Generative Adversarial Network (GAN)

© Xiyuan Chen

2023

Information

- Syllabus can be found through this link.
- This note is based on videos uploaded in 2018.
- Videos are available on YouTube but I personally watched them on BiliBili.

Contents

1 Introduction

1.1 What is GAN?

Generative Adversarial Network (GAN) is a Deep-Learning-based generative model.

1.2 Idea Overview

GAN has two essential components: a **Generator** and a **Discriminator**.

A Generator is a NN or function that takes an arbitrary **vector** and output object(image, text, speech, etc.). We use it to produce the final object.

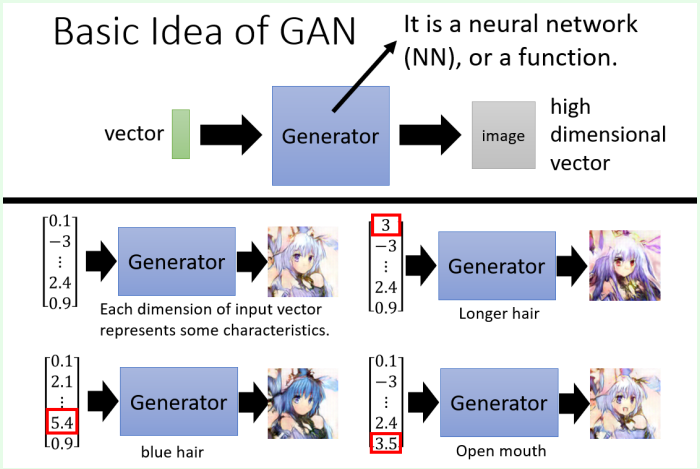

Figure 1: Basic idea for an image Generator

The input vector usually comes from a known distribution, such as Gaussian Dist, so it is randomly sampled. The intuition is that the entries of the vector indicate different latent features of the object, so ideally, we can change one of the entries to control that specific feature.
简单来说就是生成器吃一个 vector 然后吐一个目标物品，比如一张图片。这个 vector 的每个维度可能对应目标物品的某个特征，比如眼睛的颜色。所以我们其实是学习一个 function，把一个随机的向量当成生成参数然后投影到目标物品所在的分布中的一个点 (高维的)。这个点就是我们生成的物品。

However, the Generator does not know how well the object it produces (We cannot just minimize the L2 distance between the produced images and real images). Therefore we need someone to tell our Generator what's wrong with the product.

A Discriminator is a NN or function that takes an **object**(maybe smt else we will see later) and output a scalar usually ranging from 0 to 1. This scalar represents how real the input object is compared to our database, which contains real data.

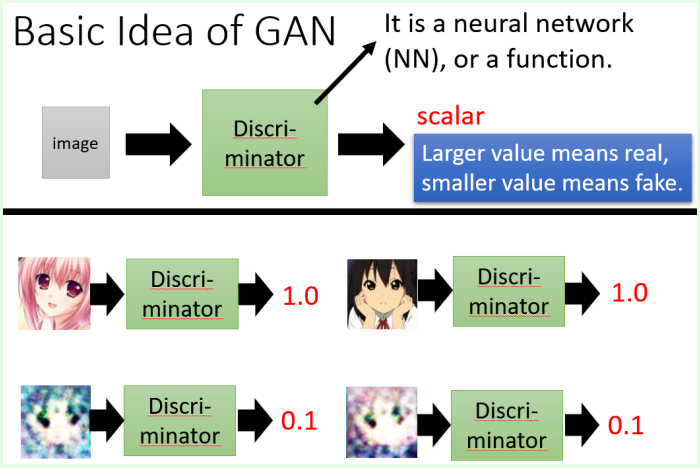

Figure 2: Basic idea for an image Discriminator

A larger output value indicates that the input object is more likely real. We train the Discriminator so that it gives high scores to objects in the database and low scores to the ones produced by our Generator.
判别器用于打分，分越高说明越不像生成的，也就越好。在训练的时候我们知道输入是生成的还是真实的，所以判别器会相对应地调整学习，导致对输入的要求越来越高 (对生成器越来越严格)。

The main idea/procedure of GAN is:

i. Sample a vector from any distribution.

ii. Generate an object from the Generator.

iii. Rate the output by the Discriminator.

iv. Discriminator learns parameters to give high scores to real images and low scores to fake images.

v. Generator learns parameters to produce objects that can receive high scores from the Discriminator.

vi. Repeat the steps above until we are satisfied with the Generator's output.

This workflow is very similar to biological evolution, where predator and prey make adjustments to cheat and identify each other, respectively.

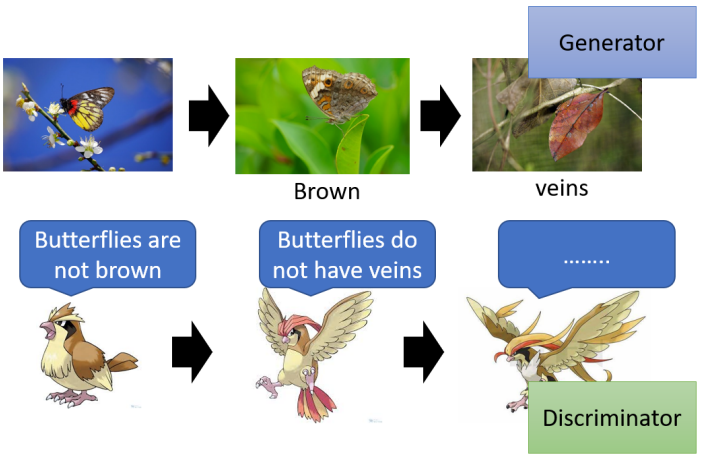

Figure 3: GAN Evolution

2 Basic Theory

2.1 Generation

Suppose we want to generate an image. An image is just a high-dimensional vector. Among all the possible locations(the entire vector space), our desired images(say anime girls) have a complicated but fixed distribution, denoted as $P_{data}(x)$ in Figure **??**.

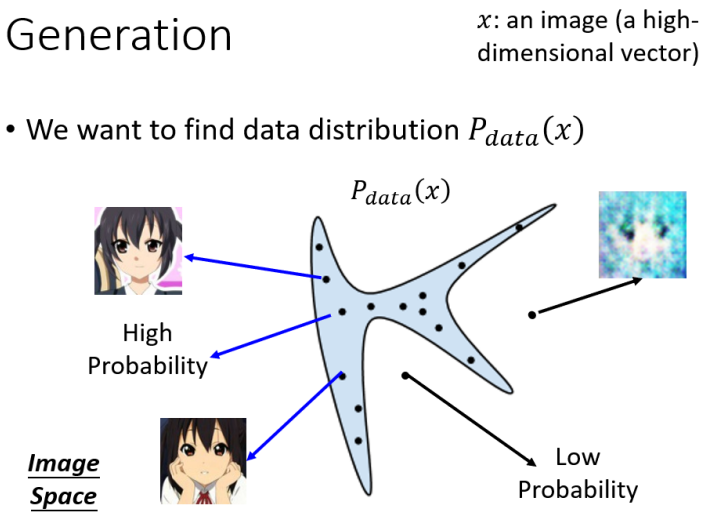

Figure 4: Vector Space for Image Generation

Only images within a certain area(the blue section) look like anime girls. Our goal is to learn such a distribution so that we can generate anime girl images by just sampling instances from this distribution.

2.2 Before GAN: Maximum Likelihood Estimation

MLE is a statistical method to find proper parameters that have the highest probability to produce the outcome of sampled data.

- Given a data distribution $P_{data}(x)$ (We do not know its distribution, but we can sample data from it).
- We want to find parameter θ such that distribution $P_G(x;\theta)$ close to $P_{data}(x)$. E.g. $P_G(x;\theta)$ is a Gaussian Mixture Model, θ are means and variances of the Gaussians.

How do we find it using MLE:

1. Sample $x^1, x^2, ..., x^m$ from $P_{data}(x)$.
2. Compute $P_G(x^i;\theta)$ for a specific θ.
3. Compute a likelihood value for this θ:
$$L = \prod_{i=1}^{m} P_G(x^i;\theta) \tag{1}$$
Multiplying all the probabilities is just one way to do maximization. We just want each probability to be as large as possible. So theoretically we can use average or smt else.
4. Find θ^* that maximize L.

Relation between Maximum Likelihood Estimation and Minimum KL Divergence:
Brief description of MKLD: Kullback-Leibler divergence, or KL divergence, is a measure of the difference between two probability distributions. Minimizing KL divergence means finding the values of the distribution that minimize the difference between the two distributions. This is often used in machine learning to find the best parameters for a model by minimizing the difference between the predicted distribution and the true distribution of the data.
Proof:(MLE = MKLD)

$$\begin{aligned}
\theta^* &= \arg\max_\theta \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= \arg\max_\theta log \prod_{i=1}^{m} P_G(x^i;\theta) \quad \text{(Adding } log \text{ does not change the maximization problem.)} \\
&= \arg\max_\theta \prod_{i=1}^{m} logP_G(x^i;\theta) \\
&\approx \arg\max_\theta \mathbb{E}_{x\sim P_{data}}[logP_G(x;\theta)] \\
&= \arg\max_\theta \int_x P_{data}(x) logP_G(x;\theta)dx - \int_x P_{data}(x) logP_{data}(x;\theta)dx \\
&\quad \text{(The later part is just a constant so we are still good)} \\
&= \arg\min_\theta KL(P_{data}||P_G)
\end{aligned}$$

Limitation:

We do **NOT** know $P_G(x^i,\theta)$ at all. There is no way to compute the probability for different data x^i. So this method won't work in most real-world scenarios.

2.3 Generator

A generator G is a network. The network defines a probability distribution P_G. It maps one distribution to another arbitrary distribution.

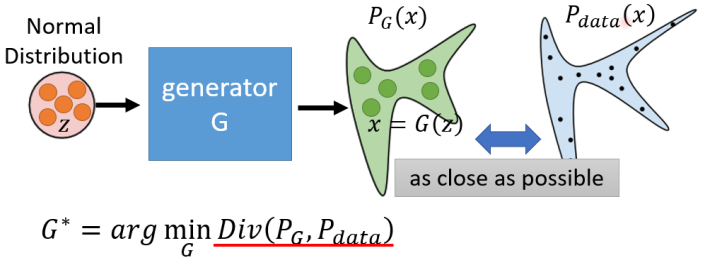

Figure 5: A Generator works as a distribution transformer.

We want the result distribution to be as close as possible to the real distribution. Therefore we need to find one that can minimize **some kind of divergence** $Div(P_G, P_{data})$ between the two distributions. This divergence could be any measurement(different models have different ways of representing the divergence).

Up to this point, we still don't know P_G and P_{data}, so we cannot compute the divergence directly. This is the job of a discriminator.

2.4 Discriminator

Although we do not know the distributions of P_G and P_{data}, we can sample from them. The job of a discriminator D is to measure the divergence between data sampled from P_G and data sampled from P_{data}.
The **objective function** for D is:

$$V(G,D) = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}[log(1-D(x))] \tag{2}$$

Maximize $D(x)$ from P_{data}.
Minimize $D(x)$ from P_G

When training, we want to find a D^* such that it maximize $V(G,D)$.

What is $V(G,D)$ exactly?

We will see that $V(G,D)$ is related to a famous divergence measurement called ***Jensen-Shannon divergence***(JS divergence).

Proof:

$$\begin{aligned} V(G,D) &= \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}[log(1-D(x))] \\ &= \int_x P_{data}(x)logD(x)dx + \int_x P_G(x)log(1-D(x))dx \\ &= \int_x [P_{data}(x)logD(x) + P_G(x)log(1-D(x))]dx \end{aligned}$$

So given x, the optimal D^* maximizing

$$\underset{a}{P_{data}(x)}\underset{D}{logD(x)} + \underset{b}{P_G(x)}\underset{D}{log(1-D(x))}$$

Proof($\Longrightarrow$): Find D^* maximizing $f(D) = alog(D) + Blog(1-D)$.

$$\frac{df D}{dD} = a \times \frac{1}{D} + b \times \frac{1}{1-D} \times (-1) = 0$$

$$\Longrightarrow D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= \mathbb{E}_{x \sim P_{data}} \left[log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \right] + \mathbb{E}_{x \sim P_G} \left[log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \right]$$

$$= \int_x P_{data}(x) log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} dx + \int_x P_G(x) log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} dx$$

$$= \int_x P_{data}(x) log \frac{\frac{1}{2} \cdot P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx + \int_x P_G(x) log \frac{\frac{1}{2} \cdot P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx$$

$$= -2log2 + \int_x P_{data}(x) log \frac{P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx + \int_x P_G(x) log \frac{P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx$$

$$= -2log2 + KL \left(P_{data} || \frac{P_{data} + P_G}{2} \right) + KL \left(P_G || \frac{P_{data} + P_G}{2} \right)$$

$$= -2log2 + 2JSD(P_{data} || P_G) \quad \blacksquare$$

In practice, we do **NOT** know $\mathbb{E}$ for both P_{data} and P_G. So we sample $\{x^1, x^2, \ldots, x^m\}$ from $P_{data}(x)$, and sample $\{\widetilde{x}^1, \widetilde{x}^2, \ldots, \widetilde{x}^m\}$. Then maximize

$$\widetilde{V} = \frac{1}{m} \sum_{i=1}^{m} logD(x^i) + \frac{1}{m} \sum_{i=1}^{m} log(1 - D(\widetilde{x}^i)) \tag{3}$$

Equation **??** can be seen as the objective function of a binary classifier. It is **equal** to **minimize Cross-entropy**.

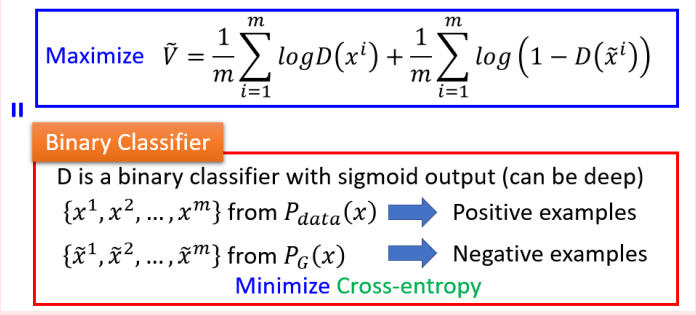

Figure 6: Relation with CE

2.5 Algorithm

So now we know $V(G, D)$ is a divergence measurement, we can replace $Div(P_G, P_{data})$ in Section **??** by

$$\max_D V(G, D)$$

and get

$$G^* = \arg\min_G \max_D V(G, D) \tag{4}$$

How to calculate Equation **?? in algorithm:**

- Initialize generator and discriminator
- In each training iteration:
  1. Fix generator G, and update discriminator D
  2. Fix discriminator D, and update generator G

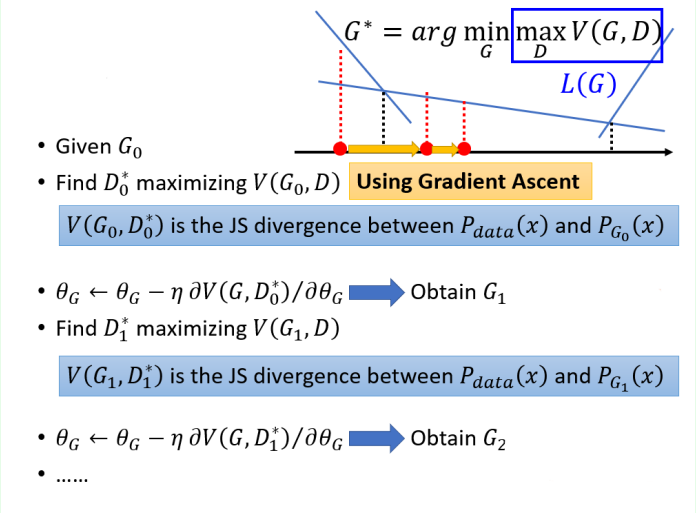

Figure 7: Algorithm Breakdown