Generative Adversarial Network

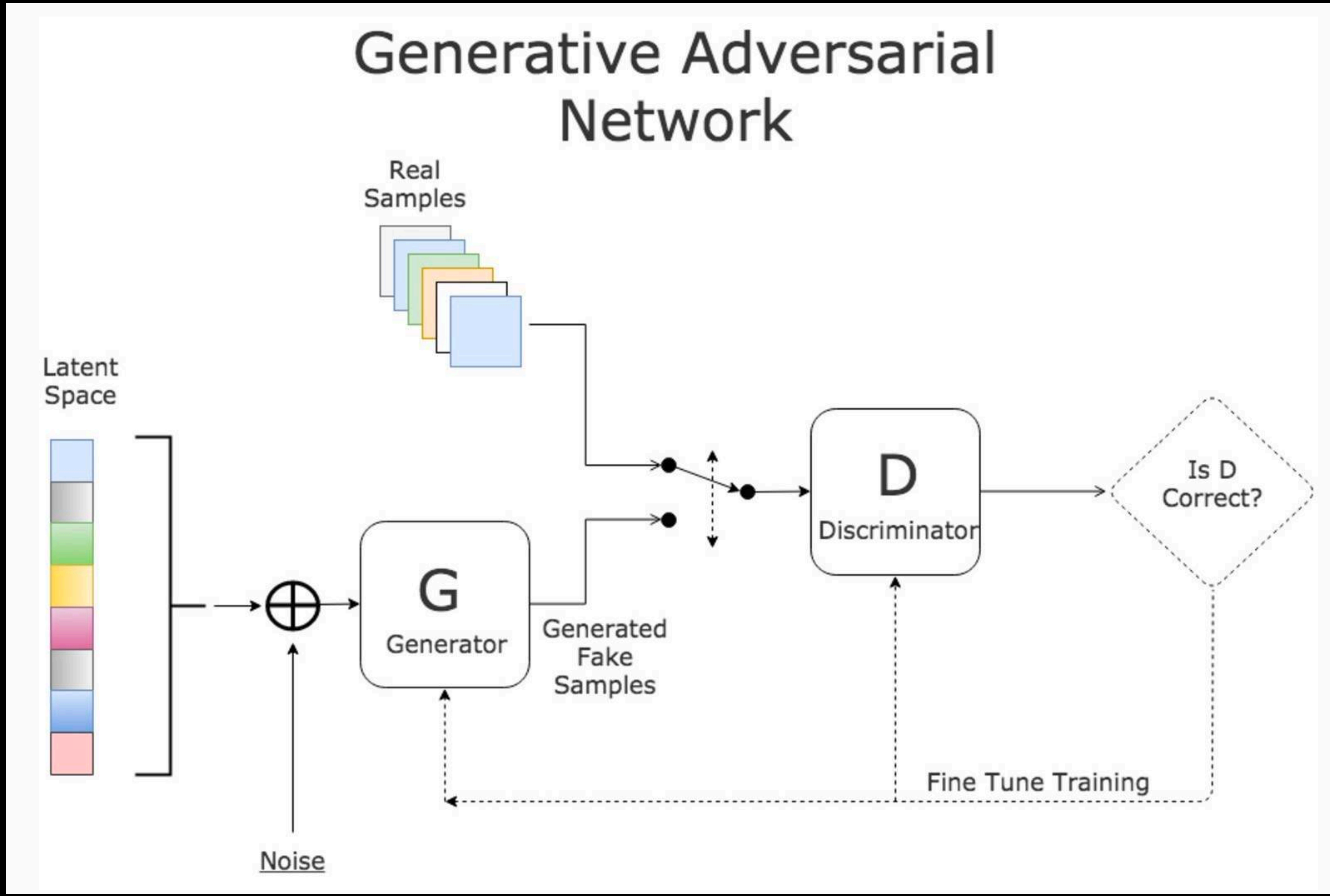
生成对抗网络

贾鑫康

Table

- GAN概述
- 原理推导
  - 前置知识
  - 最优G和D
  - 训练过程
- GAN存在的问题
- 总结

GAN概述



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

前置知识

- KL散度：
  - 香浓熵：
    $$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)]$$
  - KL散度：
    $$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}[\log P(x) - \log Q(x)]$$
    - 特性：非负性，非对称性

前置知识

- 推导中的问题：
  - 可逆条件的忽略

$$\int_x p_{data}(x)\log D(x)\,\mathrm{d}x + \int_z p(z)\log(1 - D(G(z)))\,\mathrm{d}z$$
$$= \int_x p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x))\,\mathrm{d}x$$

$$E_{z\sim p_z(z)}\log(1 - D(G(z))) = E_{x\sim p_G(x)}\log(1 - D(x))$$

最优G和D

- 最优鉴别器的推导

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

$$V(G,D) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx$$

$$f(y) = a\log y + b\log(1-y)$$

最优G和D

- 最优鉴别器的推导

$$f'(y) = 0 \Rightarrow \frac{a}{y} - \frac{b}{1-y} = 0 \Rightarrow y = \frac{a}{a+b}$$

$$f''\left(\frac{a}{a+b}\right) = -\frac{a}{(\frac{a}{a+b})^2} - \frac{b}{1-(\frac{a}{a+b})^2} < 0$$

$$\frac{p_{data}}{p_{data} + p_G}$$

$$V(G, D) = \int_x p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)) \, dx$$
$$\le \int_x \max_y p_{data}(x) \log y + p_G(x) \log(1-y) \, dx.$$

最优G和D

- 最优生成器的推导

$$D_G^* = \frac{p_{data}}{p_{data} + p_G} = \frac{1}{2}.$$

自 P_data 和 P_G 的概率都为 1/2。基于这一观点，GAN 作者证明了 G 就是极小极大博弈的解。
该定理如下
「当且仅当 P_G=P_data，训练标准 C(G)=maxV(G,D) 的全局最小点可以达到。」

最优G和D

- 最优生成器的反向推导

$$V(G, D_G^*) = \int_x p_{data}(x) \log \frac{1}{2} + p_G(x) \log \left(1 - \frac{1}{2}\right) \mathrm{d}x$$

and

$$V(G, D_G^*) = -\log 2 \int_x p_G(x) \,\mathrm{d}x - \log 2 \int_x p_{data}(x) \,\mathrm{d}x = -2 \log 2 = -\log 4.$$

最优G和D

- 最优生成器的正向推导

$$C(G) = -\log 2 \int_x p_G(x) + p_{data}(x)\,\mathrm{d}x$$

$$+\int_x p_{data}(x)\Big(\log 2 + \log\big(\frac{p_{data}(x)}{p_G(x) + p_{data}(x)}\big)\Big)$$

$$+p_G(x)\Big(\log 2 + \log\big(\frac{p_G(x)}{p_G(x) + p_{data}(x)}\big)\Big)\,\mathrm{d}x.$$

最优G和D

- 最优生成器的正向推导

$$C(G) = -\log 4 + \int_x p_{data}(x) \log \left(\frac{p_{data}(x)}{(p_G(x) + p_{data}(x))/2} \right) \mathrm{d}x$$
$$+ \int_x p_G(x) \log \left(\frac{p_G(x)}{(p_G(x) + p_{data}(x))/2} \right) \mathrm{d}x.$$

$$C(G) = -\log 4 + KL\left(p_{data} \middle| \frac{p_{data} + p_G}{2}\right) + KL\left(p_G \middle| \frac{p_{data} + p_G}{2}\right)$$

最优G和D

- 最优生成器的正向推导

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

$$M = \frac{1}{2}(P + Q)$$

$$C(G) = -\log 4 + 2 \cdot JSD\left(p_{data} | p_G\right)$$

训练过程

- 参数优化过程：
  - 梯度下降的优化过程：$\theta_G \leftarrow \theta_G - \eta\, \partial L(G)/\partial\theta_G$
  - 给定 G_0，最大化 V(G_0,D) 以求得 D_0*，即 max[JSD(P_data(x)||P_G0(x)]；
  - 固定 D_0*，计算θ_G1 ← θ_G0 –η(∂V(G,D_0*) /∂θ_G) 以求得更新后的 G_1；
  - 固定 G_1，最大化 V(G_1,D_0*) 以求得 D_1*，即 max[JSD(P_data(x)||P_G1(x)]；
  - 固定 D_1*，计算θ_G2 ← θ_G1 –η(∂V(G,D_0*) /∂θ_G) 以求得更新后的 G_2；
  - 。。。

训练过程

- 实际训练过程：
  - P_data(x) 采样 m 个样本 $\{x^1, x^2, \ldots, x^m\}$，从生成器 P_G(x) 采样 m 个样本 。最大化价值函数 V(G,D) 就可以使用以下表达式近似替代：

$$\text{Maximize} \quad \tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD\left(x^i\right) + \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D\left(\tilde{x}^i\right)\right)$$

$\{x^1, x^2, \ldots, x^m\}$ from $P_{data}(x)$ → 正样本

$\{\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^m\}$ from $P_G(x)$ → 负样本

$$\text{Minimize} \; L = -\frac{1}{m}\sum_{i=1}^{m} logD\left(x^i\right) - \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D\left(\tilde{x}^i\right)\right)$$

训练过程

- 鉴别器D学习过程：
  - 从真实数据分布 P_data 抽取 m 个样本
  - 从先验分布 P_prior(z) 抽取 m 个噪声样本
  - 将噪声样本投入 G 而生成数据 $\{\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^m\}, \tilde{x}^i = G(z^i)$，通过最大化 V 的近似而更新判别器参数θ_d，且判别器参数的更新迭代式为 $\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$

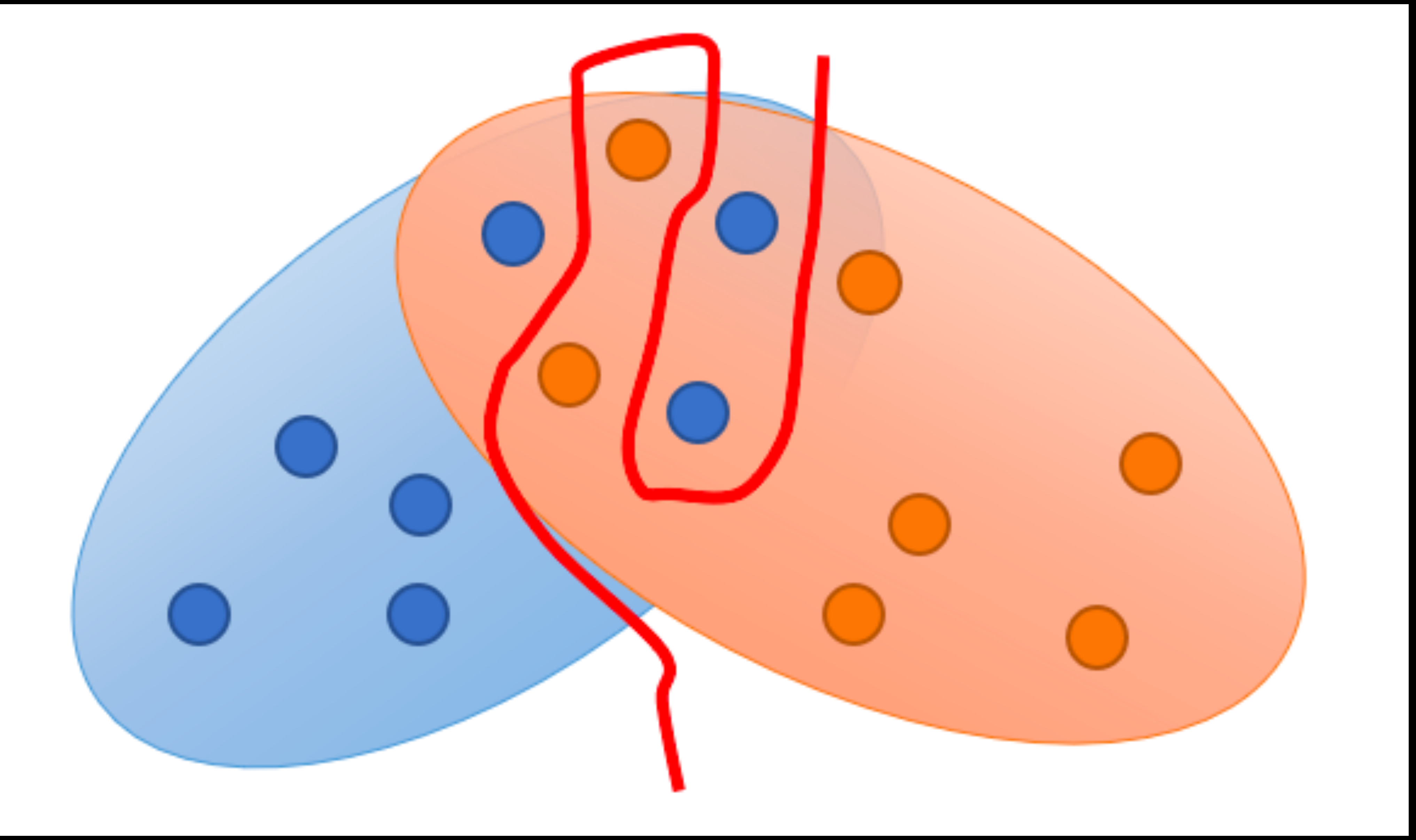
训练过程

- 生成器G学习过程：
    - 从先验分布 P_prior(z) 抽取 m 个噪声样本
    - 通过极小化 V 而更新生成器参数θ_g，即极小化 $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D\left(G\left(z^i\right)\right)\right)$ ，且生成器参数的更新迭代式为 $\theta_g \leftarrow \theta_g - \eta\nabla\tilde{V}\left(\theta_g\right)$

GAN存在的问题

- 训练不稳定：

$$\max_D V(G, D) = -2\log 2 + 2\,\underbrace{\mathbf{JSD}(P_{data}(x)||P_G(x))}\,\log 2 = 0$$



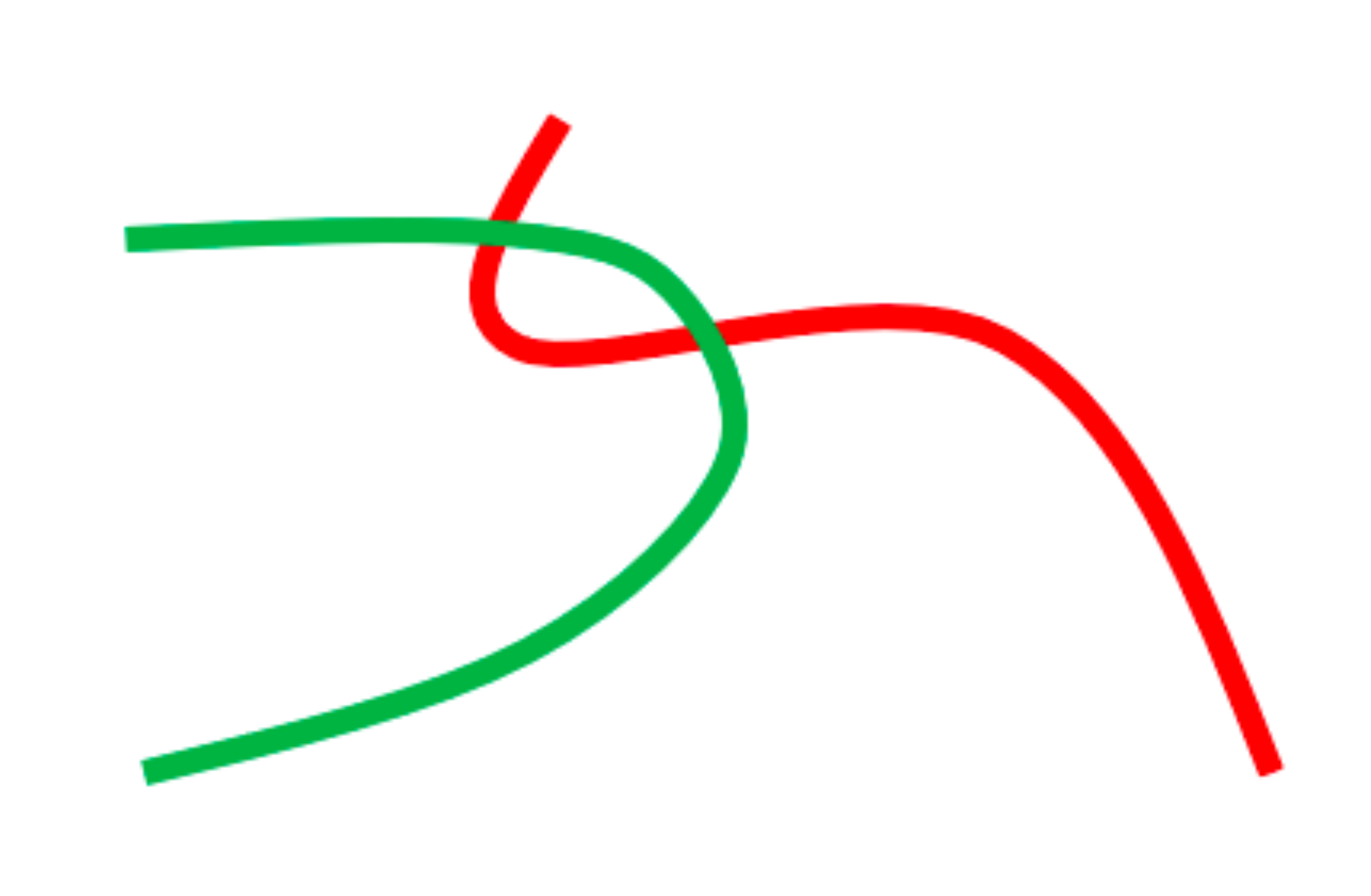
情况一：过拟合

- 传统的正则化方法（regularization等）
- 减少模型的参数让它变得弱一些

GAN存在的问题

- 训练不稳定：

$$\max_D V(G, D) = -2\log 2 + 2\,\underbrace{\mathbf{JSD}(P_{data}(x)||P_G(x))}\,\log 2 = 0$$



情况二：数据本身

- 给真实数据增加噪声
- 改进lossfunction，参考wgan

GAN存在的问题

- 模式崩溃：
- 所有的输出都一样！
- 原因可能是由于真实数据在空间中很多地方都有一个较大的概率值，但是我们的生成模型没有直接学习到真实分布的特性。
- 为了保证最小化损失，宁可永远输出一样但是肯定正确的输出，也不尝试其他不同但可能错误的输出。
- 生成器有时可能无法兼顾数据分布的所有内部模式，只会保守地挑选出一个肯定正确的模式。

Code

- 数据集的准备

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os
from tensorflow.examples.tutorials.mnist import input_data

sess = tf.InteractiveSession()

mb_size = 128
Z_dim = 100

mnist = input_data.read_data_sets('../../MNIST_data', one_hot=True)
```

Code

- 变量的声明

```
def weight_var(shape, name):
    return tf.get_variable(name=name, shape=shape, initializer=tf.contrib.layers.xavier_initializer())


def bias_var(shape, name):
    return tf.get_variable(name=name, shape=shape, initializer=tf.constant_initializer(0))


# discriminater net

X = tf.placeholder(tf.float32, shape=[None, 784], name='X')

D_W1 = weight_var([784, 128], 'D_W1')
D_b1 = bias_var([128], 'D_b1')

D_W2 = weight_var([128, 1], 'D_W2')
D_b2 = bias_var([1], 'D_b2')


theta_D = [D_W1, D_W2, D_b1, D_b2]


# generator net

Z = tf.placeholder(tf.float32, shape=[None, 100], name='Z')

G_W1 = weight_var([100, 128], 'G_W1')
G_b1 = bias_var([128], 'G_B1')

G_W2 = weight_var([128, 784], 'G_W2')
G_b2 = bias_var([784], 'G_B2')

theta_G = [G_W1, G_W2, G_b1, G_b2]
```

Code

- 模型定义

```
def generator(z):
    G_h1 = tf.nn.relu(tf.matmul(z, G_W1) + G_b1)
    G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
    G_prob = tf.nn.sigmoid(G_log_prob)

    return G_prob


def discriminator(x):
    D_h1 = tf.nn.relu(tf.matmul(x, D_W1) + D_b1)
    D_logit = tf.matmul(D_h1, D_W2) + D_b2
    D_prob = tf.nn.sigmoid(D_logit)
    return D_prob, D_logit

G_sample = generator(Z)
D_real, D_logit_real = discriminator(X)
D_fake, D_logit_fake = discriminator(G_sample)
```

Code

```
D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
G_loss = -tf.reduce_mean(tf.log(D_fake))
```

- 损失函数的定义

OR

```
D_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
    logits=D_logit_real, labels=tf.ones_like(D_logit_real)))
D_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
    logits=D_logit_fake, labels=tf.zeros_like(D_logit_fake)))
D_loss = D_loss_real + D_loss_fake
G_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
    logits=D_logit_fake, labels=tf.ones_like(D_logit_fake)))
```

Code

- 优化

```
D_optimizer = tf.train.AdamOptimizer().minimize(D_loss, var_list=theta_D)
G_optimizer = tf.train.AdamOptimizer().minimize(G_loss, var_list=theta_G)
```

Code

- 训练

```python
def sample_Z(m, n):
    '''Uniform prior for G(Z)'''
    return np.random.uniform(-1., 1., size=[m, n])

sess.run(tf.global_variables_initializer())
for it in range(1000000):
   X_mb, _ = mnist.train.next_batch(mb_size)

    _, D_loss_curr = sess.run([D_optimizer, D_loss], feed_dict={
                              X: X_mb, Z: sample_Z(mb_size, Z_dim)})
    _, G_loss_curr = sess.run([G_optimizer, G_loss], feed_dict={
                              Z: sample_Z(mb_size, Z_dim)})

    if it % 1000 == 0:
        print('Iter: {}'.format(it))
        print('D loss: {:.4}'.format(D_loss_curr))
        print('G_loss: {:.4}'.format(G_loss_curr))
        print()
```