distributions over the same sample space. The Jensen-Shannon divergence between $P$ and $Q$ is defined as

$$D_{\mathrm{JS}}(P, Q) = \frac{1}{2} D_{\mathrm{KL}}(P \parallel M) + \frac{1}{2} D_{\mathrm{KL}}(Q \parallel M) \tag{B.3}$$

where $M$ is the mean of $P$ and $Q$: $M = \frac{P+Q}{2}$

**Definition B.3 - f-divergence.**

Given two densities $p$ and $q$, the f -divergence is given by

$$D_f(p, q) = E_{\mathbf{x} \sim q}\left[f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)\right] \tag{B.4}$$

where $f$ is any convex, lower-semicontinuous function with $f(1) = 0$.

**Definition B.4 - Fenchel Conjugate.**

For any function $f(\cdot)$, its convex conjugate is

$$f^*(t) = \sup_{u \in \text{dom}_f} (ut - f(u)) \tag{B.5}$$

where $\text{dom}_f$ is the domain of $f$.

# B References


Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., & Jacobsen, J.-H. (2019). Invertible residual networks. *International conference on machine learning*, 573–582.

Dinh, L., Krueger, D., & Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Frey, B. J., Hinton, G. E., & Dayan, P. (1995). Does the wake-sleep algorithm produce good density estimators? *Advances in neural information processing systems*, *8*.

Germain, M., Gregor, K., Murray, I., & Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. *International conference on machine learning*, 881–889.

Kingma, D. P. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, *31*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, *29*.

Larochelle, H., & Murray, I. (2011, November). The neural autoregressive distribution estimator. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 29–37, Vol. 15). PMLR.

Ng, A., & Jordan, M. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, *14*.

Papamakarios, G., Pavlakou, T., & Murray, I. (2017). Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, *30*.

Song, Y., Kim, T., Nowozin, S., Ermon, S., & Kushman, N. (2017). Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*.

Song, Y., Meng, C., & Ermon, S. (2019). Mintnet: Building invertible neural networks with masked convolutions. *Advances in Neural Information Processing Systems*, *32*.

Uria, B., Murray, I., & Larochelle, H. (2014). Rnade: The real-valued neural autoregressive density-estimator. https://arxiv.org/abs/1306.0186

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., et al. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, *12*.

Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016). Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, *29*.

Van Den Oord, A., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *International conference on machine learning*, 1747–1756.