

# 生成模型——生成对抗网络 V2.0\_2019年3月25日更

## 目 录

第一章 初步了解 GANs .....	3
1. 生成模型与判别模型 .....	3
2. 对抗网络思想 .....	3
3. 详细实现过程 .....	3
3.1 前向传播阶段 .....	4
3.2 反向传播阶段 .....	4
4. GANs 大家族分类 .....	6
第二章 GANs 的理论与提升 .....	7
1. GANs 相关理论 .....	7
Part1 GANs 基于 Div 的改进 .....	12
1. GANs 并不完美 .....	12
2. fGAN——深度理解 GAN 理论 .....	15
3. JS Div 不是最佳的 Div .....	19
4. LSGAN .....	20
5. WGAN .....	21
6. WGAN-GP .....	24
7. SNGAN .....	26
Part2 GANs 基于 Network 的改进 .....	30
1. DCGAN .....	30
2. ImprovedDCGAN .....	32
3. SAGAN .....	34
4. BigGAN .....	36
5. S3GAN .....	41
Part3 GANs 的其他改进 .....	44
1. RGAN .....	44
2. EBGAN .....	46
3.* BEGAN .....	47
第三章 GANs 的应用 .....	48
Part1 GANs 在图像生成上的应用 .....	48
1. CGAN .....	48
2. TripleGAN .....	52
3. StackGAN .....	53
4. LapGAN .....	56
5. ProGAN (也称 PGGAN) .....	58
6. StyleGAN .....	60
7.* SRGAN .....	64
Part2 GANs 在风格迁移上的应用 .....	65
1. CycleGAN .....	65
2. StarGAN .....	68
3. 如何解决 BIG TRANSFORMATION .....	71

Part3 GANs 在特征提取上的应用 .....	75
1. InfoGAN .....	75
2. VAEGAN .....	78
3. BiGAN .....	80
第四章 附录 .....	82
1.致谢及引用 .....	82

# 第一章 初步了解 GANs

## 1. 生成模型与判别模型

理解对抗网络，首先要了解生成模型和判别模型。判别模型比较好理解，就像分类一样，有一个判别界限，通过这个判别界限去区分样本。从概率角度分析就是获得样本  $x$  属于类别  $y$  的概率，是一个条件概率  $P(y|x)$ 。而生成模型是需要在整个条件内去产生数据的分布，就像高斯分布一样，需要去拟合整个分布，从概率角度分析就是样本  $x$  在整个分布中的产生的概率，即联合概率  $P(xy)$ 。具体可以参考博文：

<http://blog.csdn.net/zoux09/article/details/8195017>

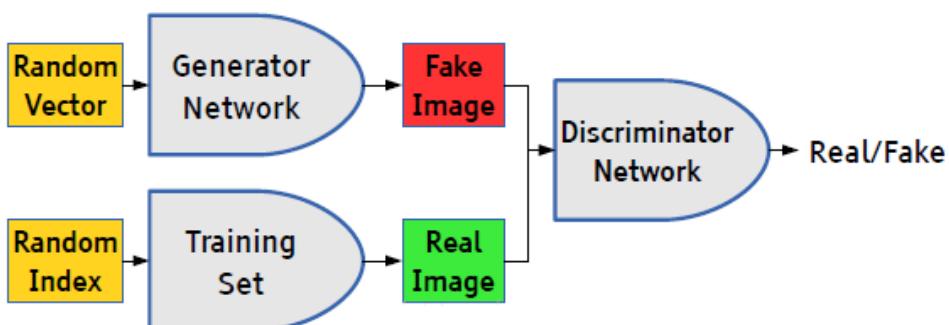
## 2. 对抗网络思想

理解了生成模型和判别模型后，再来理解对抗网络就很直接了，对抗网络只是提出了一种网络结构，总体来说，GANs 简单的想法就是用两个模型，一个生成模型，一个判别模型。判别模型用于判断一个给定的图片是不是真实的图片（从数据集里获取的图片），生成模型的任务是去创造一个看起来像真的图片一样的图片。而在开始的时候这两个模型都是没有经过训练的，这两个模型一起对抗训练，生成模型产生一张图片去欺骗判别模型，然后判别模型去判断这张图片是真是假，最终在这两个模型训练的过程中，两个模型的能力越来越强，最终达到稳态。（本书仅介绍 GANs 在计算机视觉方面的应用，但是 GANs 的用途很广，不单是图像，其他方面，譬如文本、语音，或者任何只要含有规律的数据合成，都能用 GANs 实现。）

## 3. 详细实现过程

假设我们现在的数据集是手写体数字的数据集 minst，生成模型的输入可以是二维高斯模型中一个随机的向量，生成模型的输出是一张伪造的 fake image，同时通过索引获取数据集中的真实手写数字图片 real image，然后将 fake image 和 real image 一同传给判别模型，由判别模型给出 real 还是 fake 的判别结果。于是，一个简单的 GANs 模型就搭建好了。

值得注意的是，生成模型 G 和判别模型 D 可以是各种各样的神经网络，对抗网络的生成模型和判别模型没有任何限制。



## 3.1 前向传播阶段

### 1. 模型输入

1、我们随机产生一个随机向量作为生成模型的数据，然后经过生成模型后产生一个新的向量，作为 Fake Image，记作  $D(z)$ 。

2、从数据集中随机选择一张图片，将图片转化成向量，作为 Real Image，记作  $x$ 。

### 2. 模型输出

将由 1 或者 2 产生的输出，作为判别网络的输入，经过判别网络后输出值为一个 0 到 1 之间的数，用于表示输入图片为 Real Image 的概率，real 为 1，fake 为 0。

使用得到的概率值计算损失函数，解释损失函数之前，我们先解释下判别模型的输入。根据输入的图片类型是 Fake Image 或 Real Image 将判别模型的输入数据的 label 标记为 0 或者 1。即判别模型的输入类型为  $(x_{fake}, 0)$  或者  $(x_{real}, 1)$ 。

## 3.2 反向传播阶段

### 1. 优化目标

原文给了这么一个优化函数： 真样本数据

假样本数据

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

我们来理解一下这个目标公式，先优化  $D$ ，再优化  $G$ ，拆解之后即为如下两步：

**第一步：**优化  $D$

$$\max_D V(D, G) = E_{x \sim p_{data}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

优化  $D$ ，即优化判别网络时，没有生成网络什么事，后面的  $G(z)$  就相当于已经得到的假样本。优化  $D$  的公式的第一项，使得真样本  $x$  输入的时候，得到的结果越大越好，因为真样本的预测结果越接近 1 越好；对于假样本  $G(z)$ ，需要优化的是其结果越小越好，也就是  $D(G(z))$  越小越好，因为它的标签为 0。但是第一项越大，第二项越小，就矛盾了，所以把第二项改为  $1 - D(G(z))$ ，这样就是越大越好。

**第二步：**优化  $G$

$$\min_G V(D, G) = E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

在优化  $G$  的时候，这个时候没有真样本什么事，所以把第一项直接去掉，这时候只有假样本，但是这个时候希望假样本的标签是 1，所以是  $D(G(z))$  越大越好，但是为了统一成  $1 - D(G(z))$  的形式，那么只能是最小化  $1 - D(G(z))$ ，本质上没有区别，只是为了形式的统一。之后这两个优化模型可以合并起来写，就变成最开始的最大最小目标函数了。

我们依据上面的优化目标函数，便能得到如下模型最终的损失函数。

## 2. 判别模型的损失函数

$$-(1-y)\log(1-D(G(z)))+y\log D(x) \quad \text{模型的分类损失}$$

当输入的是从数据集中取出的 real Image 数据时，我们只需要考虑第二部分， $D(x)$  为判别模型的输出，表示输入  $x$  为 real 数据的概率，我们的目的是让判别模型的输出  $D(x)$  的输出尽量靠近 1。

当输入的为 fake 数据时，我们只计算第一部分， $G(z)$  是生成模型的输出，输出的是一张 Fake Image。我们要做的是让  $D(G(z))$  的输出尽可能趋向于 0。这样才能表示判别模型是有区分力的。

相对判别模型来说，这个损失函数其实就是交叉熵损失函数。计算 loss，进行梯度反传。这里的梯度反传可以使用任何一种梯度修正的方法。

当更新完判别模型的参数后，我们再去更新生成模型的参数。

## 3. 生成模型的损失函数

$$(1-y)\log(1-D(G(z)))$$

对于生成模型来说，我们要做的是让  $G(z)$  产生的数据尽可能的和数据集中的数据一样。就是所谓的同样的数据分布。那么我们要做的就是最小化生成模型的误差，即只将由  $G(z)$  产生的误差传给生成模型。

但是针对判别模型的预测结果，要对梯度变化的方向进行改变。当判别模型认为  $G(z)$  输出为真实数据集的时候和认为输出为噪声数据的时候，梯度更新方向要进行改变。

即最终的损失函数为：

$$(1-y)\log(1-D(G(z)))(2*\bar{D}(G(z))-1)$$

其中  $\bar{D}$  表示判别模型的预测类别，对预测概率取整，为 0 或者 1. 用于更改梯度方向，阈值可以自己设置，或者正常的话就是 0.5。

## 4. 反向传播

我们已经得到了生成模型和判别模型的损失函数，这样分开看其实就是两个单独的模型，针对不同的模型可以按照自己的需要去实现不同的误差修正，我们也可以选择最常用的 BP 做为误差修正算法，更新模型参数。

其实说了这么多，生成对抗网络的生成模型和判别模型是没有任何限制，生成对抗网络提出的只是一种网络结构，我们可以使用任何的生成模型和判别模型去实现一个生成对抗网络。当得到损失函数后就安装单个模型的更新方法进行修正即可。

## 4. GANs 大家族分类

随着 GANs 的火热，相关的衍生模型出现了至少有上百种，在下面这个博客 <https://deephunt.in/the-gan-zoo-79597dc8c347> 中整理了非常多的 GANs 变种。本书仅选择与计算机视觉相关的 GANs 作介绍，简要地介绍其核心思想和算法原理。本书共涉及到 31 种经典的 GANs 架构，现在将其按不同类型进行分类，并按时间排列列表如下。

发表时间	名称	中文名称	论文地址
14 年 06 月	GANs	生成对抗网络	<a href="https://arxiv.org/pdf/1406.2661.pdf">https://arxiv.org/pdf/1406.2661.pdf</a>
14 年 11 月	CGAN	条件生成对抗网络	<a href="https://arxiv.org/pdf/1411.1784.pdf">https://arxiv.org/pdf/1411.1784.pdf</a>
15 年 06 月	LAPGAN	拉普拉斯金字塔 GAN	<a href="https://arxiv.org/pdf/1506.05751.pdf">https://arxiv.org/pdf/1506.05751.pdf</a>
15 年 11 月	DCGAN	深度卷积生成对抗网络	<a href="https://arxiv.org/pdf/1511.06434.pdf">https://arxiv.org/pdf/1511.06434.pdf</a>
15 年 12 月	VAEGAN	变分自编码器 GAN	<a href="https://arxiv.org/pdf/1512.09300.pdf">https://arxiv.org/pdf/1512.09300.pdf</a>
16 年 05 月	BiGAN	双向生成对抗网络	<a href="https://arxiv.org/pdf/1605.09782.pdf">https://arxiv.org/pdf/1605.09782.pdf</a>
16 年 06 月	CoGAN	耦合生成对抗网络	<a href="https://arxiv.org/pdf/1606.07536.pdf">https://arxiv.org/pdf/1606.07536.pdf</a>
16 年 06 月	fGAN	f-散度生成对抗网络	<a href="https://arxiv.org/pdf/1606.00709.pdf">https://arxiv.org/pdf/1606.00709.pdf</a>
16 年 06 月	ImprovedDCGAN	提升的 DCGAN	<a href="https://arxiv.org/pdf/1606.03498.pdf">https://arxiv.org/pdf/1606.03498.pdf</a>
16 年 06 月	InfoGAN	互信息生成对抗网络	<a href="https://arxiv.org/pdf/1606.03657.pdf">https://arxiv.org/pdf/1606.03657.pdf</a>
16 年 09 月	EBGAN	基于能量的生成对抗网络	<a href="https://arxiv.org/pdf/1609.03126.pdf">https://arxiv.org/pdf/1609.03126.pdf</a>
16 年 09 月	SRGAN	超分辨率生成对抗网络	<a href="https://arxiv.org/pdf/1609.04802.pdf">https://arxiv.org/pdf/1609.04802.pdf</a>
16 年 11 月	LSGAN	最小二乘生成对抗网络	<a href="https://arxiv.org/pdf/1611.04076.pdf">https://arxiv.org/pdf/1611.04076.pdf</a>
16 年 12 月	StackGAN	堆栈式生成对抗网络	<a href="https://arxiv.org/pdf/1612.03242.pdf">https://arxiv.org/pdf/1612.03242.pdf</a>
17 年 01 月	WGAN	Was 距离生成对抗网络	<a href="https://arxiv.org/pdf/1701.07875.pdf">https://arxiv.org/pdf/1701.07875.pdf</a>
17 年 03 月	BEGAN	边界均衡生成对抗网络	<a href="https://arxiv.org/pdf/1703.10717.pdf">https://arxiv.org/pdf/1703.10717.pdf</a>
17 年 03 月	CycleGAN	循环生成对抗网络	<a href="https://arxiv.org/pdf/1703.10593.pdf">https://arxiv.org/pdf/1703.10593.pdf</a>
17 年 03 月	TripleGAN	三部分生成对抗网络	<a href="https://arxiv.org/pdf/1703.02291.pdf">https://arxiv.org/pdf/1703.02291.pdf</a>
17 年 04 月	WGAN-GP	加强版 WGAN	<a href="https://arxiv.org/pdf/1704.00028.pdf">https://arxiv.org/pdf/1704.00028.pdf</a>
17 年 05 月	DRAGAN	深度回归分析 GAN	<a href="https://arxiv.org/pdf/1705.07215.pdf">https://arxiv.org/pdf/1705.07215.pdf</a>
17 年 10 月	PGGAN	渐进生成对抗网络	<a href="https://arxiv.org/pdf/1710.10196.pdf">https://arxiv.org/pdf/1710.10196.pdf</a>
17 年 10 月	StackGAN++	提升的堆栈式 GAN	<a href="https://arxiv.org/pdf/1710.10916.pdf">https://arxiv.org/pdf/1710.10916.pdf</a>
17 年 11 月	StarGAN	星型结构 GAN	<a href="https://arxiv.org/pdf/1711.09020.pdf">https://arxiv.org/pdf/1711.09020.pdf</a>
17 年 11 月	XGAN	X 型结构 GAN	<a href="https://arxiv.org/pdf/1711.05139.pdf">https://arxiv.org/pdf/1711.05139.pdf</a>
17 年 12 月	ComboGAN	合一式生成对抗网络	<a href="https://arxiv.org/pdf/1712.06909.pdf">https://arxiv.org/pdf/1712.06909.pdf</a>
18 年 02 月	SNGAN	频谱归一化生成对抗网络	<a href="https://arxiv.org/pdf/1802.05957.pdf">https://arxiv.org/pdf/1802.05957.pdf</a>
18 年 05 月	SAGAN	自注意力生成对抗网络	<a href="https://arxiv.org/pdf/1805.08318.pdf">https://arxiv.org/pdf/1805.08318.pdf</a>
18 年 07 月	RGAN	相对生成对抗网络	<a href="https://arxiv.org/pdf/1807.00734.pdf">https://arxiv.org/pdf/1807.00734.pdf</a>
18 年 09 月	BigGAN	大规模生成对抗网络	<a href="https://arxiv.org/pdf/1809.11096.pdf">https://arxiv.org/pdf/1809.11096.pdf</a>
18 年 12 月	StyleGAN	基于样式的生成对抗网络	<a href="https://arxiv.org/pdf/1812.04948.pdf">https://arxiv.org/pdf/1812.04948.pdf</a>
19 年 03 月	S3GAN	更少标签的生成对抗网络	<a href="https://arxiv.org/pdf/1903.02271.pdf">https://arxiv.org/pdf/1903.02271.pdf</a>

上述 31 篇 paper 的打包下载地址如下：

链接: <https://pan.baidu.com/s/1Pq-LpgusBQGHdRpZdB3qQ>

提取码: tpeh

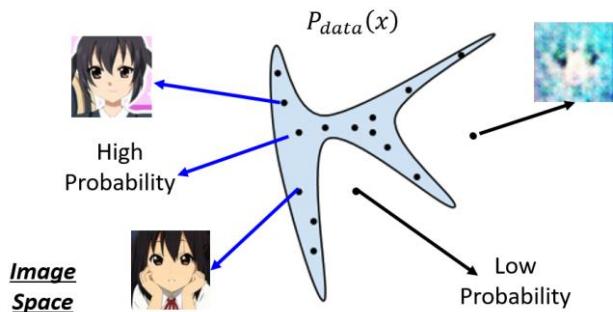
下面将会逐一介绍这些 GANs 变种的实现原理。

## 第二章 GANs 的理论与提升

本章会介绍一些与提升和改进 GANs 相关的 paper, 这涉及到一些与 GANs 理论相关的知识, 在第一部分会用较简短的话语介绍一些 GANs 的相关核心理论, 第二部分是 GANs 存在的一些缺陷和改进手段, 第三部分会介绍一个很有趣的 paper——fGAN, 它能帮助我们更深的理解 GANs 的算法思想, 第四部分开始就是更多的 paper, 它们会用到一些很有意思的 tricks 去提升和改进 GANs。

### 1. GANs 相关理论

GANs 本质上在做的事情是什么?



我们假设把每一个图片看作二维空间中的一个点, 并且现有图片会满足于某个数据分布, 我们记作 $P_{data}(x)$ 。以人脸举例, 在很大的一个图像分布空间中, 实际上只有很小一部分的区域是人脸图像。如上图所示, 只有在蓝色区域采样出的点才会看起来像人脸, 而在蓝色区域以外的区域采样出的点就不是人脸。今天我们需要做的, 就是让机器去找到人脸的分布函数。具体来说, 就是我们会有很多人脸图片数据, 我们观测这些数据的分布, 大致能猜测到哪些区域出现人脸图片数据的概率比较高, 但是如果让我们找出一个具体的定义式, 去给明这些人脸图片数据的分布规律, 我们是没有办法做到的。但是如今, 我们有了机器学习, 希望机器能够学习到这样一个分布规律, 并能够给出一个极致贴合的表达式。

在 GANs 出现之前, 人们采用的方法是 Maximum Likelihood Estimation。

### Maximum Likelihood Estimation

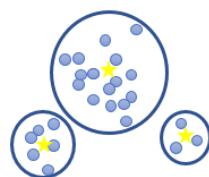
- Given a data distribution  $P_{data}(x)$  (We can sample from it.)
- We have a distribution  $P_G(x; \theta)$  parameterized by  $\theta$ 
  - We want to find  $\theta$  such that  $P_G(x; \theta)$  close to  $P_{data}(x)$
  - E.g.  $P_G(x; \theta)$  is a Gaussian Mixture Model,  $\theta$  are means and variances of the Gaussians

Sample  $\{x^1, x^2, \dots, x^m\}$  from  $P_{data}(x)$

We can compute  $P_G(x^i; \theta)$

Likelihood of generating the samples

$$L = \prod_{i=1}^m P_G(x^i; \theta)$$



Find  $\theta^*$  maximizing the likelihood

简单来说，就是我们有一个生成器 $P_G$ 和一组参数 $\theta$ ，我们还有从真实分布 $P_{data}(x)$ 中采样出的数据 $\{x^1, x^2, \dots, x^m\}$ ，我们不知道数据的真实分布具体长什么样，但是我们希望不断地调整 $P_G$ 和 $\theta$ ，让 $P_G(x; \theta)$ 越接近 $P_{data}(x)$ 越好。具体的做法是，对于每一组参数 $\theta$ 和真实分布的抽样 $x^i$ ，我们能够计算出参数 $\theta$ 下的生成器生成该真实抽样 $x^i$ 的 likelihood，于是我们希望找到一个最佳的参数组 $\theta^*$ ，使得生成器的结果最接近 $P_{data}(x)$ ，也就是对于每个真实抽样 $x^i$ 的 likelihood 都最大，这等价于所有真实抽样 $x^i$ 的 likelihood 的乘积最大，那原始问题就转换为如下这个最大似然问题：

$$L = \prod_{i=1}^m P_G(x^i; \theta)$$

Find  $\theta^*$  maximizing the likelihood

下面我们要求解这个 maximizing the likelihood 问题，我们先证明，它其实等价于求 minimize KL Divergence (KL Divergence 是一个衡量两个分布之间的差异的计算式) 问题。

首先我们加上一个对数 log，将累乘转化为累加问题。然后再将累加转化为期望问题

$$\begin{aligned} \theta^* &= \arg \max_{\theta} \prod_{i=1}^m P_G(x^i; \theta) = \arg \max_{\theta} \log \prod_{i=1}^m P_G(x^i; \theta) \\ &= \arg \max_{\theta} \sum_{i=1}^m \log P_G(x^i; \theta) \quad \{x^1, x^2, \dots, x^m\} \text{ from } P_{data}(x) \\ &\approx \arg \max_{\theta} E_{x \sim P_{data}} [\log P_G(x; \theta)] \end{aligned}$$

然后这个期望也就是积分问题：

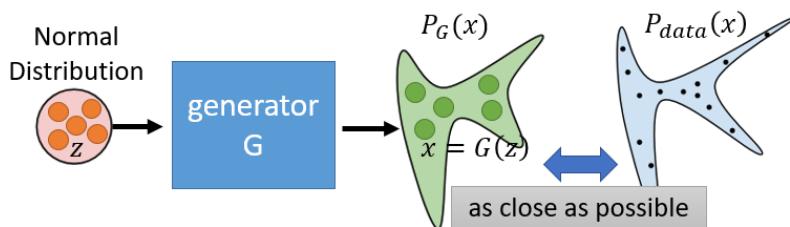
$$\begin{aligned} &\approx \arg \max_{\theta} E_{x \sim P_{data}} [\log P_G(x; \theta)] \\ &= \arg \max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) dx \end{aligned}$$

因为是 arg max，后面可以再补上一项：

$$\begin{aligned} &\approx \arg \max_{\theta} E_{x \sim P_{data}} [\log P_G(x; \theta)] \\ &= \arg \max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) dx - \int_x P_{data}(x) \log P_{data}(x) dx \\ &= \arg \min_{\theta} KL(P_{data} || P_G) \end{aligned}$$

补完之后的式子便刚好等价于 $P_{data}$ 与 $P_G$ 的 KL Divergence 的最小化求解。

现在这个 KL Divergence 的最小化问题如何求解呢？我们不介绍传统的数学方法求解了，我们下面考虑引用神经网络。

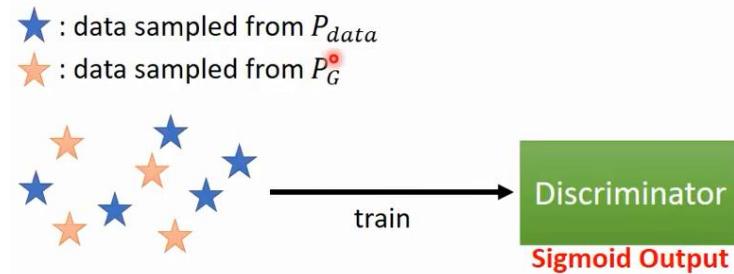


$$G^* = \arg \min_G \underline{Div(P_G, P_{data})}$$

Divergence between distributions  $P_G$  and  $P_{data}$   
How to compute the divergence?

我们把这个神经网络称作生成网络。通过上述的分析我们知道，我们需要训练出这样的生成器，对于一个已知分布的数据  $z$ ，它可以把数据  $z$  转化成一个未知分布的数据  $x$ ，这个未知分布可以与  $z$  所在的分布完全不一样，我们把它称作  $P_G(x)$ ，并且我们希望  $P_G(x)$  与  $P_{data}(x)$  之间的散度距离 (Divergence, 下称 Div) 越小越好。如果能找到这样的  $P_G$ ，那也就意味着我们找到了真实数据分布  $P_{data}(x)$  的近似解，也就意味着我们能够生成各种各样符合真实分布规律的数据。

现在一个最关键的问题是，这个 Div 要如何计算出来呢？理论上来说我们不知道  $P_G(x)$  是什么，我们也不知道  $P_{data}(x)$  是什么，因此 Div 是我们无法计算的。但是，人无法计算的东西，交给神经网络或许就知道如何计算了。于是，我们新建了一个神经网络，专门来量  $P_G(x)$  与  $P_{data}(x)$  之间的 Div，这个神经网络，就叫做判别器。



如上图所示，蓝色星星是从  $P_{data}$  中采样出的数据，黄色星星是从  $P_G$  中采样出的数据，现在交给判别器去判断读入数据是来自  $P_G$  还是  $P_{data}$ ，实际上就是在衡量  $P_G$  与  $P_{data}$  之间的 Div，因为如果二者之间的 Div 越大，判别器就会给  $P_G$  的数据更低的分数，给  $P_{data}$  的数据更高的分数；而如果二者之间的 Div 越小，判别器就会给二者的分数越接近；当  $P_G$  与  $P_{data}$  完全一致时，也就是  $Div=0$  时，判别器给二者的分数就都是 0.5 了。

当然，上述只是我们直观上觉得说，判别器是与 Div 有关的，下面我们需要用数学方法证明：判别器真的可以衡量 Div 的值。我们先来看一下判别器的目标式：

**Example Objective Function for D**

$$V(G, D) = E_{x \sim P_{data}}[\log D(x)] + E_{x \sim P_G}[\log(1 - D(x))]$$

↑      ↓  
 (G is fixed)

**Training:**  $D^* = \arg \max_D V(D, G)$

这个式子很好理解，如果来源  $x \sim P_{data}$ ,  $D(x)$  尽可能高；如果来源  $x \sim P_G$ ,  $D(x)$  尽可能低。下面我们求解一下这个目标式。

首先将目标式转化为一个积分：

- Given  $G$ , what is the optimal  $D^*$  maximizing

$$\begin{aligned}
 V &= E_{x \sim P_{data}}[\log D(x)] + E_{x \sim P_G}[\log(1 - D(x))] \\
 &= \int_x P_{data}(x) \log D(x) dx + \int_x P_G(x) \log(1 - D(x)) dx \\
 &= \int_x [P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))] dx
 \end{aligned}$$

我们假设  $D(x)$  可以是任意函数。那么现在这个表达式，对于所有的  $x$ ，计算出一个表达式  $P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))$ ，使得所有表达式的积分求和最大，这等价于，如

果对于每一个表达式  $P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))$ , 我们能找到一个  $D$  的取值  $D^*$ , 使得这个表达式的最大, 那么最终所有表达式的积分求和也最大, 即:

- Given  $x$ , the optimal  $D^*$  maximizing

$$P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))$$

这个方程的求解非常容易, 最后的结果是:

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

现在把  $D^*(x)$  代入到目标表达式中得到:

$$\begin{aligned} \max_D V(G, D) &= V(G, D^*) \\ &= E_{x \sim P_{data}} \left[ \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \right] \\ &\quad + E_{x \sim P_G} \left[ \log \frac{P_G(x)}{P_{data}(x) + P_G(x)} \right] \\ &= \int_x P_{data}(x) \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} dx \\ &\quad + \int_x P_G(x) \log \frac{P_G(x)}{P_{data}(x) + P_G(x)} dx \end{aligned}$$

进一步变形 (分子分母同除以 2), 转化为:

$$\begin{aligned} \max_D V(G, D) &= V(G, D^*) \\ &= -2\log 2 + \int_x P_{data}(x) \log \frac{P_{data}(x)}{(P_{data}(x) + P_G(x))/2} dx \\ &\quad + \int_x P_G(x) \log \frac{P_G(x)}{(P_{data}(x) + P_G(x))/2} dx \end{aligned}$$

这个表达式等价为:

$$\begin{aligned} &= -2\log 2 + \text{KL}\left(P_{data} \parallel \frac{P_{data} + P_G}{2}\right) + \text{KL}\left(P_G \parallel \frac{P_{data} + P_G}{2}\right) \\ &= -2\log 2 + 2JSD(P_{data} \parallel P_G) \quad \text{Jensen-Shannon divergence} \end{aligned}$$

至此, 我们证明了, 最大化  $V(G, D)$  问题的求解实际上就是在求解  $P_{data}$  与  $P_G$  之间 JS Div 的值 (与前面提到的 KL Div 可以认为是等效的)。

于是, 我们可以再回到生成器要解决的问题上:

$$G^* = \arg \min_G \text{Div}(P_G, P_{data})$$

生成器的目的，是让生成数据 $P_G$ 与真实数据 $P_{data}$ 之间的 Div 最小，本来 Div 是没有办法计算的，但是现在有了判别器之后，Div 变得可以计算了，于是生成器的新的目标表达式变为：

$$G^* = \arg \min_G \max_D V(G, D)$$

接下来我们需要求解这个表达式。值得注意的是，在实际的网络训练中，判别器与生成器是交替训练的，并且是先训练判别器再训练生成器，因此，当生成器需要求解上述表达式的时候，判别器是已经训练好的，于是  $\max_D V(G, D)$  这个式子就可以写成  $L(G)$ ，目标表达式就转化成：

$$G^* = \arg \min_G L(G)$$

这是一个最初级的目标表达式，用基本的 gradient descent 就能求解  $G^*$ ：

$$\theta_G \leftarrow \theta_G - \eta \partial L(G) / \partial \theta_G$$

综上，我们求解出了判别器，然后我们也求解出了生成器，下面我们先用一个完整的算法来回顾一下这整套流程，然后我们会反思目前的这套架构是否已经是完美的。

- In each training iteration:

Learning  
D

- Sample m examples  $\{x^1, x^2, \dots, x^m\}$  from data distribution  $P_{data}(x)$
- Sample m noise samples  $\{z^1, z^2, \dots, z^m\}$  from the prior  $P_{prior}(z)$
- Obtaining generated data  $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$ ,  $\tilde{x}^i = G(z^i)$
- Update discriminator parameters  $\theta_d$  to maximize
  - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(x^i) + \frac{1}{m} \sum_{i=1}^m \log (1 - D(\tilde{x}^i))$
  - $\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$
- Sample another m noise samples  $\{z^1, z^2, \dots, z^m\}$  from the prior  $P_{prior}(z)$

Learning  
G

- Update generator parameters  $\theta_g$  to minimize
  - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(\tilde{x}^i) + \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^i)))$
  - $\theta_g \leftarrow \theta_g - \eta \nabla \tilde{V}(\theta_g)$

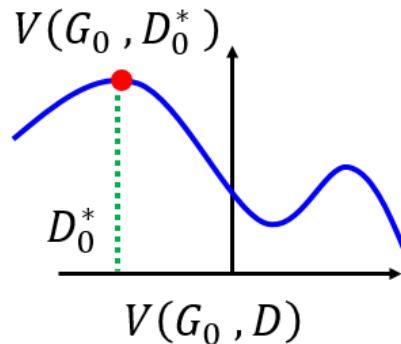
第一个部分是训练判别器，先从真实数据分布 $P_{data}(x)$ 中抽样 $x$ ，然后从先验分布中抽样 $z$ ，并通过生成器产生仿造数据 $\tilde{x}$ ，接着把 $x$ 和 $\tilde{x}$ 丢入判别器中训练，使得目标函数 $\tilde{V}$ 最大；第二个部分是训练生成器，从先验分布中抽样新的 $z$ ，接着把 $z$ 丢入生成器中训练，使得目标函数 $\tilde{V}$ 最小。这样循环交替，最终生成器产生的数据 $\tilde{x}$ 就会越来越接近真实数据 $x$ 。

## Part1 GANs 基于 Div 的改进

### 1. GANs 并不完美

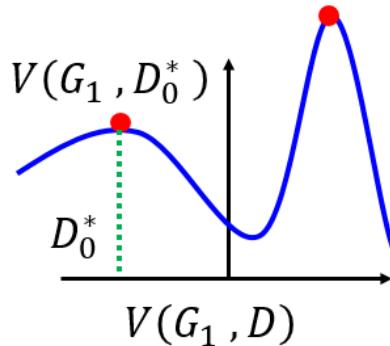
#### 1.1 JS Div 距离偏差问题

我们现在来认真反思，前一篇所介绍的 GANs 理论，它真的能够学习到，或者说它真的能够无限逼近真实数据分布  $P_{data}(x)$  吗？回想一下，在前面的介绍中，我们提到判别器，它的最大化  $\tilde{V}$  的过程实际上就是在求解  $P_{data}$  与  $P_G$  之间的 JS Div 距离，但是到了生成器中，它的最小化  $\tilde{V}$  的过程是在最小化  $P_{data}$  与  $P_G$  之间的 JS Div 距离吗？答案是不一定，我们画张图来理解一下：



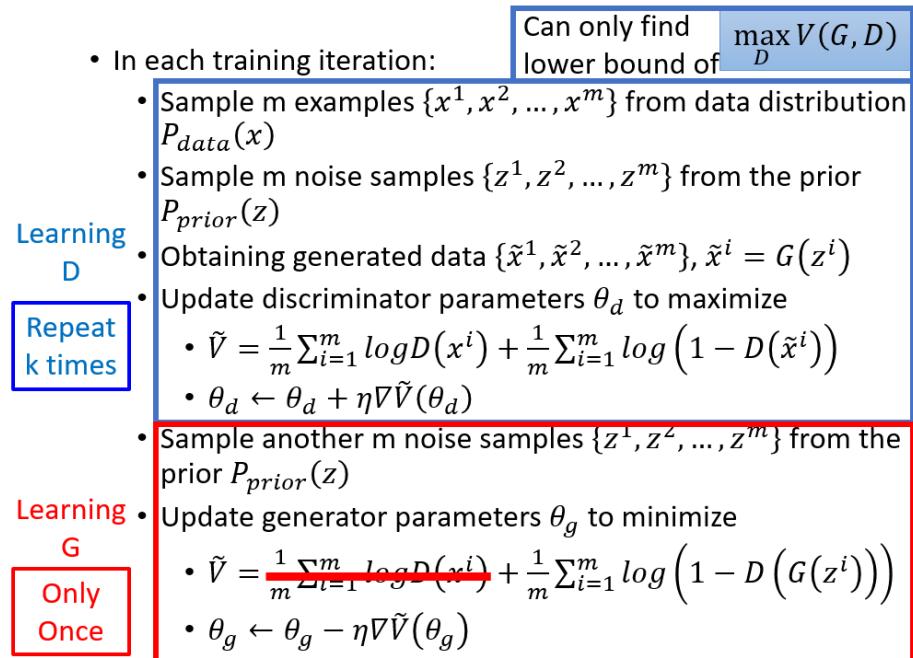
假设上面这张图是判别器训练完成之后，找到了  $D=D_0^*$  使得  $V(G_0, D)$  最大，接下来我们开始训练生成器部分。

假设训练了一次生成器之后， $G_0$  变成  $G_1$ ， $V(G_0, D)$  变成了下图的  $V(G_1, D)$ 。



因为在训练生成器的时候我们是固定  $D$  的，但是我们可以看到此时的  $D_0^*$  不再是让  $V(G_1, D)$  达到最大值的解，也就是说从这时开始，把  $D=D_0^*$  代入  $V(G, D)$  中计算出的值不再等于  $P_{data}$  与  $P_G$  之间的 JS Div 距离（只有 Max  $V(G, D)$  才等于 JS 距离），那么在接下来在继续训练生成器的过程中，minimize 的值就不再会是  $P_{data}$  与  $P_G$  之间的 JS Div 距离了。

这个问题可以通过限制训练次数来解决。对于判别器，理论上我们需要让它训练非常多 次，直到判别器找到的  $D_0^*$  是  $\text{ArgMax } V(G, D)$  的全局最大解，这样  $D_0^*$  在传给生成器时才能保证  $V(G, D)$  是  $P_{data}$  与  $P_G$  之间的 JS Div 距离；而对于生成器，我们限制它只能训练 1 次，这是为了防止训练完一次后  $V(G, D)$  发生变化导致  $D_0^*$  不再是  $\text{ArgMax } V(G, D)$  的解。于是原始的算法优化为如下算法：



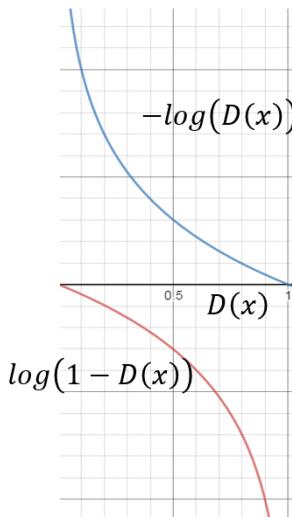
不过，在实际操作中我们往往不会非常多次地训练判别器，因为找到真正的解 $D_0^*$ 需要的训练次数太多，为了减小训练代价我们只会训练 k 次，找到 $D_0^*$ 的近似解 $D_0$ 即可停止。所以在实际的应用中，我们计算的都是 JS Div 的近似值，最终 GANs 学到的是近似分布而不是数据的真实分布，这是我们需要注意的地方。

## 1.2 训练速度问题

原始 GAN 中，判别器对于伪造数据的处理式是：

$$E_{x \sim P_G} [\log(1 - D(x))]$$

我们注意到， $\log(1 - D(x))$ 这个表达式，一开始减得很慢，后面减得很快，我们不妨与 $-\log(D(x))$ 这个表达式对比一下：



我们知道,一开始判别器是很容易鉴别伪造数据的,因此  $D(x)$  的初始值是在靠近 0 的左端。而对于刚开始训练的模型,我们希望在初期  $D(x)$  能够快速地更新,但不幸的是,目标函数  $\log(1 - D(x))$  左端刚好是平缓的区域,依据梯度下降原理这会阻碍  $D(x)$  的快速更新。

为了解决这一问题,有人提出了把  $\log(1 - D(x))$  这个表达式换成  $-\log(D(x))$ ,同样能满足判别器的目标函数要求,并且在训练初期还能更新得比较快。

上述方法便是在这个非常小的地方做了改进。不过后来,人们为了区分这两种 GAN,还是分别起了不同的名字。第一种 GAN 被叫做 MMGAN (Minimax GAN),它也是人们常说的原始 GANs;第二种 GAN 被叫做 NSGAN (Non-saturating GAN)。

前面两节是在一些细节上对 GANs 做了改进,那其实还有很多专门的 paper 对 GANs 做了架构上的改进,我会在第 3 节进行介绍,不过在那之前我会先介绍一篇非常有趣的 paper,叫做 fGAN,它能帮助我们对 GANs 有更深的理解。

## 2. fGAN——深度理解 GAN 理论

fGAN 其实想要表达的就是一件事，不只是 JS Div，任何的 Div（统称为 f-Div）都可以被放到 GANs 的架构中去。

我们先来看一下 fGAN 的证明（如果不感兴趣可以直接跳到△处）。

设定 P 和 Q 是两个不同的分布， $p(x)$  和  $q(x)$  代表着分别从 P 和 Q 采样出 x 的几率，则我们将 f-Div 定义为：

$$D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \quad (1)$$

上述公式衡量 P 和 Q 有多不一样，公式里边的函数 f 可以是很多不同的版本，只要 f 满足以下条件：它是一个凸函数同时  $f(1)=0$ 。

稍微分析一下这个公式：

- 假设对于所有的 x 来说，都有  $p(x)=q(x)$ ，则有  $D_f(P, Q)=0$ ，也就是意味着两个分布没有区别，和假设一样。

- 同时 0 是  $D_f$  能取到的最小值：

$$D_f(p||q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \geq f\left(\int_x q(x)\frac{p(x)}{q(x)}dx\right) = f(1) = 0 \quad (2)$$

也就是说，只要两个分布稍有不同，就能通过  $D_f$  得到的正值反映出来。这个时候我们发现之前常用的 KL Div 其实就是 F Div 的一种。

当你设置  $f(x)=x \log x$ ，即将 F Div 转换为了 KL Div 了。

$$D_f(P||Q) = \int_x q(x)\frac{p(x)}{q(x)}\log\left(\frac{p(x)}{q(x)}\right)dx = \int_x p(x)\log\left(\frac{p(x)}{q(x)}\right)dx \quad (3)$$

当你设置  $f(x)=-\log x$ ，即将 F Div 转换为了 Reverse KL Div。

$$D_f(P||Q) = \int_x q(x)(-\log\left(\frac{p(x)}{q(x)}\right))dx = \int_x q(x)\log\left(\frac{q(x)}{p(x)}\right)dx \quad (4)$$

当你设置  $f(x)=(x-1)^2$ ，即将 F Div 转换为了 Chi Square。

$$D_f(P||Q) = \int_x q(x)\left(\frac{p(x)}{q(x)}-1\right)^2dx = \int_x \frac{(p(x)-q(x))^2}{q(x)}dx \quad (5)$$

Fenchel 共轭(Fenchel Conjugate)

每一个凸函数  $f(x)$  都有一个共轭函数取作  $f^*(x)$ ：

$$f^*(x) = \max_{x \in \text{dom}(f)} xt - f(x) \quad (6)$$

上述公式的意思就是给定 t 找出一个在  $f(x)$  里边有定义的 x 使得  $xt-f(x)$  最大，当然 t 可以无限取值，那么假定我们取值  $t=t_1$  和  $t=t_2$  则有：

$$f^*(t) = \max_{x \in \text{dom}(f)} \{xt - f(x)\}$$

$$f^*(\mathbf{t}_1) = \max_{x \in \text{dom}(f)} \{x\mathbf{t}_1 - f(x)\}$$

$$x_1 \mathbf{t}_1 - f(x_1) \quad f^*(\mathbf{t}_1) \quad f^*(\mathbf{t}_2) = \max_{x \in \text{dom}(f)} \{x\mathbf{t}_2 - f(x)\}$$

$$x_2 \mathbf{t}_1 - f(x_2) \quad f^*(\mathbf{t}_2)$$

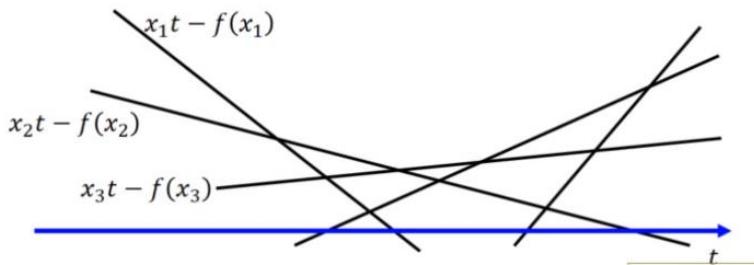
$$x_3 \mathbf{t}_2 - f(x_3) \quad f^*(\mathbf{t}_2)$$

$$x_2 \mathbf{t}_2 - f(x_2)$$

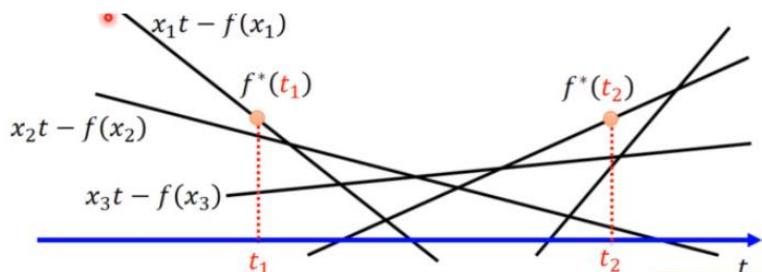
$$x_1 \mathbf{t}_2 - f(x_1)$$



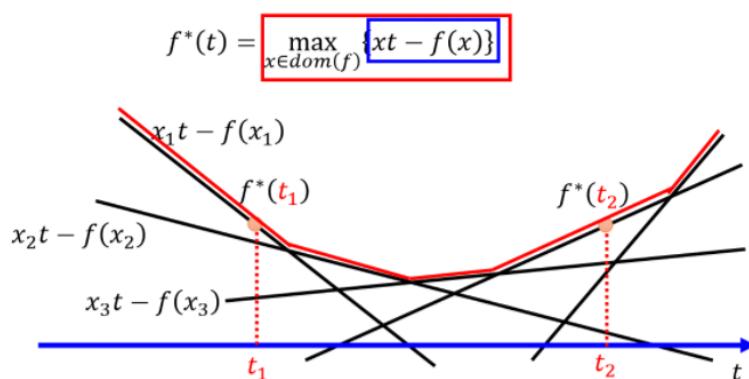
对于所有可能的变量  $t$ ,  $xt-f(x)$  对应了无数条直线:



这个时候给定某个  $t$  看看哪个  $x$  可以取得最大值:

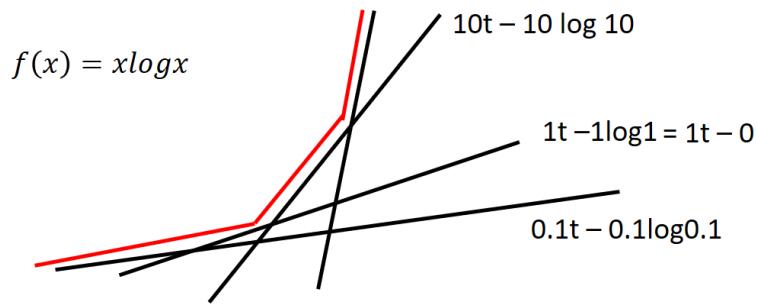


如上图, 当  $t=t1$  的时候, 找到最大点  $f^*(t1)$ , 当  $t=t2$  的时候, 找到最大点  $f^*(t2)$ , 遍历所有的  $t$  即可得到红色的这条函数就是  $f^*(t)$ :



下面我们看一个具体一些的例子。

当  $f(x)=x \log x$  时, 我们可以将对应的  $f^*(t)$  画出来:



这个图实际上是一个指数函数, 当  $f(x)=x \log x$  时,  $f^*(t)=e^{t-1}$ 。

由于  $f^*(t) = \max_{x \in \text{dom}(f)} xt - f(x)$ , 假设让  $g(x)=xt-x \log x$ , 那么现在的问题就变成了: 给定一个  $t$  时, 求  $g(x)$  的最大值问题。对  $g(x)$  求导并让导数为 0:  $dg(x)/dx=t-\log x-1=0$ , 可解得  $x=e^{t-1}$ 。再带入回原公式可得:  $f^*(t)=e^{t-1} \times t - e^{t-1} \times (t-1)=e^{t-1}$ 。

### f-Div GAN

那我们怎么用上边的数学知识和 GAN 联系在一起呢? 我们首先要记得这条公式, 关于  $f^*(t)$  和  $f(x)$  的转换关系式:

$$f^*(t) = \sup_{x \in \text{dom}(f)} \{xt - f(x)\} \Leftrightarrow f(x) = \max_{t \in \text{dom}(f^*)} \{xt - f^*(t)\} \quad (7)$$

利用这个关系, 我们能够将 F Div 的定义变形为一个类似于 GAN 的式子。

$$D_f(P||Q) = \int_x q(x) f\left(\frac{p(x)}{q(x)}\right) dx \quad (8)$$

$$= \int_x q(x) \left( \max_{t \in \text{dom}(f^*)} \left\{ \frac{p(x)}{q(x)} t - f^*(t) \right\} \right) dx \quad (9)$$

$$\geq \int_x q(x) \left( \frac{p(x)}{q(x)} D(x) - f^*(D(x)) \right) dx \quad (10)$$

$$= \int_x p(x) D(x) dx - \int_x q(x) f^*(D(x)) dx \quad (11)$$

$$\approx \max_D \int_x p(x) D(x) dx - \int_x q(x) f^*(D(x)) dx \quad (12)$$

解释一下上式:

- 第一行就是 F Div 的定义式;
- 第三行将  $t$  替换成  $D(x)$  并将  $=$  替换成  $\geq$  原因是: 我们要求得的是给定  $x$  找到一个  $t$  使得式子最大, 也就是说不管  $D(x)$  取什么值都一定小于或者等于第二行的式子;

• 最后一步就是, 我要找到一个  $D$  使得, 式子最大, 上界就是等于第二行的式子。

现在我们推导出关于  $f$  Div 的变式:

$$D_f(P||Q) \approx \max_D \int_x p(x)D(x)dx - \int_x q(x)f^*(D(x))dx \quad (13)$$

$$= \max_D \{E_{x \sim P}[D(x)] - E_{x \sim Q}[f^*(D(x))]\} \quad (14)$$

我们知道, GAN 的目的是训练生成器  $G$ , 使其产生的数据分布  $P_G$  与真实数据的分布  $P_{data}$  尽可能小。换言之, 如果我们用  $f$ -Div 来表达  $P_G$  与  $P_{data}$  的差异, 则希望最小化  $D_f(P_{data}||P_G)$ 。

$$D_f(P_{data}||P_G) = \max_D \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[f^*(D(x))]\} \quad (15)$$

对于生成器来说, 我们就是要找到一个  $P_G$  使得有:

$$G^* = \arg \min_G D_f(P_{data}||P_G) \quad (16)$$

$$= \arg \min_G \max_D \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[f^*(D(x))]\} \quad (17)$$

$$= \arg \min_G \max_D V(G, D) \quad (18)$$

上述从数学推导上给出了  $V(G,D)$  的定义方式。但实际上要注意, 此处的  $V(G,D)$  不一定就是原生 GAN 的形式。  $f$ -Div GAN 是对 GAN 模型的统一, 对任意满足条件的  $f$  都可以构造一个对应的 GAN。

综上便是 fGAN 的完整证明过程, 原论文中还给出了各种不同的 Div function, 想要使用不同的 Div 距离直接选择其对应的函数即可:

Name	$D_f(P  Q)$	Generator $f(u)$
Total variation	$\frac{1}{2} \int  p(x) - q(x)  dx$	$\frac{1}{2} u - 1 $
Kullback-Leibler	$\int p(x) \log \frac{p(x)}{q(x)} dx$	$u \log u$
Reverse Kullback-Leibler	$\int q(x) \log \frac{q(x)}{p(x)} dx$	$-\log u$
Pearson $\chi^2$	$\int \frac{(q(x)-p(x))^2}{p(x)} dx$	$(u - 1)^2$
Neyman $\chi^2$	$\int \frac{(p(x)-q(x))^2}{q(x)} dx$	$\frac{(1-u)^2}{u}$
Squared Hellinger	$\int (\sqrt{p(x)} - \sqrt{q(x)})^2 dx$	$(\sqrt{u} - 1)^2$
Jeffrey	$\int (p(x) - q(x)) \log \left( \frac{p(x)}{q(x)} \right) dx$	$(u - 1) \log u$
Jensen-Shannon	$\frac{1}{2} \int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} dx$	$-(u + 1) \log \frac{1+u}{2} + u \log u$
Jensen-Shannon-weighted GAN	$\int p(x) \pi \log \frac{p(x)}{\pi p(x)+(1-\pi)q(x)} + (1-\pi)q(x) \log \frac{q(x)}{\pi p(x)+(1-\pi)q(x)} dx$ $\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} dx - \log(4)$	$\pi u \log u - (1 - \pi + \pi u) \log(1 - \pi + \pi u)$ $u \log u - (u + 1) \log(u + 1)$

Name	Conjugate $f^*(t)$
Total variation	$t$
Kullback-Leibler (KL)	$\exp(t - 1)$
Reverse KL	$-1 - \log(-t)$
Pearson $\chi^2$	$\frac{1}{4}t^2 + t$
Neyman $\chi^2$	$2 - 2\sqrt{1-t}$
Squared Hellinger	$\frac{t}{1-t}$
Jeffrey	$W(e^{1-t}) + \frac{1}{W(e^{1-t})} + t - 2$
Jensen-Shannon	$-\log(2 - \exp(t))$
Jensen-Shannon-weighted GAN	$(1 - \pi) \log \frac{1 - \pi}{1 - \pi e^{t/\pi}}$ $-\log(1 - \exp(t))$

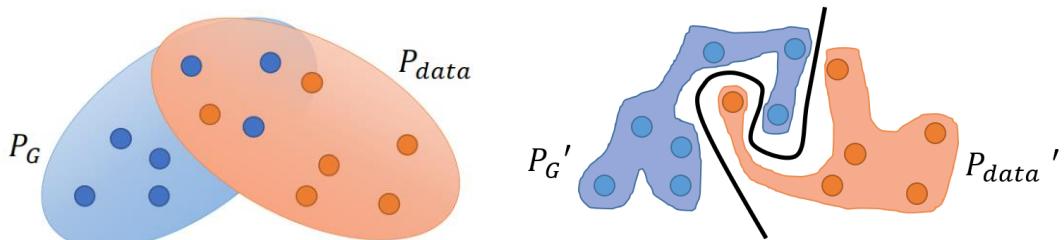
△: 如果上面的证明没有看懂完全没有关系, 我们只需要记住, f-GAN 告诉我们的就是一句话: 不只是 JS Div, 任何的 Div (统称为 f-Div) 都可以被放到 GANs 的架构中去。

### 3. JS Div 不是最佳的 Div

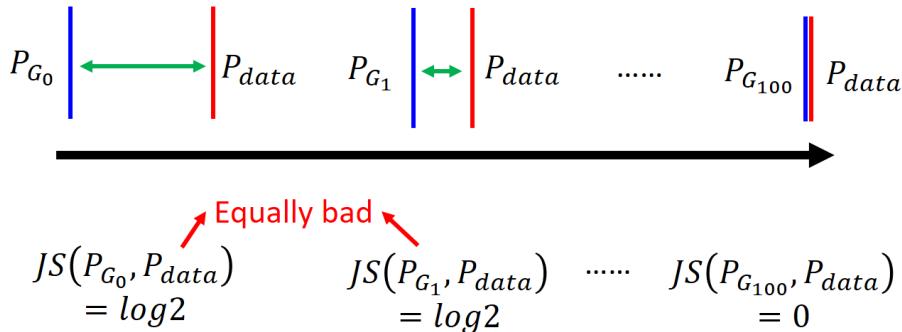
在第三篇当中我们介绍了 fGAN 告诉我们的结论：不只是 JS Div，任何的 Div（统称为 f-Div）都可以被放到 GANs 的架构中去。

现在有了这个结论，我们开始思考，原始 GANs 的 JS Div 到底是不是最好的 Div。也就是说，我们得去发掘，JS Div 是否存在一些毛病。

我们先考虑一下  $P_G$  与  $P_{data}$  的实际分布情况，我们会发现，大多数情况下  $P_G$  与  $P_{data}$  是没有重合的。因为一方面，从理论上来说， $P_G$  与  $P_{data}$  都属于高维空间中的低维流形，二者具有重合的可能性是非常低的；另外一方面，从实际上来看，即算  $P_G$  与  $P_{data}$  的分布有了重合区域（如下左图），但是在实际训练中我们是从  $P_G$  与  $P_{data}$  中取的采样，这些采样形成的分布很有可能是互不相交的（如下右图），我们仍然能找到一条分割线将  $P_G$  与  $P_{data}$  完美分割开来（如右图中的黑线）。所以我们可以认为，大多数情况下  $P_G$  与  $P_{data}$  是没有重合的。



那如果  $P_G$  与  $P_{data}$  是没有重合的，然后用 JS Div 去衡量  $P_G$  与  $P_{data}$  的距离的话，就会造成如下障碍：



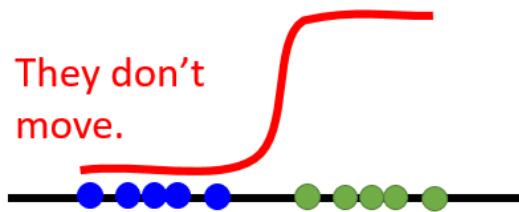
在上图中可以看出， $P_{G_0}$  与  $P_{G_1}$  都与  $P_{data}$  没有交集，但是  $P_{G_1}$  与  $P_{data}$  的距离比  $P_{G_0}$  与  $P_{data}$  的距离近，然而用 JS Div 去衡量二者的距离却是一样的，都为  $\log 2$ ，这是我们觉得 JS Div 不合理的地方，因为实际情况是  $\text{Div}(P_{G_1}, P_{data})$  应当比  $\text{Div}(P_{G_0}, P_{data})$  要小，才能反映出  $P_{G_1}$  与  $P_{data}$  比  $P_{G_0}$  与  $P_{data}$  要靠的更近。有必要说明一下，为什么如果两个分布完全没有重合的话，那么这两个分布的 JS Div 会是一样的。前面有提到，JS Div 是通过判别器计算出来的，而判别器的本质是二分类器，只要  $P_G$  与  $P_{data}$  完全没有重合，判别器就能 100% 地鉴别出  $P_G(x)$  与  $P_{data}(x)$  的差异，因此二者的 JS Div 就是一样的。

因此，我们发现 JS Div 是存在问题的。为了解决这些问题，我们开始介绍下面几篇 paper，它们从不同角度采用了不同的方法，实现 GANs 的改进与提升。

## 4. LSGAN

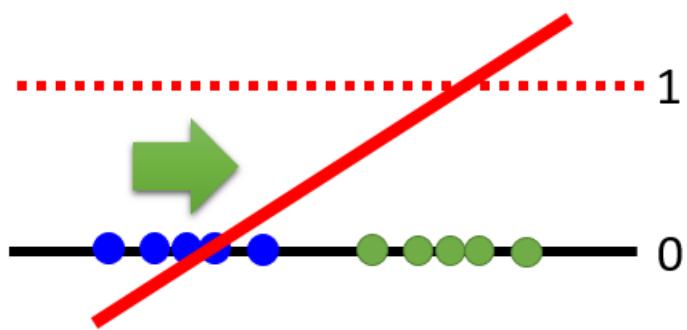
在上一节当中提到, JS Div 出现问题的原因是: 只要  $P_G$  与  $P_{data}$  完全没有重合, 判别器就能 100% 地鉴别出  $P_G(x)$  与  $P_{data}(x)$  的差异, 因此二者的 JS Div 就是一样的。那现在的一个解决思路就是, 让判别器始终都不能 100% 地鉴别出  $P_G(x)$  与  $P_{data}(x)$  的差异, 这样即便  $P_G$  与  $P_{data}$  完全没有重合, 二者的 JS Div 也会不一样, 而只要 Div 存在差异, 就能反映出  $P_G$  的优劣度来。基于这样的思路, LSGAN (最小二乘 GAN) 被提出了。

当然, 上面的描述还是有些拗口, 下面我们通过图片来详细说明一下。



第一张图是判别器训练得太好的例子, 它能够 100% 地鉴别出  $P_G(x)$  与  $P_{data}(x)$  的差异。蓝色样本点, 是生成样本  $P_G$ , 它们的得分为 0; 绿色样本点, 是真实样本  $P_{data}$ , 它们的得分为 1;  $P_G$  与  $P_{data}$  之间完全没有交集。这样会出现什么问题呢? 当轮到生成器训练的时候, 它希望蓝色的点能够向右移, 但是因为对于所有蓝色点, 判别器计算出的 JS Div 都是一样的, 这意味着所有点的梯度都是 0, 于是基于 gradient descent 所有的生成样本的点都无法移动了。这种情况其实是很有趣的, 因为在之前的理论中, 我们希望判别器尽可能训练到最好, 但是当  $P_G$  与  $P_{data}$  之间完全没有交集时, 判别器就不能训练得太好, 因为那意味着梯度消失至 0, 生成器无法更新。

接下来, 我们就要想办法去限制判别器不要训练得太好。其实方法非常简单, 只需要将判别器的最后的 sigmoid 激活层改成 linear 激活层, 这样训练出的 D 就会是一个线性的直线, 如下图所示。



D 只有一种情况下才会梯度为 0, 就是  $P_G$  与  $P_{data}$  完全重合时, D 变为一条  $y=1/2$  处的水平直线; 其他情况下  $P_G$  都会顺着直线的梯度向  $P_{data}$  靠拢。这条直线是如何计算出的呢, 它用到的方法是最小二乘法, 详细理论可以参阅 LSGAN 的 paper。

上述便是 LSGAN, 它解决的是 GANs 当中梯度消失为 0 的问题; 其实关于梯度消失问题, 还有一篇 paper 从判别器的角度给出了方案, 详见 Part3 的 RGAN, 就不放在这儿了。

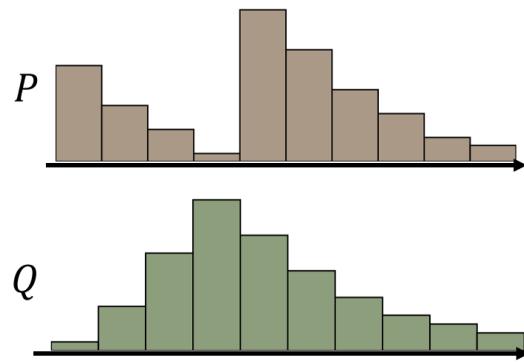
可是, 我们真正想要解决的问题, 即如何去更好地测量  $P_G$  与  $P_{data}$  之间的 Div, LSGAN 绕过了这一核心问题, 真正在这一问题上作出了突破的是下面这篇 paper——WGAN。

## 5. WGAN

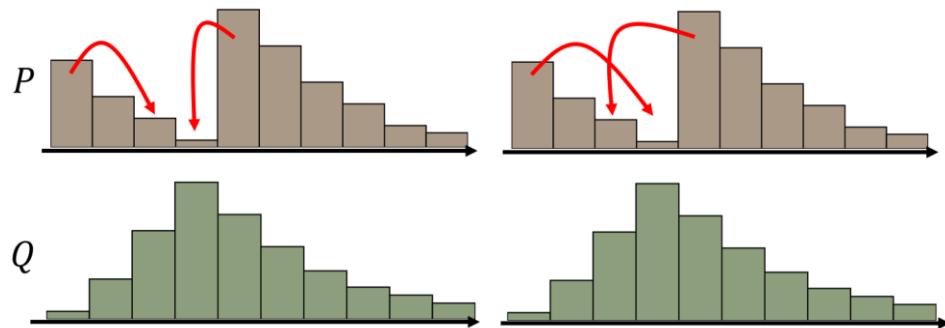
WGAN 的全称是 WassersteinGAN，它提出了用 Wasserstein 距离（也称 EM 距离）去取代 JS 距离，这样能更好的衡量两个分布之间的 Div。我们先介绍一下什么是 EM 距离。

### 5.1 EM 距离

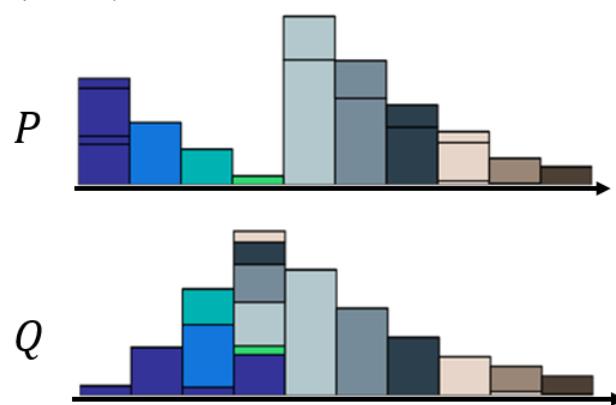
EM 距离的全称是 EarthMover (推土距离)，它的定义非常直观：假设有两堆数据分布  $P$  和  $Q$ ，看作两堆土，现在把  $P$  这堆土推成  $Q$  这堆土所需要的最少的距离就是 EM 距离。



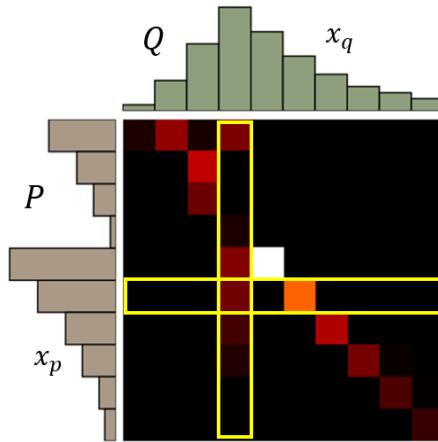
假设  $P$  的分布是上图棕色柱块区域， $Q$  的分布是上图绿色柱块区域，现在需要把  $P$  的分布推成  $Q$  的分布，我们可以制定出很多不同的 MovingPlan (推土计划)。



这些不同的推土计划都能把分布  $P$  变成分布  $Q$ ，但是它们所要走的平均推土距离是不一样的，我们最终选取最小的平均推土距离值作为 EM 距离。例如上面这个例子的 EM 距离就是下面这个推土方案对应的值。



那为了更好地表示这个推土问题, 我们可以把每一个 moving plan 转化为一个矩阵图:



每一个色块表示  $P$  分布到  $Q$  分布需要分配的土量 (移动距离), 那每一行的色块之和就是  $P$  分布该行位置的柱高度, 每一列的色块之和就是  $Q$  分布该列位置的柱高度。于是我们的求解目标表达式就如下所示:

Average distance of a plan  $\gamma$ :

$$B(\gamma) = \sum_{x_p, x_q} \gamma(x_p, x_q) \|x_p - x_q\|$$

Earth Mover's Distance:

$$W(P, Q) = \min_{\gamma \in \Pi} B(\gamma)$$

The best plan

表达式中  $\gamma$  函数计算当前计划下  $x_p$  到  $x_q$  的推土量,  $\|x_p - x_q\|$  表示二者间的推土距离。

那如果这个时候我们想直接求解这个表达式的话, 是非常麻烦的, 因为需要穷举所有的 moving plan 然后再选择其最小值。如果我们对之前的理论有印象的话, 我们会想到这个优化问题依然可以交给判别器来解决。

于是接下来要做的, 就是去改判别器, 让它能够衡量  $P_G$  与  $P_{data}$  之间的 wasserstein 距离。

## 5.2 WGAN

下面我们直接给出 WGAN 的判别器的目标表达式:

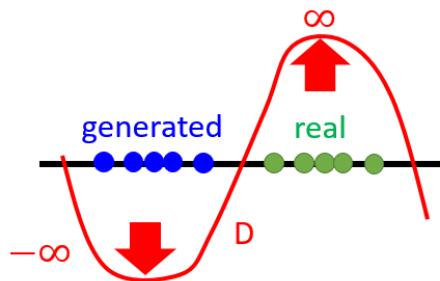
$$V(G, D) = \max_{D \in 1-Lipschitz} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

这个表达式的求解结果就是  $P_G$  与  $P_{data}$  之间的 wasserstein 距离。至于为什么会等于 Was 距离, 详细证明请参阅 WGAN paper 附录当中的证明部分, 因为过于繁琐, 在此就不赘述 (逃)。反正目前我们构造出了求解 was 距离的判别器。

关于这个表达式, 值得注意的是,  $D$  被加上了 1-Lipschitz function (如下图) 的限制。

$$\begin{aligned} \|f(x_1) - f(x_2)\| &\leq K \|x_1 - x_2\| \\ K=1 \text{ for "1 - Lipschitz"} \end{aligned}$$

先说明一下，为什么要对判别器做限制。传统 GANs 的判别器输出的结果是在(0,1)区间之内，但是在 WGAN 中输出的结果是 was 距离，was 距离是没有上下界的，这意味着，随着训练进行， $P_G$  的 was 值会越来越小， $P_{data}$  的 was 值会越来越大，判别器将永远无法收敛。



因此，为了解决这个问题，我们需要给判别器加上一些限制，让 $P_G$ 不会持续地一直降低，让 $P_{data}$ 也不会持续地一直升高，简言之，就是让 D 函数变得更平滑一些。

但是我们知道，一般的神经网络的训练，参数都是没有限制的，而现在我们希望给判别器的参数增加一些限制，其实是不太好做的。

在最原始的 WGAN 中，采用的做法是 weight clipping，很简单，设定一个上限  $c$  与下限  $-c$ ，如果更新参数  $w$  大于  $c$ ，改成  $w=c$ ；如果更新参数  $w$  小于  $-c$ ，改成  $w=-c$ 。这样 D 在  $P_g$  与  $P_{data}$  处的值就不会被无限拉远。但是这个方法并没有让 D 真的限制在 1-Lipschitz function 内，所以原始的 WGAN 并没有严格地给出 was 距离计算方法。

直到后来，直到 WGAN 的增强版 WGAN-GP，以及 SNGAN 被提出，才解决了这个问题。

## 6. WGAN-GP

### 6.1 WGAN-GP 原理

WGAN 待解决的问题是，未能将 D 真的限制在 1-Lipschitz function 内。我们不妨观察一下 1-Lipschitz function，会发现它其实等价于如下表达式：

$$D \in 1 - Lipschitz \iff \|\nabla_x D(x)\| \leq 1 \text{ for all } x$$

也就是说，对于一个可微函数，当且仅当对于任意的  $x$ ，梯度的模都小于或等于 1，则该可微函数是 1-Lipschitz function。

那现在我们对判别器的目标表达式增添一个条件：

$$\begin{aligned} V(G, D) \approx \max_D & \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)] \\ & - \lambda \int_x \max(0, \|\nabla_x D(x)\| - 1) dx\} \end{aligned}$$

增添的条件（第三项），实际上统计的就是所有梯度的模不满足小于或等于 1 的项，给这些项分配一个惩罚参数  $\lambda$ ，计算出惩罚值，并将所有惩罚值累加起来。当这个累加惩罚够大时，它会拖累  $\max_D \{V(D, G)\}$  的取值，最终导致这样的  $D$  不再是最优解。

但是，这个增添的条件由于对所有  $x$  有效，会让惩罚变得非常高，也带来很多不必要的计算，于是我们需要对新增条件做进一步的更改。

事实上我们真正需要考虑的惩罚项，应该是对判别器产生实质影响的区域。考虑到整个 WGAN 的目的是让  $P_G$  渐渐向  $P_{data}$  靠拢，那位于  $P_G$  和  $P_{data}$  之间的区域一定会对判别器产生实质的影响。因此，我们将惩罚项中  $x$  的范围缩小为  $P_{penalty}$ ， $P_{penalty}$  是介于  $P_G$  和  $P_{data}$  之间的区域。目标表达式转化为如下式子：

$$\begin{aligned} V(G, D) \approx \max_D & \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)] \\ & - \lambda E_{x \sim P_{penalty}}[\max(0, \|\nabla_x D(x)\| - 1)]\} \end{aligned}$$

当然，我们只是直观上觉得，将惩罚项  $x$  的区域缩小为  $P_{penalty}$  会对于限制  $D$  为 1-Lipschitz function 有效，那有没有理论证明呢？很遗憾，原论文中是没有的，作者写了这样一段话：“Given that enforcing the Lipschitz constraint everywhere is intractable, enforcing it only along these straight lines seems sufficient and experimentally results in good performance.”也就是说，实验结果证明这样子是好的。

还有一个有意思的一点是，在实验中作者发现  $\|\nabla_x D(x)\|$  越接近 1，训练得越快，效果也越好，于是不妨把表达式直接改成：

$$\begin{aligned} V(G, D) \approx \max_D & \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)] \\ & - \lambda E_{x \sim P_{penalty}}[(\|\nabla_x D(x)\| - 1)^2]\} \end{aligned}$$

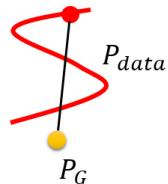
也就是说，在惩罚项中希望  $\|\nabla_x D(x)\|$  越接近 1，惩罚就越少。事实证明这样做的效果是非常好的。

综上 WGAN-GP 的介绍到这就结束了，它的核心突破，在于用 gradient penalty 实现了对判别器的近似 1-Lipschitz 限制，并且在实验中取得了非常不错的效果。

## 6.2 WGAN-GP 的讨论

很明显，WGAN-GP 它依然有值得改进的空间。

举一个例子，如下图：



我们认为，把  $P_G$  和  $P_{data}$  之间的区域看作  $P_{penalty}$  是因为它是会产生影响的，但是在上图中， $P_G$  和  $P_{data}$  连线的区域对于  $P_G$  的移动是没有影响的，因为  $P_G$  真正的走向应该是朝着更近的  $P_{data}$  去变化，而不是朝着上图的这条连线去变成一个更远的  $P_{data}$ 。

也有一些 paper 试图在解决这些问题，譬如 DRAGAN 就认为， $P_{penalty}$  应该还加入  $P_{data}$  区域，等等。

另外，WGAN-GP 还有一个问题，就是它只是对于梯度的模大于 1 的区域的  $x$  作出了惩罚，它并没有保证每一个  $x$  的梯度的模都小于或等于 1，也就是说它并没有从根本上解决判别器的 1-Lipschitz 限制问题。那有没有办法能够保证每个区域上的  $x$  的梯度的模都小于等于 1 呢？有。接下来将在下一章介绍 Spectral Normalization GAN 方法。

## 7. SNGAN

### 7.1 SNGAN 设计思路

现在我们的目的，是要保证对于每一个位置的  $x$ ，梯度的模都小于等于 1。在神经网络中，将梯度的模限制在一个范围内，抽象地来说就是让产生的函数更平滑一些，最常见的做法便是正则化。SNGAN（频谱归一化 GAN）为了让正则化产生更明确地限制，提出了用谱范数标准化神经网络的参数矩阵  $W$ ，从而让神经网络的梯度被限制在一个范围内。

### 7.2 频谱范数

我们先以前馈神经网络为一个简单的例子来解释频谱范数(下称谱范数)的作用。(7.2-7.4 节是一些相关的理论基础，如果不感兴趣可以直接跳到 7.5 节)

一个前馈神经网络可以表示为级联计算:  $x^l = f^l(W^l x^{l-1} + b^l)$ 。其中  $l$  代表层数， $l \in \{1, \dots, L\}$ ； $x^{l-1}$  是第  $l$  层的输入， $x^l$  是第  $l$  层的输出， $f^l$  是一个（非线性的）激活函数， $W^l$  和  $b^l$  分别代表  $l$  层的权重矩阵和偏置向量。现在我们把全体参数的集合记作  $\Theta$ ， $\Theta = \{W^l, b^l\}_{l=1}^L$ ；全体网络层所形成的函数记作  $f_\Theta$ ，即有： $f_\Theta(x^0) = x^L$ 。给定  $K$  组训练数据， $(x_i, y_i)_{i=1}^K$ ，损失函数定义为： $\frac{1}{K} \sum_{i=1}^K L(f_\Theta(x_i), y_i)$ ，通常  $L$  被选择为交叉熵或是  $l_2$  距离，分别用于分类和回归任务。要学习的模型参数是  $\Theta$ 。

现在我们开始考虑如何获得对输入的噪音干扰不敏感的模型。我们的目标是获得一个模型  $\Theta$ ，使得  $f(x + \xi) - f(x)$  的模（指的是 2-范数，即各个元素的平方和）很小，其中  $\xi$  是具有小的模的扰动向量。假设我们选用的激活函数是 ReLU 或 maxout 等分段线性函数，在这种情况下， $f_\Theta$  也是分段线性函数。因此，如果我们考虑  $x$  的小邻域，我们可以将  $f_\Theta$  视为线性函数。换句话说，我们可以用仿射映射表示它， $x \rightarrow W_{\Theta,x}x + b_{\Theta,x}$ ，其中  $W_{\Theta,x}$  是矩阵， $b_{\Theta,x}$  是向量，它们都取决于  $\Theta$  和  $x$  的值。然后，对于小扰动  $\xi$ ，我们有：

$$\frac{\|f_\Theta(x + \xi) - f_\Theta(x)\|_2}{\|\xi\|_2} = \frac{\|(W_{\Theta,x}(x + \xi) + b_{\Theta,x}) - (W_{\Theta,x}x + b_{\Theta,x})\|_2}{\|\xi\|_2} = \frac{\|W_{\Theta,x}\xi\|_2}{\|\xi\|_2} \leq \sigma(W_{\Theta,x})$$

其中  $\sigma(W_{\Theta,x})$  就是  $W_{\Theta,x}$  的谱范数的计算式，数学上它等价于计算矩阵  $W_{\Theta,x}$  的最大奇异值（奇异值的介绍见 7.2 节）。矩阵最大奇异值的表达式参见下式：

$$\sigma(A) = \max_{\xi \in \mathbb{R}^n, \xi \neq 0} \frac{\|A\xi\|_2}{\|\xi\|_2}$$

上述论证表明我们应当训练模型参数  $\Theta$ ，使得对于任何  $x$ ， $W_{\Theta,x}$  的谱范数都很小。为了进一步研究  $W_{\Theta,x}$  的性质，让我们假设每个激活函数  $f^l$  都是 ReLU（该参数可以很容易地推广到其他分段线性函数）。注意，对于给定的向量  $x$ ， $f^l$  充当对角矩阵  $D_{\Theta,x}^l$ ，其中如果  $x^{l-1}$  中的对应元素为正，则对角线中的元素等于 1；否则，它等于零（这是 ReLU 的定义）。于是，我们可以重写  $W_{\Theta,x}$  为下式：

$$W_{\Theta,x} = D_{\Theta,x}^L W^L D_{\Theta,x}^{L-1} W^{L-1} \cdots D_{\Theta,x}^1 W^1$$

又注意到对于每个  $l \in \{1, \dots, L\}$ ，有  $\sigma(D_{\Theta,x}^l) \leq 1$ ，所以我们有：

$$\sigma(W_{\Theta,x}) \leq \sigma(D_{\Theta,x}^L) \sigma(W^L) \sigma(D_{\Theta,x}^{L-1}) \sigma(W^{L-1}) \cdots \sigma(D_{\Theta,x}^1) \sigma(W^1) \leq \prod_{\ell=1}^L \sigma(W^\ell)$$

至此我们得出了一个非常重要的结论, 为了限制  $W_{\Theta,x}$  的谱范数, 只需要每个  $l \in \{1, \dots, L\}$  限制  $W^l$  的谱范数就足够了。这促使我们考虑谱范数正则化, 这将在 7.3 节中描述。

## 7.3\* 奇异值与奇异值分解

在介绍频谱范数正则化之前, 先简要介绍一下后面会用到的技巧: 奇异值分解。奇异值是线性代数中的概念, 奇异值分解是矩阵论中一种重要的矩阵分解法, 奇异值一般通过奇异值分解定理求得。如果读者了解奇异值的话这一节可以跳过。

### 奇异值的定义

设  $A$  为  $m \times n$  矩阵,  $q = \min(m, n)$ ,  $A^*A$  的  $q$  个非负特征值的算术平方根叫作  $A$  的奇异值。

### 奇异值分解定理

设给定  $A \in M_{n,m}$ , 令  $q = \min\{m, n\}$ , 并假设  $\text{rank } A = r$ :

(a) 存在酉矩阵  $V \in M_n$  与  $W \in M_m$ , 以及一个对角方阵

$$\sum_q = \begin{bmatrix} \sigma_1 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \sigma_q \end{bmatrix}$$

使得  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0 = \sigma_{r+1} = \dots = \sigma_q$  以及  $A = V \sum_q W^*$ ,

$$\text{其中 } \sum_q = \begin{cases} \sum_q & \text{if } m = n \\ [\sum_q \ 0] \in M_{n,m} & \text{if } m > n \\ [\sum_q \ 0]^T \in M_{n,m} & \text{if } m < n \end{cases}$$

(b) 参数  $\sigma_1, \sigma_2, \dots, \sigma_r$  是  $AA^*$  的按照递减次序排列的非零特征值的正的平方根, 它们与  $AA^*$  的按照递减次序排列的非零特征值的正的平方根是相同的。

在奇异值分解定理中, 矩阵  $\sum_q$  的对角元素 (即纯量  $\sigma_1, \sigma_2, \dots, \sigma_q$ , 它们是方阵  $\sum_q$  的对角元素) 称为矩阵  $A$  的奇异值。

### \*奇异值分解定理的证明

证明比较复杂, 在此不赘述了, 推荐一篇博文, 感兴趣的读者可以去了解一下:

<https://blog.csdn.net/zhongkejingwang/article/details/43053513>。

但是要注意的是, 7.3 节当中提到了一些概念, 其中左奇异向量指的是  $AA^T$  的特征向量, 右奇异向量指的是  $A^TA$  的特征向量。

## 7.4 频谱范数正则化

频谱范数正则化方法是 17 年 5 月提出来的, 虽然最终的 SNGAN 没有完全采用这一方法, 但是它借鉴了这个方法非常重要的思想。

为了约束每个权重矩阵的频谱范数  $W^l$ , 我们考虑以下经验风险最小化问题:

$$\underset{\Theta}{\text{minimize}} \frac{1}{K} \sum_{i=1}^K L(f_{\Theta}(\mathbf{x}_i), \mathbf{y}_i) + \frac{\lambda}{2} \sum_{\ell=1}^L \sigma(W^{\ell})^2$$

其中  $\lambda \in R^+$  是正则化因子, 第二项被称为谱范数正则项, 它降低了权重矩阵的谱准则。在执行标准梯度下降时, 我们需要计算谱范数正则项的梯度。为此, 让我们考虑对于一

一个特定  $l$  的梯度  $\sigma(W^l)^2/2$ , 其中  $l \in \{1, \dots, L\}$ 。设  $\sigma_1 = \sigma(W^l)$  和  $\sigma_2$  分别是第一和第二奇异值。如果  $\sigma_1 > \sigma_2$ , 则  $\sigma(W^l)^2/2$  的梯度为  $\sigma_1 u_1 v_1^T$ , 其中,  $u_1$  和  $v_1$  分别是第一个左奇异向量和第一个右奇异向量。如果  $\sigma_1 = \sigma_2$ , 则  $\sigma(W^l)^2/2$  是不可微的。然而, 出于实际目的, 我们可以假设这种情况从未发生, 因为实际训练中的数值误差会让  $\sigma_1$  和  $\sigma_2$  不可能完全相等。

由于计算  $\sigma_1$ ,  $u_1$  和  $v_1$  在计算上是昂贵的, 我们使用功率迭代方法来近似它们。从随机初始化的  $v$  开始 (开始于  $l - 1$  层), 我们迭代地执行以下过程足够次数:

$$\mathbf{u} \leftarrow W^\ell \mathbf{v} \text{ and } \mathbf{v} \leftarrow (W^\ell)^\top \mathbf{u}, \text{ and } \sigma \leftarrow \|\mathbf{u}\|_2 / \|\mathbf{v}\|_2$$

最终我们得到了使用频谱范数正则项的 SGD 算法如下:

---

**Algorithm 1** SGD with spectral norm regularization

---

- 1: **for**  $\ell = 1$  to  $L$  **do**
  - 2:      $\mathbf{v}^\ell \leftarrow$  a random Gaussian vector.
  - 3: **for** each iteration of SGD **do**
  - 4:     Consider a minibatch,  $\{(\mathbf{x}_{i_1}, y_{i_1}), \dots, (\mathbf{x}_{i_k}, y_{i_k})\}$ , from training data.
  - 5:     Compute the gradient of  $\frac{1}{k} \sum_{i=1}^k L(f_\Theta(\mathbf{x}_{i_j}), y_{i_j})$  with respect to  $\Theta$ .
  - 6:     **for**  $\ell = 1$  to  $L$  **do**
  - 7:         **for** a sufficient number of times **do**   ▷ One iteration was adequate in our experiments
  - 8:          $\mathbf{u}^\ell \leftarrow W^\ell \mathbf{v}^\ell$ ,  $\mathbf{v}^\ell \leftarrow (W^\ell)^\top \mathbf{u}^\ell$ ,  $\sigma^\ell \leftarrow \|\mathbf{u}^\ell\| / \|\mathbf{v}^\ell\|$
  - 9:         Add  $\lambda \sigma^\ell \mathbf{u}^\ell (\mathbf{v}^\ell)^\top$  to the gradient of  $W^\ell$ .
  - 10:      Update  $\Theta$  using the gradient.
- 

值得注意的是, 为了最大化  $\sigma_1$ ,  $u_1$  和  $v_1$ , 在 SGD 的下一次迭代开始时, 我们可以用  $v_1$  代替第 2 步中的初始向量  $v$ 。然后在第 7 步中右方的标注是, paper 作者在实验中发现, 只进行一次迭代就能够获得足够好的近似值。文章中还提到对于含有卷积的神经网络架构, 我们需要将参数对齐为  $b \times a k_w k_h$  的矩阵, 再去计算该矩阵的谱范数并添加到正则项中。

综上, 频谱范数正则化看起来非常复杂, 但是它的实际做法, 可以简单地理解为, 把传统 GANs 中的 loss 函数:

$$\underset{\Theta}{\text{minimize}} \frac{1}{K} \sum_{i=1}^K L(f_\Theta(\mathbf{x}_i), \mathbf{y}_i) + \lambda \sum (w_i)^2$$

其中的正则项替换成谱范数:

$$\underset{\Theta}{\text{minimize}} \frac{1}{K} \sum_{i=1}^K L(f_\Theta(\mathbf{x}_i), \mathbf{y}_i) + \frac{\lambda}{2} \sum_{\ell=1}^L \sigma(W^\ell)^2$$

并且谱范数的计算利用了功率迭代的方法去近似。

## 7.5 SNGAN 的实现

之前我们说到, 对于 GANs 最重要的目的是实现 D 的 1-lipschitz 限制, 频谱范数正则化固然有效, 但是它不能保证把  $f_\Theta$  的梯度限制在一个确定的范围内, 真正解决了这一问题的, 是直到 18 年 2 月才被提出的 SNGAN。SNGAN 基于 spectral normalization 的思想, 通过对  $W$  矩阵归一化的方式, 真正将  $f_\Theta$  的梯度控制在了小于或等于 1 的范围内。

我们先来证明, 只要将每一层  $W^\ell$  的谱范数都限制为 1, 最终得到的  $f_\Theta$  函数就会满足 1-lipschitz 限制。

对于一个线性层函数  $g(h) = Wh$ , 我们可以计算出它的 lipschitz 范式:

$$\|g\|_{\text{Lip}} = \sup_{\mathbf{h}} \sigma(\nabla g(\mathbf{h})) = \sup_{\mathbf{h}} \sigma(W) = \sigma(W).$$

如果激活层函数的 lipschitz 范式  $\|a_l\|_{lip}=1$ (比如 ReLU), 我们就有如下不等式:

$$\|g_1 \circ g_2\|_{Lip} \leq \|g_1\|_{Lip} \cdot \|g_2\|_{Lip}$$

其中  $\circ$  表示复合函数。我们利用上面的不等式, 就能够得到  $f_\theta$  的 lipschitz 范式的限制式:

$$\begin{aligned}\|f\|_{Lip} &\leq \|(\mathbf{h}_L \mapsto W^{L+1}\mathbf{h}_L)\|_{Lip} \cdot \|a_L\|_{Lip} \cdot \|(\mathbf{h}_{L-1} \mapsto W^L\mathbf{h}_{L-1})\|_{Lip} \\ &\cdots \|a_1\|_{Lip} \cdot \|(\mathbf{h}_0 \mapsto W^1\mathbf{h}_0)\|_{Lip} = \prod_{l=1}^{L+1} \|(\mathbf{h}_{l-1} \mapsto W^l\mathbf{h}_{l-1})\|_{Lip} = \prod_{l=1}^{L+1} \sigma(W^l)\end{aligned}$$

于是现在, 我们只需要保证  $\sigma(W^l)$  恒等于 1, 就能够让  $f_\theta$  函数满足 1-lipschitz 限制。做法非常简单, 只需要将  $W$  矩阵归一化即可:

$$\bar{W}_{SN}(W) := W/\sigma(W)$$

至此, SNGAN 通过将  $W$  矩阵归一为谱范数恒等于 1 的式子, 进而控制  $f_\theta$  的梯度恒小于等于 1, 最终实现了对  $D$  的 1-lipschitz 限制, 最后我们给出 SNGAN 中的梯度下降算法:

---

**Algorithm 1** SGD with spectral normalization

---

- Initialize  $\tilde{\mathbf{u}}_l \in \mathcal{R}^{d_l}$  for  $l = 1, \dots, L$  with a random vector (sampled from isotropic distribution).
  - For each update and each layer  $l$ :
    1. Apply power iteration method to a unnormalized weight  $W^l$ :
 
$$\begin{aligned}\tilde{\mathbf{v}}_l &\leftarrow (W^l)^T \tilde{\mathbf{u}}_l / \| (W^l)^T \tilde{\mathbf{u}}_l \|_2 \\ \tilde{\mathbf{u}}_l &\leftarrow W^l \tilde{\mathbf{v}}_l / \| W^l \tilde{\mathbf{v}}_l \|_2\end{aligned}$$
    2. Calculate  $\bar{W}_{SN}$  with the spectral norm:
 
$$\bar{W}_{SN}^l(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\mathbf{u}}_l^T W^l \tilde{\mathbf{v}}_l$$
    3. Update  $W^l$  with SGD on mini-batch dataset  $\mathcal{D}_M$  with a learning rate  $\alpha$ :
 
$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}_{SN}^l(W^l), \mathcal{D}_M)$$
- 

可以看出, 与传统的 SGD 相比, 带有谱归一化的 SGD 做的额外处理就是对  $W$  矩阵做的归一化处理:

$$\bar{W}_{SN}^l(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\mathbf{u}}_l^T W^l \tilde{\mathbf{v}}_l$$

## Part2 GANs 基于 Network 的改进

在这一部分我们开始探讨生成器与判别器内部网络的结构,之前我们一直在探讨二者在外部的连接方式和如何使用 Div 能让结果更好,而涉及到生成器与判别器本身时一直粗略地描述成神经网络,但其实,使用不同的神经网络的结构对结果会产生不同的影响。本章我们首先介绍基于卷积网络改进的 DCGAN 与 ImprovedDCGAN,然后介绍消除感受野障碍的 SAGAN,最后会介绍迄今为止规模最大、效果最好的 BigGAN。

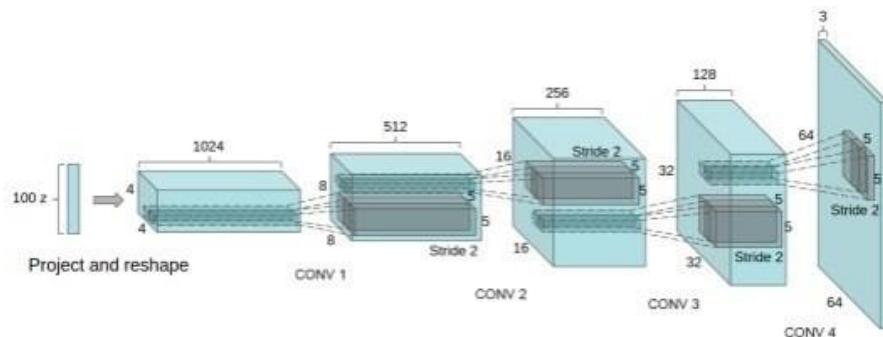
### 1. DCGAN

我们知道深度学习中对图像处理应用最好的模型是 CNN,那么如何把 CNN 与 GAN 结合? DCGAN 是这方面最好的尝试之一。

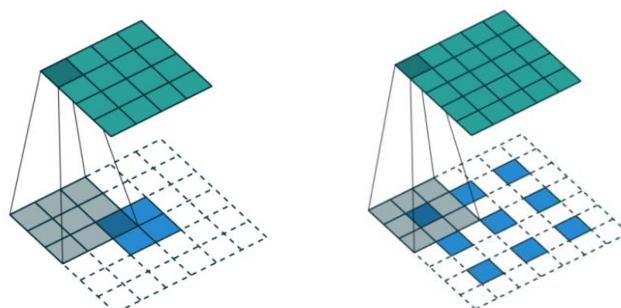
DCGAN 的原理和 GAN 是一样的,这里就不在赘述。它只是把上述的 G 和 D 换成了两个卷积神经网络(CNN)。但不是直接换就可以了,DCGAN 对卷积神经网络的结构做了一些改变,以提高样本的质量和收敛的速度,这些改变有:

- 取消所有 pooling 层。G 网络中使用微步幅度卷积(fractionally strided convolution)代替 pooling 层, D 网络中使用步幅卷积(strided convolution)代替 pooling 层。
- 在 D 和 G 中均使用 batch normalization
- 去掉 FC 层,使网络变为全卷积网络
- G 网络中使用 ReLU 作为激活函数,最后一层使用 tanh
- D 网络中使用 LeakyReLU 作为激活函数

我们来看一下 DCGAN 中 G 的具体网络结构:

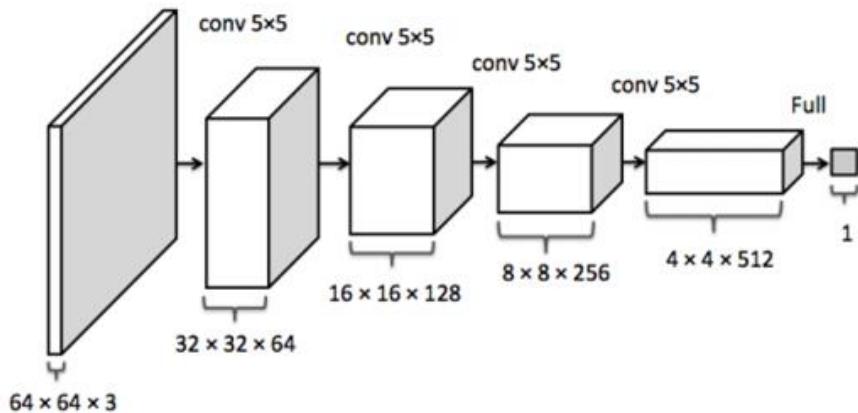


可以看出,生成器的输入是一个 100 维的噪声,中间会通过 4 层卷积层,每通过一个卷积层通道数减半,长宽扩大一倍,最终产生一个  $64 \times 64 \times 3$  大小的图片输出。值得说明的是,在很多引用 DCGAN 的 paper 中,误以为这 4 个卷积层是 deconv (反卷积) 层,但其实在 DCGAN 的介绍中这 4 个卷积层是 fractionally strided convolution (微步幅度卷积),二者的差别如下图所示:



上图左边是反卷积，用  $3 \times 3$  的卷积核把  $2 \times 2$  的矩阵反卷积成  $4 \times 4$  的矩阵；而右边是微步幅度卷积，用  $3 \times 3$  的卷积核把  $3 \times 3$  的矩阵卷积成  $5 \times 5$  的矩阵，二者的差别在于，反卷积是在整个输入矩阵周围添 0，而微步幅度卷积会把输入矩阵拆开，在每一个像素点的周围添 0。

接下来我们再看一下 DCGAN 中 D 网络的结构，由于原论文中没有给出相关图，我找了一篇近似论文中的图说明：



D 可以看成是 G 结构反过来的样子，那具体结构也没什么好说的了，简言之不断地做卷积，最终得到一个 0,1 之间的结果。

最后，见到有人在博客中给出了一个建议，引述如下：上述结构是 DCGAN 应用在 LSUN 数据集上的架构，我们自己搭建 DCGAN 的时候应该视实际数据集大小做相应更改，譬如在 MNIST 数据上网络结构的参数就应该要适当调小。

## 2. ImprovedDCGAN

GANs 的主要问题之一是收敛性不稳定，尽管 DCGAN 做了结构细化，训练过程仍可能难以收敛。我们先来分析一下为什么会出现收敛性不稳定。

GANs 的优化就是寻找两玩家非合作博弈的纳什均衡点。这里的两玩家指的就是生成器和判别器。生成器的目标是最小化目标函数：

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

判别器的目标是最小化目标函数：

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

纳什均衡点就是使得上面两个目标都能最小的  $\theta(D)$  和  $\theta(G)$ 。GANs 的训练常常是同时在两个目标上使用梯度下降，然而这并不能保证到达均衡点，毕竟目标未必是凸的。也就是说 GANs 可能永远达不到这样的均衡点，于是就会出现收敛性不稳定。

为了解决这一问题，ImprovedDCGAN 针对 DCGAN 训练过程提出了不同的增强方法。以下是其主要内容：

### 特征匹配(feature mapping)

为了不让生成器尽可能地去蒙骗鉴别器，ImprovedDCGAN 希望以特征作为匹配标准，而不是图片作为匹配标准，于是提出了一种新的生成器的目标函数，即：

$$\|\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \mathbf{f}(\mathbf{x}) - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \mathbf{f}(G(\mathbf{z}))\|_2^2$$

其中  $f(x)$  是指的生成器把判别器的中间层输出  $f(x)$  作为目标（一般中间层都是 D 最后几层， $f(x)$  实际上是 feature map），这样可以让生成的中间层输出和真实数据的中间层输出尽可能相同。这种做法虽然不能保证到达均衡点，但是收敛的稳定性应该是有所提高。

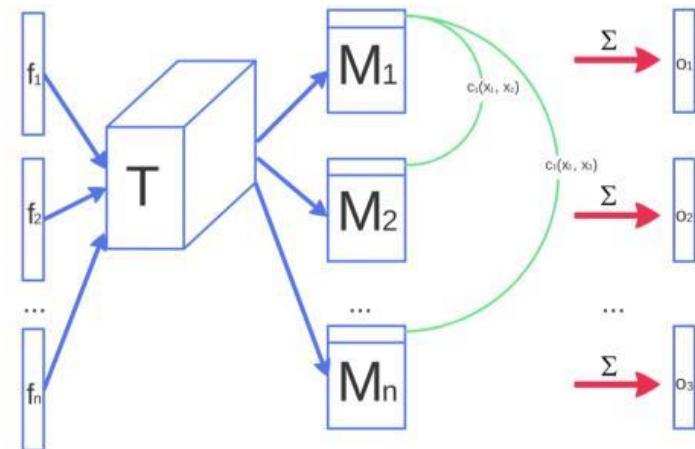
### 批次判别(minibatch discrimination)

GAN 的一个常见的失败就是收敛到同一个点，并没有什么机制可以让生成器生成不一样的内容。而只要生成一个会被判别器误认的内容，那么梯度方向就会不断朝那个方向前进。

ImprovedDCGAN 使用的方法是用 minibatch 判别器。也就是说每次不是判别单张图片，而是判别一批图片。

具体来说，将一个中间层  $f(x)$  乘以一个 tensor，得到一个新的矩阵  $M$ ，计算  $M$  每一行之间的 L1 距离  $o$ ，以此为  $f(x)$  下一层的输入。

$$\begin{aligned} o(\mathbf{x}_i)_b &= \sum_{j=1}^n c_b(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R} \\ o(\mathbf{x}_i) &= [o(\mathbf{x}_i)_1, o(\mathbf{x}_i)_2, \dots, o(\mathbf{x}_i)_B] \in \mathbb{R}^B \\ o(\mathbf{X}) &\in \mathbb{R}^{n \times B} \end{aligned}$$



假设  $x$  是从生成器生成的，并且收敛到同一点，那么对应的  $f(x)$  必然很相似，由  $f(x)$  生成的  $M$  也必然非常相似。而这些  $M$  每一行的 L1 距离  $c(x_i, x_j)$  也就会非常接近 0，这也导致  $o(X)$  几乎是 0 向量。相对的，如果  $x$  是真实数据，那么  $o(X)$  一般不会太接近 0 向量，这样判别器就可以更简单的区分生成数据或真实数据（在生成器收敛到一点的情况下）。

### 历史平均(historical averaging)

在更新参数值时，把它们过去的值也纳入考虑，也就是在目标函数中加入下面这项：

$$\|\theta - \frac{1}{t} \sum_{i=1}^t \theta[i]\|^2$$

### 单侧标签平滑(one-sided label smoothing)

判别式的目标函数中正负样本的系数不再是 0-1，而是  $\alpha$  和  $\beta$ ，这样判别式的目标就变成了下面这个式子。这相当于，原本判别器的目标输出值是 [0=假图像，1=真图像]，现在可能变成了 [0=假图像，0.9=真图像]。

$$D(\mathbf{x}) = \frac{\alpha p_{\text{data}}(\mathbf{x}) + \beta p_{\text{model}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{model}}(\mathbf{x})}$$

### 虚拟批次正态化(virtual batch normalization)

batch normalize 在神经网络领域有广泛应用，但是也有一些问题，比如特定样例  $x$  在神经网络上的输出会受到 minibatch 上其他的样本影响。文中提出了一种 virtual batch normalization (VBN)，会在训练前取一个 batch，以后就根据这个 batch 做 normalize，不过由于这样的计算成本很高，所以它仅仅被用在生成器当中。

综上便是 ImprovedDCGAN 提出来的改进方法，这些方法能够让模型在生成高分辨率图像时表现得更好，而这正是 GANs 的弱项之一。

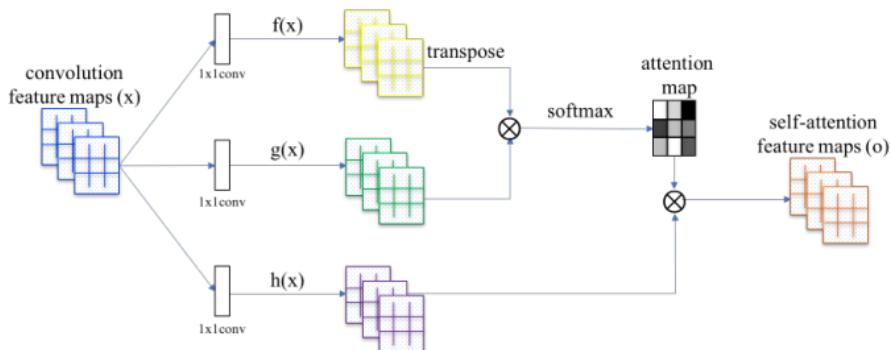
### 3. SAGAN

#### 3.1 SAGAN 解决的问题

前篇我们说到用深度卷积网络能够提升 GANs 生成高分辨率图片的细节，但是由于卷积网络的局部感受野的限制，如果要生成大范围相关（Long-range dependency）的区域，卷积网络就会出现问题。譬如说在生成人脸图片时，是非常注重细节的，以左右眼举例，只要左右眼有一点点不对称，就会显得生成的人脸特别不真实。但是因为一般的卷积核很难覆盖很大的区域，在对左眼区域做卷积时它看不到右眼对左眼的影响，这样产生的图片就会缺乏人脸结构特征的完整性。

因此，现在我们需要解决的问题是，如何找到一种能够利用全局信息的方法。传统的一些做法，比如用更深的卷积网络，或者直接采用全连接层获取全局信息，明显参数量太大计算量太大。直到 SAGAN 的提出，把 Attention 机制引入了 GANs 的图像生成当中，才找到一种比较简约且高效的方法解决了这一问题。

#### 3.2 SAGAN 的模型架构



上述结构就是用带有自注意力的特征图去代替传统的卷积特征图。方法如下：

首先， $f(x)$ 、 $g(x)$  和  $h(x)$  都是普通的  $1 \times 1$  卷积，差别只在于输出通道大小不同；将  $f(x)$  的输出转置，并和  $g(x)$  的输出相乘，再经过 softmax 归一化得到一个 attention map；将得到的 attention map 和  $h(x)$  逐像素点相乘，得到自适应注意力的特征图。

具体的计算方法如下：

我们记  $W_g \in R^{\bar{C} \times C}$ ,  $W_f \in R^{\bar{C} \times C}$ ,  $W_h \in R^{C \times C}$  是学习的权重矩阵，都是通过  $1 \times 1$  卷积实现，在实验中我们使用  $\bar{C} = C/8$ 。 $f$  与  $g$  表示两个提取图像特征空间的公式，其中  $f(x) = W_f x$ ,  $g(x) = W_g x$ 。

我们用  $\beta_{j,i}$  表示在合成第  $j$  个区域时模型对第  $i$  个位置的影响程度，有：

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^N \exp(s_{ij})}, \text{ where } s_{ij} = \mathbf{f}(\mathbf{x}_i)^T \mathbf{g}(\mathbf{x}_j)$$

然后关注层的输出是  $\mathbf{o} = (\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_j, \dots, \mathbf{o}_N) \in \mathbb{R}^{C \times N}$ ，其中，

$$\mathbf{o}_j = \sum_{i=1}^N \beta_{j,i} \mathbf{h}(\mathbf{x}_i), \text{ where } \mathbf{h}(\mathbf{x}_i) = W_h \mathbf{x}_i$$

另外，我们进一步将关注图层的输出  $o$  乘以比例参数 $\gamma$ 并添加回输入要素图  $x$ 。因此，最终输出由下式给出，

$$\mathbf{y}_i = \gamma \mathbf{o}_i + \mathbf{x}_i$$

其中 $\gamma$ 被初始化为 0，然后逐渐学会为非本地特征分配更多权重。为什么要这样做呢？理由很简单：我们希望先学习简单的任务，然后逐步增加任务的复杂性。在 SAGAN 中，所提出的自注意力模块已经应用于生成器和判别器，它们通过最小化对抗性损失的铰链形式以交替方式进行训练，SAGAN 采用的 loss 表达式如下：

$$\begin{aligned} L_D &= -\mathbb{E}_{(x,y) \sim p_{data}} [\min(0, -1 + D(x, y))] - \mathbb{E}_{z \sim p_z, y \sim p_{data}} [\min(0, -1 - D(G(z), y))] \\ L_G &= -\mathbb{E}_{z \sim p_z, y \sim p_{data}} D(G(z), y), \end{aligned}$$

### 3.3 SAGAN 的优化

SAGAN 当中提出了两种优化，分别是 Spectral Normalization 与 TTUR，前者稳定了训练和生成过程，后者平衡了 D 与 G 的训练速度，简要地介绍如下：

#### Spectral Normalization

SAGAN 为 D 和 G 加入了谱范数归一化的方式，让 D 满足了 1-lipschitz 限制，同时也避免了 G 的参数过多导致梯度异常，使得整套训练较为平稳和高效。关于 Spectral Normalization 的具体过程请参阅 Part1 中的 SNGAN。

#### TTUR

在以前的工作中，判别器的正则化通常会减慢 GAN 学习过程。实际上，使用正则化判别器的方法通常在训练期间每个生成器需要多个更新步骤（例如，5 个）。独立地，Heusel 等人主张对生成器和判别器使用单独的学习率 (TTUR)。我们建议专门使用 TTUR 来补偿正则化判别器中慢学习的问题，使得对于每个判别器步骤使用更少的生成器步骤成为可能。使用这种方法，我们能够在相同的单位时间内产生更好的结果。

综上，便是 SAGAN 提出来的用自注意力机制去解决全局信息获取的问题，它既在每一层都考虑了全局信息，也没有引入过多的参数量，在提高感受野和减小参数量之间找到了一个很好的平衡。因此，如果我们的生成任务是全局相关性比较高的图片，就可以考虑使用 SAGAN。

## 4. BigGAN

### 4.1 BigGAN 解决的问题

我们知道，GANs 的终极目标是生成让人无法辨别真伪的高清图片，如果用 Inception Score 来评价的话，我们希望生成图片的 IS 得分能逼近真实图片的 IS 值，也就是 233 分。但即便是之前效果最好的 SAGAN，IS 得分也只有 52 分。于是现在，我们要探索的就是，在好的硬件条件 (TPU) 和庞大的参数体系下，GANs 生成的图片究竟能逼真、精细到什么程度，基于这样的好奇心，BigGAN 被提出了。

### 4.2 BigGAN 的大规模实现

为了产生逼真、精细的图片，我们需要提升 GANs 的规模。首先我们想到的，就是增大 BatchSize 以及增大 Chanel，以及我们可以不断尝试修改其他的参数和方法，在后文当中会逐一细述。我们先看一下实验结果（表中的各列内容会在后文一一解释）：

Batch	Ch.	Param (M)	Shared	Hier.	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5		SA-GAN Baseline			1000	18.65
512	64	81.5	✗	✗	✗	1000	15.30	58.77( $\pm 1.18$ )
1024	64	81.5	✗	✗	✗	1000	14.88	63.03( $\pm 1.42$ )
2048	64	81.5	✗	✗	✗	732	12.39	76.85( $\pm 3.83$ )
2048	96	173.5	✗	✗	✗	295( $\pm 18$ )	9.54( $\pm 0.62$ )	92.98( $\pm 4.27$ )
2048	96	160.6	✓	✗	✗	185( $\pm 11$ )	9.18( $\pm 0.13$ )	94.94( $\pm 1.32$ )
2048	96	158.3	✓	✓	✗	152( $\pm 7$ )	8.73( $\pm 0.45$ )	98.76( $\pm 2.84$ )
2048	96	158.3	✓	✓	✓	165( $\pm 13$ )	8.51( $\pm 0.32$ )	99.31( $\pm 2.10$ )
2048	64	71.3	✓	✓	✓	371( $\pm 7$ )	10.48( $\pm 0.10$ )	86.90( $\pm 0.61$ )

表 1：我们提出的修改的模型下的 Fréchet Inception Distance (FID，越低越好) 和 Inception Score (IS，越高越好)。Batch 是批量大小，Param 是参数总数，Ch. 是每层中单元数的通道乘数，Shared 表示是否使用共享嵌入，Hier. 是否使用分层潜在空间，Ortho. 是否正交正则化，Itr 如果值为 1000，则表示该设置对  $10^6$  次迭代是稳定的，否则表示在该迭代次数下它就崩溃了。除了行 1-4 之外，还计算了 8 个不同随机初始化的结果。

#### Batch (BatchSize)

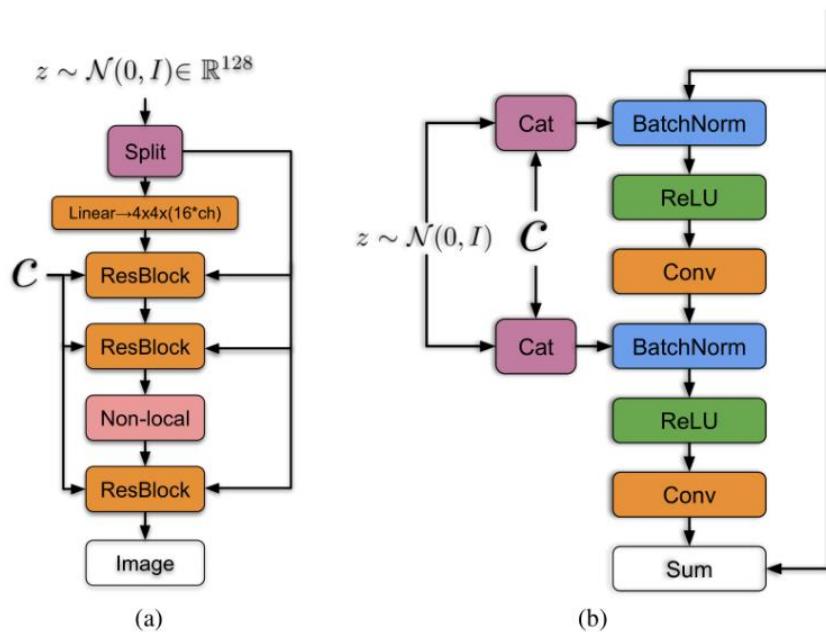
简单地增大 BatchSize 就可以实现性能上较好的提升，例如 Batch size 从 256 增大 2048 的时候，IS 提高了 46%，BigGAN 推测这可能是每批次覆盖更多内容的结果，为生成和判别两个网络提供更好的梯度。增大 Batch size 还会带来在更少的时间训练出更好性能的模型，但增大 Batch size 也会使得模型在训练上稳定性下降，4.3 节会分析如何提高稳定性。

#### Ch. (Channel)

在实验上，单单提高 Batch size 还受到限制，BigGAN 在每层的通道数也做了相应的增加，当通道增加 50%，大约两倍于两个模型中的参数数量。这会导致 IS 进一步提高 21%。文章认为这是由于模型的容量相对于数据集的复杂性而增加。有趣的是，BigGAN 在实验上发现一味地增加网络深度并不会带来更好的结果，反而在生成性能上会有一定的下降。

#### Shared

首先给出 BigGAN 生成网络的结构图，顺便能说明什么是 Shared (共享嵌入)：



如左图所示将噪声向量  $z$  通过 `split` 等分成多块，然后和条件标签  $c$  连接后一起送入到生成网络的各个层中，对于生成网络的每一个残差块又可以进一步展开为右图的结构。可以看到噪声向量  $z$  的块和条件标签  $c$  在残差块下是通过 `concat` 操作后送入 `BatchNorm` 层，这种嵌入方式就是共享嵌入，线性投影到每个层的 `bias` 和 `weight`。共享嵌入与传统嵌入的差别是，传统嵌入为每个嵌入分别设置一个层，而共享嵌入是将  $z$  与  $c$  的连接一并传入所有 `BatchNorm`。

再回到表 1 中的实验，BigGAN 采用了共享嵌入后，降低了计算和内存成本，并将训练速度（达到给定性能所需的迭代次数）提高了 37%。

## Hier. (Hierarchical Latent Space)

先解释一下潜在空间 (**Latent Space**)，它实际上指的就是噪声  $z$  的先验分布。BigGAN 发现，虽然大多数以前的工作采用  $N(0, I)$  或  $U[-1, 1]$  作为  $z$  的先验（输入到  $G$  的噪声），但实际上我们可以自由选择能够采样的任何潜在分布（详见 paper 附件 E），譬如 BigGAN 发现效果更好的两个潜在分布是  $Bernoulli\{0, 1\}$  和  $Censored Normal \max(N(0, I), 0)$ ，两者都提高了训练速度并轻微提高了最终性能，但是最终 BigGAN 没有考虑替换潜在分布的方案，因为 BigGAN 发现了比这更有用的截断技巧。

截断技巧不需要替换潜在空间，我们依然使用  $N(0, I)$  的先验分布。截断技巧的做法是在对先验分布  $z$  采样的过程中，通过设置阈值的方式来截断  $z$  的采样，其中超出范围的值被重新采样以落入该范围内。这个阈值可以根据生成质量指标 IS 和 FID 决定。实验的结果是，随着阈值的下降生成的质量会越来越好，但是由于阈值的下降、采样的范围变窄，就会造成生成上取向单一化，造成生成的多样性不足的问题，数据上来说就是，IS（反应图像的生成质量）一路上涨，FID（注重生成的多样性）先变好然后一路变坏。

BigGAN 不仅对潜在空间内部作了处理，在潜在空间的处置上用到了**分层潜在空间 (Hierarchical Latent Space)** 技术。分层潜在空间的意思是，传统的 GAN 都是将  $z$  作为输入直接嵌入生成网络，而 BigGAN 将噪声向量  $z$  送到  $G$  的多个层而不仅仅是初始层。BigGAN 认为潜在空间  $z$  可以直接影响不同分辨率和层次结构级别的特征，所以对于 BigGAN 的条件生成模型，将  $z$  分成每个分辨率的一个块，并将每个块连接到条件向量  $c$  来实现，实验结果证明，这样做提供约 4% 的适度性能提升，并将训练速度提高 18%。

### Ortho. (Orthogonal Regularization)

前面我们提到, 使用截断技巧可以提升生成图片的质量, 但是一些较大的模型不适合截断, 因为在嵌入截断噪声时会产生饱和伪影, 如下图所示:



为了抵消这种情况, BigGAN 通过将  $G$  调节为平滑来强制实现截断的适应性, 以便  $z$  的整个空间能映射到良好的输出样本。为此, BigGAN 采用正交正则化方法 (Orthogonal Regularization)。正交正则化, 也就是说让  $W$  权重矩阵尽可能是一个正交矩阵, 这样最大的好处是, 权重系数彼此之间的干扰会非常低, 受到截断之后消失的权重就不会对结果产生太大影响。

为了实现正交化, 一开始 BigGAN 想到了一个非常粗俗的方法, 就是直接把正则项改为:

$$R_\beta(W) = \beta \|W^\top W - I\|_F^2,$$

其中  $W$  是权重矩阵,  $\beta$  是超参数,  $I$  是单位矩阵, 这基本上就等价于一个正交矩阵的定义式 (若矩阵与自身转置的乘积为单位矩阵, 则该矩阵是正交矩阵)。但是这种方法明显太过局限了, BigGAN 为了放松约束, 同时实现模型所需的平滑度, 发现最好的版本是从正则化中删除对角项, 并且旨在最小化卷积核之间的成对余弦相似性, 但不限制它们的范数:

$$R_\beta(W) = \beta \|W^\top W \odot (I - I)\|_F^2,$$

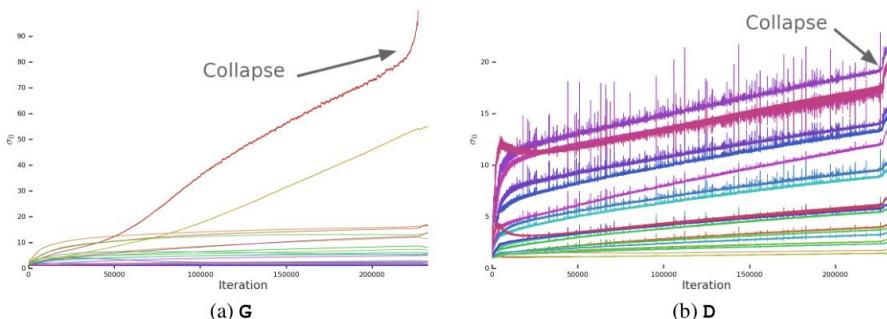
实验结果证明, 正交正则化的方法是非常有帮助的, 在表 1 中, 没有正交正则化时, 只有 16% 的模型适合截断, 而有正交正则化训练时则有 60%。

## 4.3 BigGAN 的稳定性实现

在前篇中, 我们初步实现了 BigGAN 的大规模, 并带来了实质的结果提升。但是, 大规模 BigGAN 有一个最大的问题就是非常不稳定, 在 Batch 很大的时候是非常容易崩溃的。在本节我们首先会分析产生崩溃的原因, 然后探讨解决这一问题的方法。

### 分析崩溃的原因

BigGAN 在实验中发现, 每个权重矩阵中的前三个奇异值  $\sigma_0, \sigma_1, \sigma_2$  的影响最大 (这三个奇异值可以使用 Arnoldi 迭代方法进行有效计算)。我们以  $\sigma_0$  为例, 我们先画出光谱归一化之前  $G$  和  $D$  层中第一个奇异值  $\sigma_0$  的典型图如下 (从红色到紫色的颜色表示增加深度):



可以看出，左图中大多数 G 层具有良好的光谱范式，但有些层（通常是 G 中的第一层，过于完整且非卷积）表现不佳，光谱范式在整个训练过程中增长，在崩溃时爆炸。而对于右图，D 的光谱噪声较大，但表现更好。

因此，现在需要解决的问题是，对于 G，适当调整奇异值  $\sigma_0$  以抵消光谱爆炸的影响；对于 D，寻找更多的约束来抵消噪声的影响，实现训练的稳定性。

### 对于 G 的控制

为了将权重的第一个奇异值  $\sigma_0$  控制住，防止突然性的爆炸，BigGAN 采用了两种方法：

- 第一种方法是，直接使每个权重的顶部奇异值  $\sigma_0$  正则化，朝向固定值  $\sigma_{reg}$  或者以某个比率  $r$  朝向第二奇异值  $r \cdot sg(\sigma_1)$ （其中  $sg$  为停止梯度操作，适时防止正则化增加  $\sigma_1$ ）。
- 第二种方法是，使用部分奇异值分解来代替  $\sigma_0$ 。给定权重  $W$ ，其第一个奇异向量  $u_0$  和  $v_0$ ，以及  $\sigma_0$  将被值  $\sigma_{clamp}$  钳制，我们的权重变为：

$$W = W - \max(0, \sigma_0 - \sigma_{clamp}) v_0 u_0^\top$$

其中  $\sigma_{clamp}$  被设置为  $\sigma_{reg}$  或  $r \cdot sg(\sigma_1)$ 。

BigGAN 观察到无论有无光谱归一化，这些技术都具有防止  $\sigma_0$  或  $\frac{\sigma_0}{\sigma_1}$  逐渐增加和爆炸的效果，但即使在某些情况下它们可以温和地提高性能，却依然没有任何组合可以防止训练崩溃。这一证据表明，虽然调节 G 可能会改善稳定性，但它不足以确保稳定性。因此，现在我们需要将注意力转向 D。

### 对于 D 的控制

在图 (b) 中，我们看到 D 的光谱是嘈杂的，但是  $\frac{\sigma_0}{\sigma_1}$  表现良好，并且奇异值在整个训练过程中平稳增长，在崩溃时只是突然跳跃而不是爆炸。

我们需要解决的问题有两个，第一个，嘈杂出现的原因是什么以及这种嘈杂与模型不稳定性之间是否有直接影响；第二个，奇异值在整个训练过程中平稳增长（D 在训练期间的损失接近于零），但在崩溃时经历了急剧的向上跳跃，这是什么原因导致的。

我们先考虑第一个问题，频谱噪声与模型不稳定性之间有什么样的影响。

我们先分析一下出现嘈杂的原因，D 光谱中的峰值可能表明它周期性地接收到非常大的梯度，但我们观察到 Frobenius 规范是平滑的（见 paper 附录 F），表明这种效应主要集中在前几个奇异方向上。于是我们认为这种噪声是通过对抗训练过程进行优化的结果，其中 G 定期产生强烈干扰 D 的 batch，进而导致出现光谱嘈杂。

如果这种频谱噪声与不稳定性有因果关系，我们该采用的反制措施自然是梯度惩罚，因为这明显地规范了 D 的雅可比行列式的变化。BigGAN 探索了  $R_1$  零中心梯度罚分：

$$R_1 := \frac{\gamma}{2} \mathbb{E}_{p_D(x)} [\|\nabla D(x)\|_F^2]$$

实验结果证明，在  $\gamma$  为 10 的情况下，训练变得稳定并且改善了 G 和 D 中光谱的平滑度和有界性，但是性能严重降低，导致 IS 减少 45%。减少惩罚可以部分缓解这种恶化，但会导致频谱越来越不良。即使惩罚强度降低到 1（没有发生突然崩溃的最低强度），IS 也减少了 20%。

BigGAN 还采用了很多其他正则化策略进行比较，得到的结论就是：频谱噪声确实会对模型的不稳定性产生影响，我们可以通过对 D 施加惩罚去解决。当对 D 的惩罚足够高时，可以实现训练的稳定性提升但是图像的生成质量会下降比较多。

下面来考虑第二个问题，奇异值在崩溃时的向上跳跃是什么原因导致的。

首先我们直观上猜测，出现这种现象很有可能是 D 过度拟合训练集，从而记忆训练样

本而不是学习真实图像和生成图像之间的一些有意义的边界，所以在训练 D 时损失接近 0 但在崩溃时就会出现突然的跳跃。为了评估这一猜测，BigGAN 在 ImageNet 训练和验证集上评估判别器，并测量样本分类为真实或生成的百分比。虽然在训练集下精度始终高于 98%，但验证准确度在 50-55% 的范围内，这并不比随机猜测更好（无论正则化策略如何）。这证实了 D 确实记住了训练集。

但事实上我们不需要对此感到意外，因为这符合 D 的角色：不断提炼训练数据并为 G 提供有用的学习信号。因此针对第二个问题我们不需要做过多的调整，因为 D 过拟合对模型稳定性影响不大，我们只需要保证 G 能接受到正确的学习信号即可。

## 总结

我们发现稳定性不仅仅来自 G 或 D，而是来自他们通过对抗性训练过程的相互作用。虽然他们的不良症状调节可用于追踪和识别不稳定性，但确保合理的调节是训练所必需的，但不足以防止最终的训练崩溃。可以通过强烈约束 D 来强制实现稳定性，但这样做会导致性能上的巨大成本。使用现有技术，可以通过放松这种调节并允许在训练的后期阶段发生塌陷来实现更好的最终性能，此时模型经过充分训练以获得良好的结果。

最终的 BigGAN，将 IS 得分提升到了惊人的 166 分，可以说，BigGAN 是当时（截至写稿日 18 年 10 月 14 日）细节效果处理得最好的生成模型，它将 GANs 的大规模与稳定性实现了较大的提升与平衡，产生让人惊叹的结果。最后用几张 BigGAN 的生成范例来结束这一章的学习：



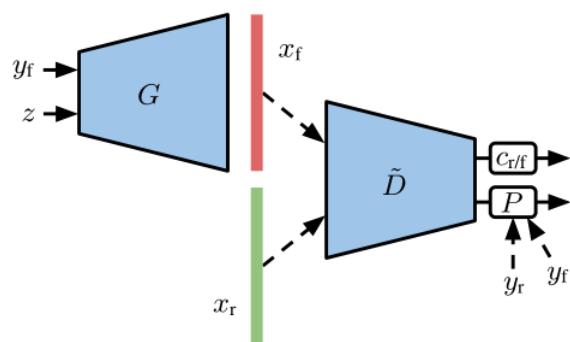
## 5. S<sup>3</sup>GAN

### 5.1 S<sup>3</sup>GAN 解决的问题

上一节我们提到，BigGAN 实现了大规模高清图片的生成，但是，BigGAN 的训练成本是非常大的，一个显著的缺陷是它需要大量的标注数据才能实现训练（因为判别器做判断的依据就是图像的标注）。因此，制作一个大规模的有标记图片数据集是耗时耗力的，为了解决这一问题，S<sup>2</sup>GAN，S<sup>2</sup>GAN-CO，以及S<sup>3</sup>GAN 相继被提出了，这些方法仅使用 10% 有标记的数据就能够匹配 ImageNet 上 BigGAN 生成的样本质量，并且在使用 20% 有标记的数据时超过它。

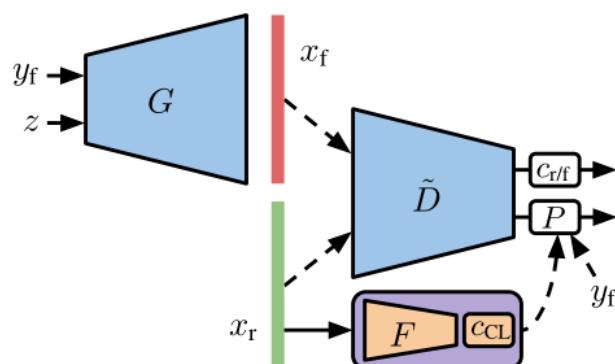
### 5.2 S<sup>2</sup>GAN

我们先回顾一下 BigGAN 使用的模型架构(重点关注判别器)，如下图：



这是一种带有投影判别器的条件 GAN。判别器试图通过组合无条件分类器  $c_{r/f}$  和通过投影层  $P$  实现的类条件分类器来从表示  $\tilde{D}$  中预测实际图像  $x_r$  (具有标签  $y_r$ ) 或生成图像  $x_f$  (具有标签  $y_f$ ) 是否在其输入处。

现在我们不希望这些标签都由人工来标注，于是我们想引入一个新的网络，能够为图像自动添加标签。这个学习网络的构造方式有两种，一种是无监督方法，另一种是半监督方法。



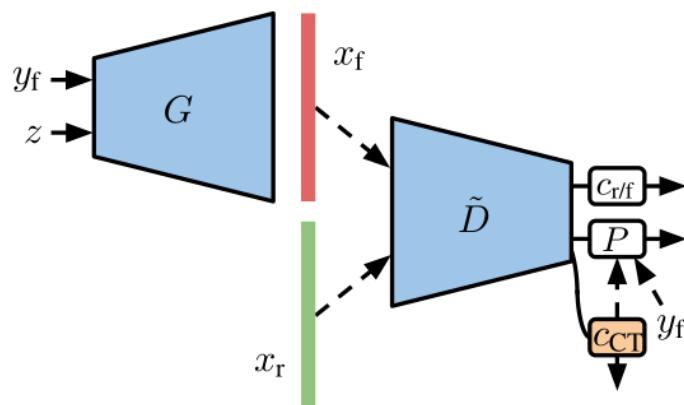
上图就是无监督方法的网络架构。很好理解，给判别器加一个特征提取器（图中的  $F$ ），从没有标注的真实训练数据里面，学到它们的表征。对这个表征做聚类（图中  $c_{CL}$ ），然后把聚类的分配结果，当成标注来用。损失函数用自监督损失。这种方法不需要任何的已有标注数据，让网络通过对真实数据聚类从而学会去给真数据与假数据添加不同的标签。

另外一种方法是基于半监督方法的网络架构，它可以利用少量已标注的数据进行学习。遗憾的是，原论文中没有为这种方法绘制示意图，但是用简单的文字也可以很好解释：在训练集的一个子集已经标注过的情况下，根据这些已知信息来学习表征，同时训练一个线性分类器。这样，损失函数会在自监督的基础上，再加一项半监督的交叉熵损失。预训练了特征提取器之后，就可以拿去训练 GAN 了。

上面第二种基于半监督学习的方法，即用一小部分已知标注训练出的 GAN，就叫做 S<sup>2</sup>GAN。

### 5.3 S<sup>2</sup>GAN-CO

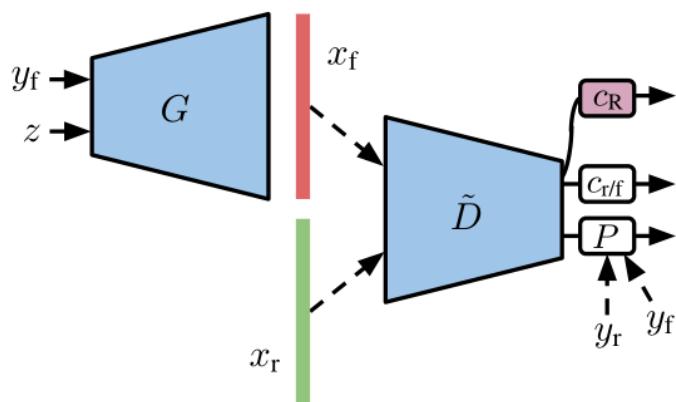
S<sup>2</sup>GAN 有一个不太好的地方是，预训练网络在实际训练中会占用很多时间，现在我们不想要这个预训练的过程，我们希望 GANs 的训练与标记网络的训练能同时进行，于是，S<sup>2</sup>GAN 的协同版——S<sup>2</sup>GAN-CO 被提出了。



如上图所示，直接在判别器的表征 ( $\tilde{D}$ ) 上面，训练一个半监督的辅助分类器 ( $c_{CT}$ )，用来预测没有标注的图像。这个过程，和 GAN 的训练一同进行。这个结构，就叫做 S<sup>2</sup>GAN-CO。

### 5.4 S<sup>3</sup>GAN

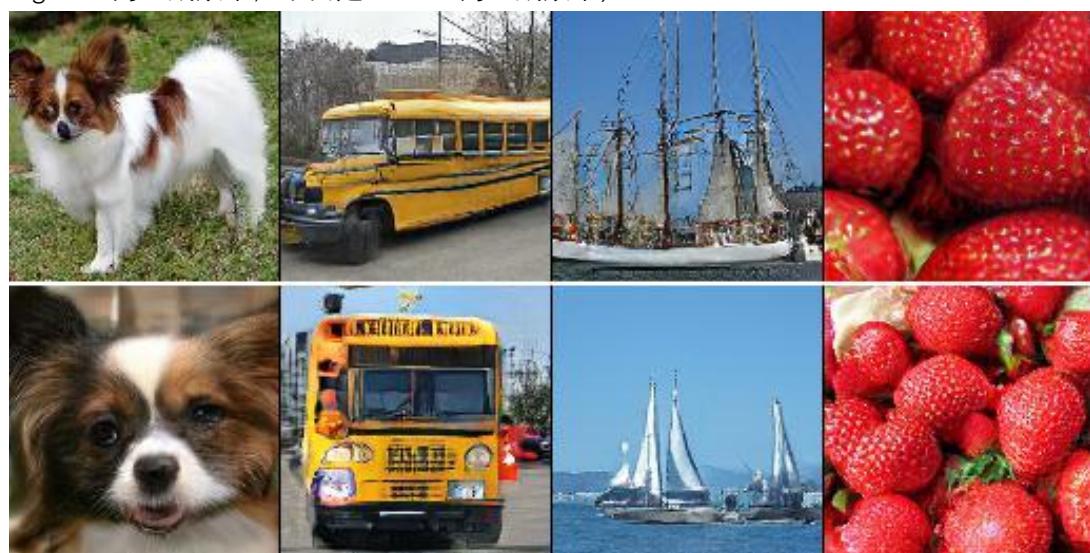
S<sup>2</sup>GAN 还不够强大，因为稳定性不佳一直是 GANs 训练过程中容易出现的问题（可参见 BigGAN 中的相关讨论），为了提升 S<sup>2</sup>GAN 的训练稳定性，S<sup>3</sup>GAN 被提出了。



如上图所示，考虑到判别器本身是一个分类器，如果把这个分类器扩增一下，对于提升训练的稳定性是有帮助的。于是，S<sup>3</sup>GAN 给了分类器一个额外的自监督任务，就是为旋转扩增过的训练集（包括真图和假图），做个预测。再把这个步骤，和前面的半监督模型结合起来，就得到了上图所示 S<sup>3</sup>GAN 的网络架构图。实验证明，升级版的 S<sup>3</sup>GAN 确实在训练的稳定性上得到提升。

综上，就是 S<sup>2</sup>GAN 及 S<sup>3</sup>GAN 的介绍，它们仅仅是在 BigGAN 的网络架构上做出了进一步调整，就实现了与 BigGAN 相媲美甚至超越 BigGAN 的图片生成表现，最重要的是它们为无监督的 GANs 训练提供了一个非常好的样板，解决了传统 GANs 过于依赖标记数据的问题。

最后，让我们来观摩一下 S<sup>3</sup>GAN 生成图片与 BigGAN 生成图片的对比。（上面是 BigGAN 的生成效果，下面是 S<sup>3</sup>GAN 的生成效果）。



## Part3 GANs 的其他改进

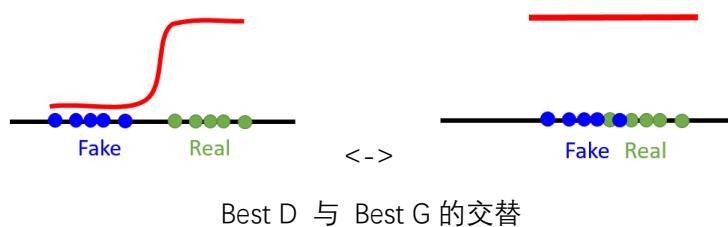
GANs 还有一些其他的改进，本部分会选择一些很有创意的 idea 介绍一下。

### 1. RGAN

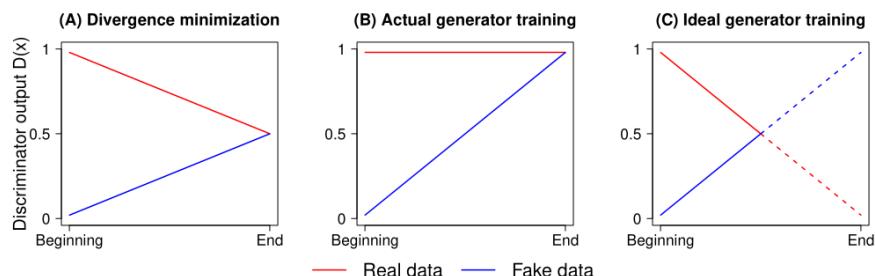
#### 1.1 RGAN 解决的问题

在标准生成对抗网络中，判别器 D 估计输入数据是真实的概率，训练生成器 G 以增加伪造数据是真实的概率，但是现在 RGAN (RelativisticGAN) 认为，在提升伪造数据是真实的概率的同时还应该降低真实数据是真实的概率。

首先解释为何传统 GANs 中，随着伪造数据得分的提升，真实数据的得分不会下降。其实这是显而易见的，假设 D 和 G 在每个交替步骤中都训练到了最佳，也就是说，D 训练结束时  $D(x_r) = 1$ ,  $D(x_f) = 0$ ，而 G 训练结束时  $D(x_r) = 1$ ,  $D(x_f) = 1$ 。由此可以发现，在训练交替过程中， $D(x_r)$  都维持着 1，这意味着  $x_r$  接受到的梯度是 0，因此不会下降。



如果真实数据的得分不会下降，会造成一些麻烦，如下图所示：



理论上（如图 A 所示），如果判别器测量的就是 Div 的最小值，最终真实数据与伪造数据的得分都应该是 0.5。但是实际上（如图 B 所示），由于训练过程中 G 只是不断提升伪造数据是真实的概率，最终真实数据与伪造数据的得分都会趋近于 1，这样就会造成一些问题。

比如，这会影响 JS Div 计算的有效性，因为它过分看重  $\max_{D:X \rightarrow [0,1]} \mathbb{E}_{x_r \sim \mathbb{P}}[\log(D(x_r))] + \mathbb{E}_{x_f \sim \mathbb{Q}}[\log(1 - D(x_f))]$  的第二项而忽视了第一项，以至于在第二项没有什么提升空间时模型依然专注于提升第二项却忽视了第一项的作用。再比如，假设我们先验知识中知道真实数据与伪造数据各占一半，图 A 的结果能体现这一点，因为判真和判假的概率各是 0.5，但如果是图 B 所示的结果，所有的数据都成了真实数据，这就与我们的先验知识矛盾了。

为了解决  $D_{real}$  不降低的问题，RGAN 被提出了，它富有创意地提出采用相对判别器，在训练过程中不仅让  $D_{fake}$  向  $D_{real}$  移动，而且让  $D_{real}$  也向  $D_{fake}$  移动。

## 1.2 相对判别器

传统的判别器给数据的真实度打分，这会导致真实数据的得分居高不下，现在 RGAN 提出采用相对判别器，采用评估给定的实际数据比随机抽样的假数据更真实的概率。具体来说，传统 GANs 的目标式是如下：

$$L_D^{GAN} = \mathbb{E}_{x_r \sim \mathbb{P}} [f_1(C(x_r))] + \mathbb{E}_{x_f \sim \mathbb{Q}} [f_2(C(x_f))]$$

$$L_G^{GAN} = \mathbb{E}_{x_r \sim \mathbb{P}} [g_1(C(x_r))] + \mathbb{E}_{x_f \sim \mathbb{Q}} [g_2(C(x_f))],$$

其中  $f_1, f_2, g_1, g_2$  是标量-标量函数，C 是判别函数。如果我们使用相对判别器，目标式就变为：

$$L_D^{RGAN} = \mathbb{E}_{(x_r, x_f) \sim (\mathbb{P}, \mathbb{Q})} [f_1(C(x_r) - C(x_f))] + \mathbb{E}_{(x_r, x_f) \sim (\mathbb{P}, \mathbb{Q})} [f_2(C(x_f) - C(x_r))]$$

and

$$L_G^{RGAN} = \mathbb{E}_{(x_r, x_f) \sim (\mathbb{P}, \mathbb{Q})} [g_1(C(x_r) - C(x_f))] + \mathbb{E}_{(x_r, x_f) \sim (\mathbb{P}, \mathbb{Q})} [g_2(C(x_f) - C(x_r))]$$

我们来比较一下二者。在传统 GANs 中， $g_1$  的值是 0，因为传统 GANs 的 G 只需考虑  $C(x_f)$  尽可能高；但是在 RGAN 中， $g_1$  的值不是 0，因为我们衡量的是真实数据的判别值减去伪造数据的判别值，我们希望在 G 中这个值越小越好，于是我们就产生了一个梯度，能够引导  $C(x_r)$  朝着更小的方向移动。由此可以看出，在 RGAN 中， $C(x_r)$  是受到 G 的影响的，也就是说我们的训练过程中存在  $D_{real}$  降低的过程。

## 1.3 相对平均判别器

相对判别器还有一个变种，叫做相对平均判别器，对应的 GAN 被称作 RaGAN。

相对平均判别器的做法，是评估平均给定的真实数据要比随机抽样的假数据更加真实的概率。我们只需要把目标式中的  $C(x)$ ，改成期望值即可：

$$L_D^{RaGAN} = \mathbb{E}_{x_r \sim \mathbb{P}} [f_1(C(x_r) - \mathbb{E}_{x_f \sim \mathbb{Q}} C(x_f))] + \mathbb{E}_{x_f \sim \mathbb{Q}} [f_2(C(x_f) - \mathbb{E}_{x_r \sim \mathbb{P}} C(x_r))]$$

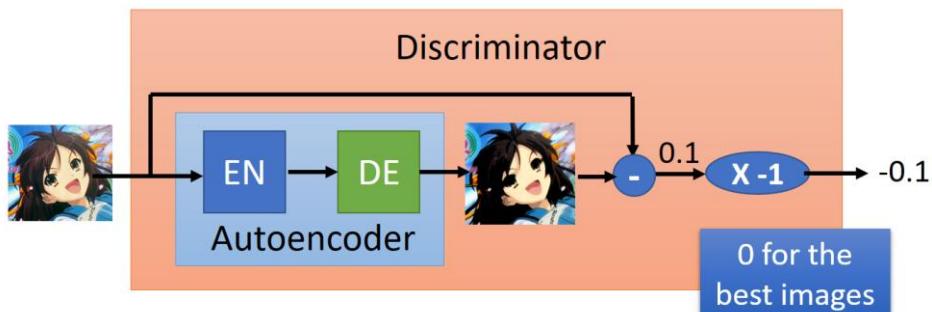
$$L_G^{RaGAN} = \mathbb{E}_{x_r \sim \mathbb{P}} [g_1(C(x_r) - \mathbb{E}_{x_f \sim \mathbb{Q}} C(x_f))] + \mathbb{E}_{x_f \sim \mathbb{Q}} [g_2(C(x_f) - \mathbb{E}_{x_r \sim \mathbb{P}} C(x_r))]$$

RaGAN 的显著优势是，能从比较小的样本生成高分辨率图像，并且到达最优性能的训练时间大幅减少。以 WGAN-GP 为例，使用了相对平均判别器后训练时间减少了 3/4。

综上，不论是相对判别器，还是相对平均判别器，都能够被直接接到一个非基于积分概率度量 (IPM) 的 GANs 中使用（因为基于 IPM 的 GANs 已经被证明自带相对（平均）判别器，譬如对于 WGAN 以及 WGAN-GP，lipschitz 限制就相当于相对判别器的作用，详见 paper）。实验证明，含有 RGAN 和 RaGAN 的结构能更稳定并产生更高质量的数据样本。

## 2. EBGAN

EBGAN 的全称是 Energy-Based GAN，它只改动了判别器，使其不再去鉴别输入图像是来自于 $P_{data}$ 还是 $P_g$ ，而是去鉴别输入图像的重构性高不高。具体来说就是，本来判别器的目的是学会寻找 $P_{data}$ 与 $P_g$ 之间的差异，进而给图片质量打分，现在我们不再是通过寻找差异来打分，而是用一种“强烈的记忆”让判别器仅仅记住 $P_{data}$ 长什么样子，然后对于一个任意的输入 $x$ ，只要 $x$ 符合这个“记忆”中的样子就给高分，只要 $x$ 与“记忆”中的样子有差异就给低分。EBGAN 就是用 autoencoder 实现了这样的“记忆”，我们会在接下来详细介绍。



首先值得说明的是，图中的 autoencoder 是提前用真实图片训练好的，也就是说，如果输入是来自真实数据集的图片，这个 autoencoder 就能产生和输入非常相似的图片；但是如果输入的是其他图片，autoencoder 的输出就不会和输入相似。现在把这个 autoencoder 放入判别器中，每当判别器输入一张 image  $x$ ，通过这个 autoencoder 得到重构图像  $x'$ ，我们就能用  $x$  与  $x'$  的差值作为评判输入图像  $x$  质量好坏的标准，当差值越低的时候意味着输入图片越符合真实图片的特征。

由此我们可以看到，EBGAN 的最大特点就是判别器一开始就非常强（因为有 pretrain），因此生成器一开始就能获得比较大的“能量驱动”(energy based)，使得在一开始生成器就进步非常快。所以如果我们比较看中训练效率，希望在短期内获得一个比较不错的生成器的话，就可以考虑 EBGAN。

另外，EBGAN 的理论基础其实是非常复杂的，我就不放在这儿介绍了，如果有对能量模型的理论感兴趣的读者，可以去阅读以下这篇文章：

<http://www.gwylab.com/note-ebgan.html>

### 3.\* BEGAN

BEGAN 看着与 EBGAN 好像，二者确实存在着一些关联。BEGAN 的全称是 Boundary Equilibrium GAN (边界均衡 GAN)，它借鉴了 EBGAN 和 WGAN 各自的一些优点，提出了一种新的评价生成器生成质量的方式，使 GAN 即使使用很简单的网络，也能实现很好的训练效果，完全不用担心模式崩溃 (model collapse) 和训练不平衡的问题。

直观来讲，如果两个分布越相近，我们可以认为他们越相似，当生成数据分布非常接近于真实数据分布的时候，这时候生成器就有足够的生成能力。BEGAN 代替了这种估计概率分布方法，它不直接去估计生成分布  $P_g$  与真实分布  $P_x$  的差距，而是估计分布的误差分布之间的差距，作者认为只要分布之间的误差分布相近的话，也可以认为这些分布是相近的。

BEGAN 中，作者做出了以下四个贡献：

1. 提出了一种新的简单强大 GAN 网络结构，使用标准的训练方式不加训练 trick 也能很快且稳定的收敛
2. 对于 GAN 中 G, D 的能力的平衡提出了一种均衡的概念
3. 提供了一个超参数，这个超参数可以在图像的多样性和生成质量之间做均衡
4. 提出了一种收敛程度的估计，这个机制只在 WGAN 中出现过。

这篇 paper 比较偏理论了，我就不再做详细介绍了（逃），感兴趣的读者可以自行去研究，附上一篇非常好的讲解博文：

[https://blog.csdn.net/qq\\_25737169/article/details/77575617?locationNum=1&fps=1](https://blog.csdn.net/qq_25737169/article/details/77575617?locationNum=1&fps=1)

## 第三章 GANs 的应用

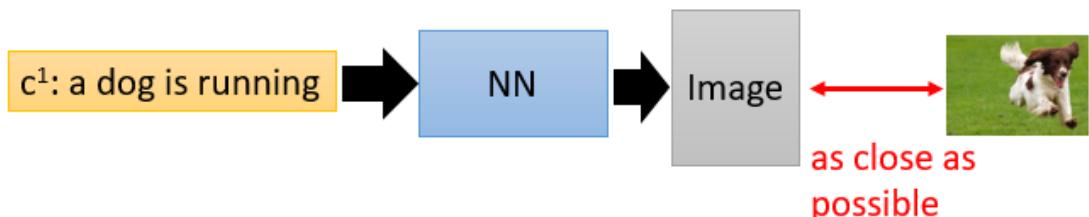
相比第二章而言, 第三章的内容会显得轻松许多, 并且很有趣(因为本章没啥数学理论, 全是 idea~)。

### Part1 GANs 在图像生成上的应用

#### 1. CGAN

##### 1.1 传统GANs 的问题

我们假设现在要做一个项目: 输入一段文字, 输出一张图片, 要让这张图片足够清晰并且符合这段文字的描述。我们搭建一个传统的 NeuralNetwork (下称 NN) 去训练。



考虑我们输入的文字是“train”, 希望 NN 能输出清晰的火车照片, 那在数据集中, 下面左图是正面的火车, 它们统统都是正确的火车图片; 下面右图是侧面的火车, 它们也统统都是正确的火车。



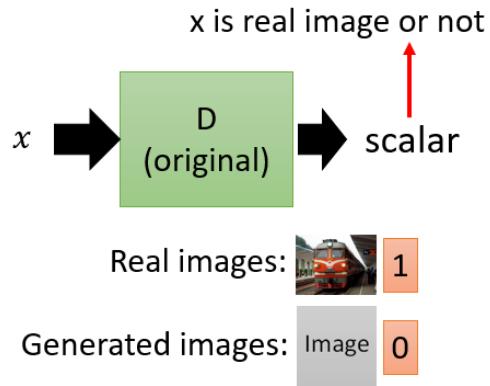
那在训练这个 NN 的时候, 网络会觉得说, 火车既要长得像左边的图片, 也要长得像右边的图片, 那最终网络的输出就会变成这一大堆图片的平均, 可想而知那会是一张非常模糊并且错误的照片。

我们需要引入 GANs 技术来保证 NN 产生清晰准确的照片。

我们把原始的 NN 叫做 G (生成器), 现在它吃两个输入, 一个是条件 word: c, 另外一个是从原始图片中采样出的分布 z, 它的输出是一个 image: x, 它希望这个 x 尽可能地符合条件 c 的描述, 同时足够清晰, 如下图。



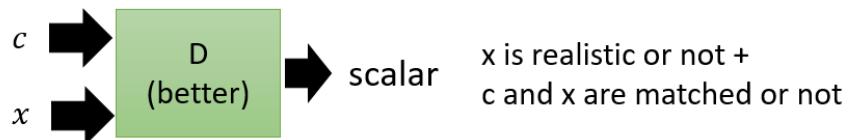
在 GANs 中为了保证输出 image 的质量会引入一个 D (判别器), 这个 D 用来判断输入的  $x$  是真实图片还是伪造图片, 如下图。



但是传统 GANs 只能保证让  $x$  尽可能地像真实图片, 它忽略了让  $x$  符合条件描述  $c$  的要求。于是, 为了解决这一问题, CGAN 便被提出了。

## 1.2 CGAN 的原理

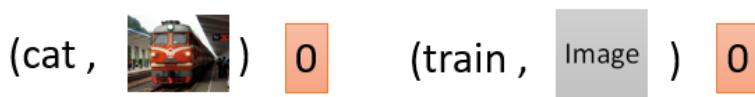
我们的目的是, 既要让输出的图片真实, 也要让输出的图片符合条件  $c$  的描述。判别器输入便被改成了同时输入  $c$  和  $x$ , 输出要做两件事情, 一个是判断  $x$  是否是真实图片, 另一个  $x$  和  $c$  是否是匹配的。



比如说, 在下面这个情况中, 条件  $c$  是 train, 图片  $x$  也是一张清晰的火车照片, 那么 D 的输出就会是 1。



而在下面两个情况中, 左边虽然输出图片清晰, 但不符合条件  $c$ ; 右边输出图片不真实。因此两种情况中 D 的输出都会是 0。



那 CGAN 的基本思路就是这样, 下面我们具体看一下 CGAN 的算法实现。

### 1.3 CGAN 的算法实现

- In each training iteration:

- Sample m positive examples  $\{(c^1, x^1), (c^2, x^2), \dots, (c^m, x^m)\}$  from database
- Sample m noise samples  $\{z^1, z^2, \dots, z^m\}$  from a distribution
- Obtaining generated data  $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$ ,  $\tilde{x}^i = G(c^i, z^i)$
- Sample m objects  $\{\hat{x}^1, \hat{x}^2, \dots, \hat{x}^m\}$  from database
- Update discriminator parameters  $\theta_d$  to maximize
 
$$\begin{aligned} & \tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(c^i, x^i) \\ & + \frac{1}{m} \sum_{i=1}^m \log (1 - D(c^i, \tilde{x}^i)) + \frac{1}{m} \sum_{i=1}^m \log (1 - D(c^i, \hat{x}^i)) \end{aligned}$$

$$\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$$
- Sample m noise samples  $\{z^1, z^2, \dots, z^m\}$  from a distribution
- Sample m conditions  $\{c^1, c^2, \dots, c^m\}$  from a database
- Update generator parameters  $\theta_g$  to maximize
 
$$\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log (D(G(c^i, z^i)))$$

$$\theta_g \leftarrow \theta_g - \eta \nabla \tilde{V}(\theta_g)$$

因为 CGAN 是有监督学习，采样的每一项都是文字和图片的 pair。CGAN 的核心就是判断什么样的 pair 给高分，什么样的 pair 给低分。

我们先关注判别器：

$$\begin{aligned} & \tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(c^i, x^i) \\ & + \frac{1}{m} \sum_{i=1}^m \log (1 - D(c^i, \tilde{x}^i)) \quad | \\ & + \frac{1}{m} \sum_{i=1}^m \log (1 - D(c^i, \hat{x}^i)) \quad 2 \quad 3 \end{aligned}$$

第一项是正确条件与真实图片的 pair，应该给高分；第二项是正确条件与伪造图片的 pair，应该给低分（于是加上了“1-”）；第三项是错误条件与真实图片的 pair，也应该给低分。

可以明显的看出，CGAN 与 GANs 在判别器上的不同之处就是多出了第三项。

下面再关注一下生成器：

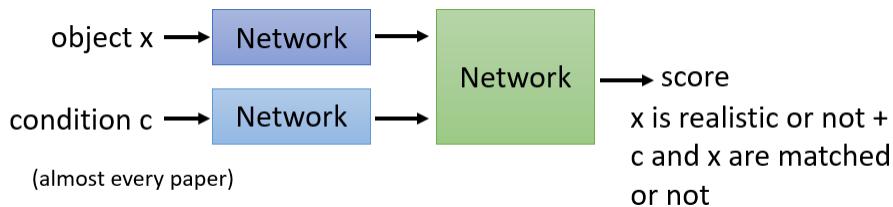
$$\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log (D(G(c^i, z^i)))$$

生成器的目的就是让判别器给伪造图片的得分越高越好，这与传统 GANs 本质上是一致的，只是在输入上多了一个参数 c。

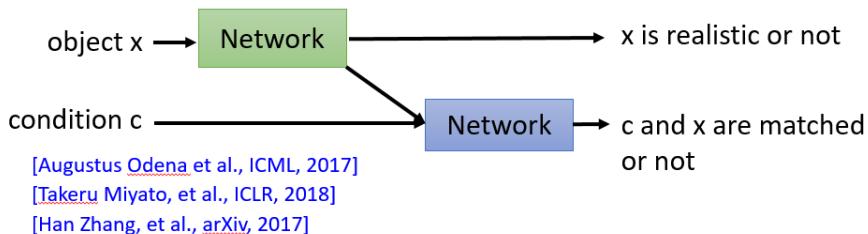
CGAN 的最终目标表达式写为：

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

## 1.4 CGAN 的讨论



大部分的 CGAN 判别器都采用上述架构，为了把图片和条件结合在一起，往往会把  $x$  丢入一个网络产生一个 embedding，condition 也丢入一个网络产生一个 embedding，然后把这两个 embedding 拼在一起丢入一个网络中，这个网络既要判断第一个 embedding 是否真实，同时也要判断两个 embedding 是否逻辑上匹配，最终给出一个分数。但是也有一种 CGAN 采用了另外一种架构，并且据李宏毅老师的介绍这种架构的效果是不错的。



首先有一个网络它只负责判断输入  $x$  是否是一个真实的图片，并且同时产生一个 embedding，与  $c$  一同传给第二个网络；然后第二个网络只需判断  $x$  和  $c$  是否匹配。最终两个网络的打分依据模型需求进行加权筛选即可。

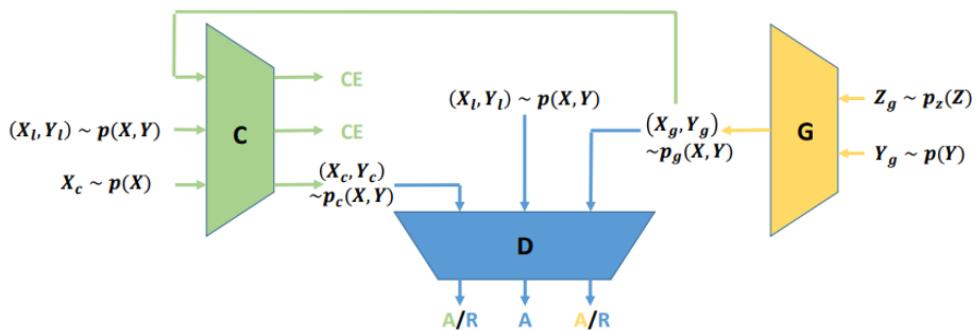
第二种模型有一个明显的好处就是判别器能区分出为什么这样的 pair 会得低分，它能反馈给生成器得低分的原因是  $c$  不匹配还是  $x$  不够真实；然而对第一种模型而言它只知道这样的 pair 得分低却不知道得分低的原因是什么，这会造成一种情况就是生成器产生的图片已经足够清晰了，但是因为不匹配  $c$  而得了低分，而生成器不知道得分低的原因是什么，依然以为是产生的图片不够清晰，那这样生成器就有可能朝着错误的方向迭代。不过，目前第一种模型还是被广泛应用的，其实事实上二者的差异在实际中也不是特别明显。

## 2. TripleGAN

### 2.1 TripleGAN 解决的问题

TripleGAN 是基于 CGAN 的改进，它主要想解决的问题是，在实际训练中，我们拥有的已配对的数据  $(c, x)$  往往是非常少量的，而人工标注配对数据  $(c, x)$  又比较麻烦，于是我们可以增添一个 classifier 去学习如何给图片  $x$  标注配对条件  $c$ ，这样就能形成比较好的训练数据。

### 2.2 TripleGAN 的模型架构



在上述架构图中， $x$  是图片， $y$  是条件(也就是  $c$ )， $(x, y)$  构成一个配对。从图中可以看出，TripleGAN 由三个部分组成，第一个是 Classifier，它负责学习并提供更多的配对信息给判别器，主要是从生成配对、真实配对和仅有图片三种输入中学会提取出它们的配对信息  $y_c$ ，并将这个配对信息  $y_c$  与图片  $x_c$  整合成一个新的配对  $(x_c, y_c)$  传递给判别器；而第二个部分判别器就需要学会鉴别输入的配对是来自真实数据，还是生成器，还是 classifier，最终在判别器的帮助下  $P_c, P_g$  都会越来越接近  $P_{data}$ ；至于第三个部分生成器，就与 CGAN 中的生成器一模一样了，输入一个条件  $y$  和先验分布  $z$ ，产生一个输出图片  $x$  和条件  $y$  的配对。

### 2.3 TripleGAN 的应用价值

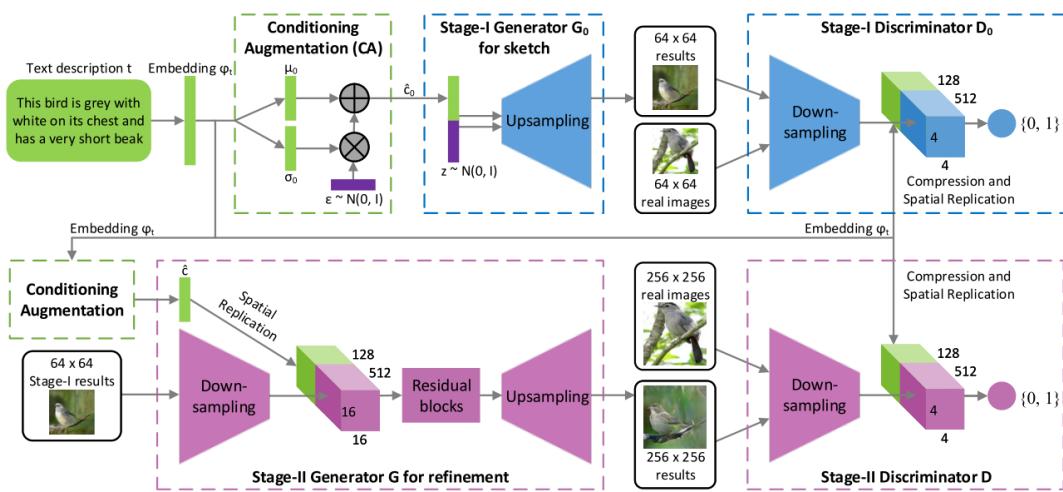
TripleGAN 最大的应用价值就是不仅生成器能够被提取出来，成为一个文字生成图片的模型，classifier 也能被提取出来，成为一个图片标注文字的模型。

### 3. StackGAN

#### 3.1 StackGAN 解决的问题

StackGAN 也是基于 CGAN 的改进，它主要想解决的问题是，CGAN 没有办法产生高清的大图，StackGAN 希望输入一个描述语  $c$ ，能够产生一张 256\*256 的清晰大图，核心思想就是搭建两个生成器，第一个产生一个 64\*64 的小图，然后把第一个生成器的结果放入第二个生成器中，在第二个生成器中产生 256\*256 的大图。

#### 3.2 StackGAN 的模型架构



StackGAN 模型主要分为两个阶段。如图所示，第一阶段的 StackGAN 就是一个标准的条件对抗生成网络 (Conditional GAN)，输入就是随机的标准正态分布采样的  $z$  和文本描述刻画的向量  $c$ 。第一步的对抗生成网络生成一个低分辨率的 64\*64 的图片和真实数据进行对抗训练得到粗粒度的生成模型。第二阶段的 StackGAN 将第一阶段的生成结果和文本描述作为输入，用第二个对抗生成网络生成高分辨率的 256\*256 的图片。

#### 第一阶段

由结构图可见，对于获得的 text\_embedding，stackGAN 没有直接将 embedding 作为 condition，而是用 embedding 接了一个 FC 层得到了一个正态分布的均值和方差，然后从这个正态分布中 采样出来要用的 condition ( $c_0$ ) 是

$$c_0 = \mu_0 + \sigma_0 \odot \epsilon$$

④意为逐元素乘积 (element\_wise multiply)。之所以这样做的原因是，embedding 通常比较高维(1024)，而相对这个维度来说，text 的数量其实很少，如果将 embedding 直接作为 condition，那么这个潜变量在潜空间里就比较稀疏，这对我们的训练不利。(实际上降了维，在处理后 1024 维降到了 128 维)。为了避免过拟合，生成器的 loss 里面加入了对这个分布的正则化：

$$D_{KL}(\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t)) || \mathcal{N}(0, I))$$

得到的  $c$  与服从标准正态分布的  $z$  连接起来，作为第一阶段生成器的输入。

生成器使用的并不是常用的反卷积，而是若干个上采样加保持大小不变的 3x3 的卷积组合，这是最近提出的一种避免反卷积棋盘效应的上采样方法。判别器是若干步长为 2 的卷积，再与 resize 的 embedding 合起来，接一个 FC。

第一阶段的输出是 64\*64 的低分辨率图像。

## 第二阶段

第二阶段的 生成器 并没有噪声输入，而是将第一阶段生成的低分辨率图像下采样以后与 replicated 的  $c_0$  连接起来作为输入。经过若干 residual blocks，再进行与第一阶段相同的上采样过程得到图片。

第二阶段的 判别器 与第一阶段大体相同。

两个阶段的损失函数分别如下：

### 第一阶段

$$\begin{aligned} \mathcal{L}_{D_0} = & \mathbb{E}_{(I_0, t) \sim p_{data}} [\log D_0(I_0, \varphi_t)] + \\ & \mathbb{E}_{z \sim p_z, t \sim p_{data}} [\log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))], \end{aligned} \quad (3)$$

$$\begin{aligned} \mathcal{L}_{G_0} = & \mathbb{E}_{z \sim p_z, t \sim p_{data}} [\log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))] + \\ & \lambda D_{KL}(\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t)) \parallel \mathcal{N}(0, I)), \end{aligned} \quad (4)$$

### 第二阶段

$$\begin{aligned} \mathcal{L}_D = & \mathbb{E}_{(I, t) \sim p_{data}} [\log D(I, \varphi_t)] + \\ & \mathbb{E}_{s_0 \sim p_{G_0}, t \sim p_{data}} [\log(1 - D(G(s_0, \hat{c}), \varphi_t))], \end{aligned} \quad (5)$$

$$\begin{aligned} \mathcal{L}_G = & \mathbb{E}_{s_0 \sim p_{G_0}, t \sim p_{data}} [\log(1 - D(G(s_0, \hat{c}), \varphi_t))] + \\ & \lambda D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t)) \parallel \mathcal{N}(0, I)), \end{aligned} \quad (6)$$

## 3.3 \*StackGAN 的改进

### 1.StackGAN++

StackGAN++是 ICCV 2017 的文章《StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks》。与前作 StackGAN 相比，StackGAN v2 有三点改进：

- 采用树状结构，多个生成器生成不同尺度的图像，每个尺度对应一个鉴别器。从而生成了多尺度 fake images。
- 除了 conditional loss，引入了 unconditional loss。即不使用条件信息，直接使用服从标准正态分布的噪声  $z$  生成 fake image 的损失。
- 引入了 color regulation，对生成的 fake images 的色彩信息加以限制。

效果是提高了训练的稳定性，且提高了生成的图像质量。

具体的模型架构就不作介绍了，感兴趣的读者可以前往论文地址处查阅：

<https://arxiv.org/abs/1710.10916>

### \*李飞飞团队 Text-To-Image Paper

在翻阅文本生成图像的相关工作，目前比较新的有突破性的工作是李飞飞工作团队 18 年 cvpr 发表的《Image Generation from Scene Graphs》。

不同于 StackGAN 存在比较突出的问题是不能处理比较复杂的文本，李飞飞小组提出的新方法能处理更长更复杂的文本，并且有不错的生成效果，感兴趣的读者可以前往论文地址处查阅：<https://arxiv.org/abs/1804.01622>。

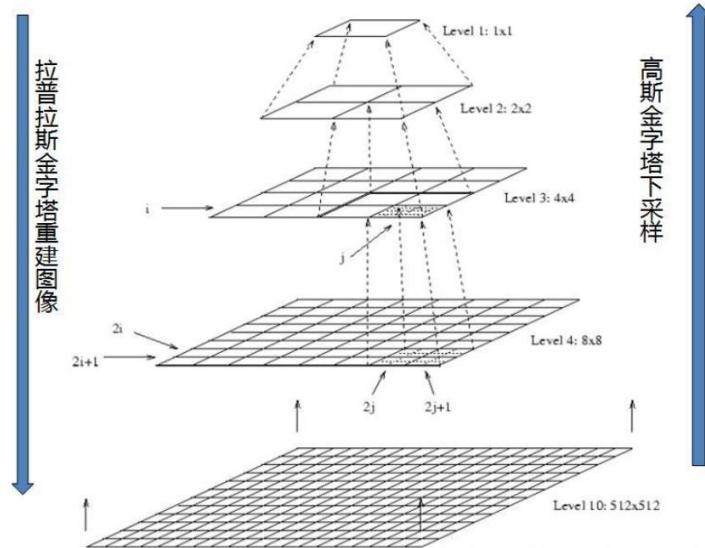
## 4. LapGAN

### 4.1 LapGAN 基本思路

如果我们希望生成高分辨率图像, 还有一种 GANs 可以考虑, 那就是 LapGAN。LapGAN 与 StackGAN 有着非常类似的思路, 都是通过先产生低分辨率图像再不断生成高分辨率图像, 但 LapGAN 是基于拉普拉斯金字塔实现的, 在金字塔的每一层都是学习与相邻层之间的残差, 也就是说, 高分辨率图像的生成是以低分辨率图像作为条件去生成残差, 然后低分辨率图上采样再跟残差求和得到高分辨率图, 这种低分辨率图向高分辨率图生成的过程其实就是一个 CGAN, 通过不断堆叠 CGAN 得到我们想要的分辨率。

### 4.2 拉普拉斯金字塔

我们先介绍一下什么是拉普拉斯金字塔:



简单来说, 拉普拉斯金字塔就是图片在尺度空间中不断上采样的结果 (从低分辨率层向高一层变化的过程叫上采样, 反之就称作下采样), 与之对应的, 高斯金字塔就是图片在尺度空间中不断下采样的结果。

下采样的过程, 分为两步:

- (1) 对图像进行高斯核卷积
- (2) 将偶数行除去。

用公式表示就是:

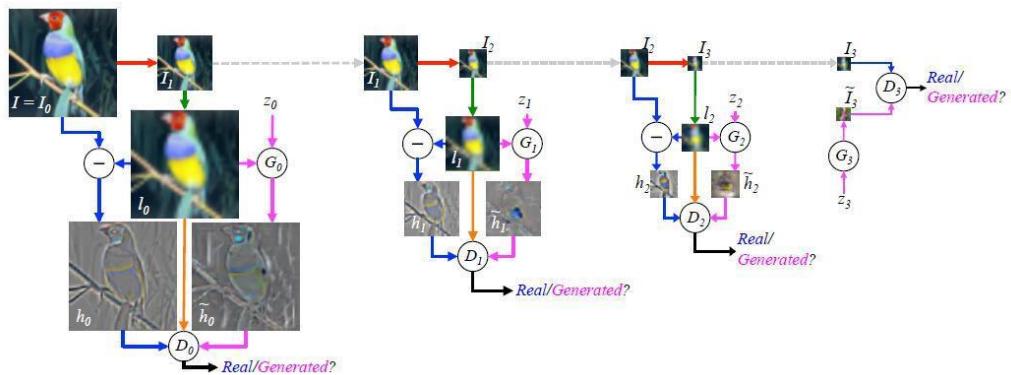
$$g_l(i, j) = \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) g_{l-1}(2i+m, 2j+n)$$

而上采样的过程, 是把当前层的残差值  $L_i$  和上一层图像上采样之后的结果加起来所得到的图像。其中残差值  $L_i$  的定义是:

$$L_i = G_i - \text{UP}(G_{i+1}) \otimes \mathcal{G}_{5 \times 5}$$

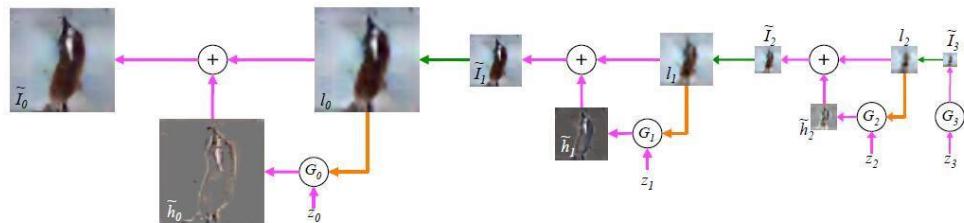
式中的  $G_i$  表示第  $i$  层的图像。而  $\text{UP}()$  操作是将源图像中位置为  $(x, y)$  的像素映射到目标图像的  $(2x+1, 2y+1)$  位置, 即在进行向上取样。符号  $\otimes$  表示卷积,  $\mathcal{G}_{5 \times 5}$  为  $5 \times 5$  的高斯内核。

### 4.3 LapGAN 的模型架构



上图是 LapGAN 搭建 4 层时的情形，一共用到了 4 个 GAN，其中  $I_0$  是  $64*64$  的图像， $I_1$  为  $32*32$ ， $I_2$  为  $16*16$ ， $I_3$  为  $8*8$ 。拉普拉斯金字塔的顶端（也就是像素最低的图像）用来训练普通的 GAN，生成器的输入只有噪声。而后像素更高的图像用来训练 CGAN，输入的不光有噪声，还有同级的高斯金字塔的图像经过上采样后得到的图像。

下面我们来看一下，对于一个训练好的 LapGAN，我们如何提取出它的生成器。



可以看出，这其实是由多个生成器级联出的一个模型。如果我们想在尺度空间  $k$  层下得到一张生成图片，我们获取该图片的公式便是：

$$\tilde{I}_k = u(\tilde{I}_{k+1}) + \tilde{h}_k = u(\tilde{I}_{k+1}) + G_k(z_k, u(\tilde{I}_{k+1}))$$

综上，LapGAN 就是依据拉普拉斯金字塔，把各个层之间的采样过程用 CGAN 实现，再把这些 CGAN 串联起来，形成了不断生成更高分辨率图片的一个结构。LapGAN 的一个核心优点是它让每个 GANs 只需要学会计算到不同层之间的残差值，大大减少了每一次 GANs 需要学习的内容，并且针对残差的逼近和学习也相对容易，从这一点上说与 Residual Network 有异曲同工之妙。但是 LapGAN 也存着一些弊端，最明显的一点就是 LapGAN 实质上的网络深度是非常深的，逐级独立训练提高了网络简单记忆输入样本的难度，许多高性能的深度网络都面临着这样的问题。

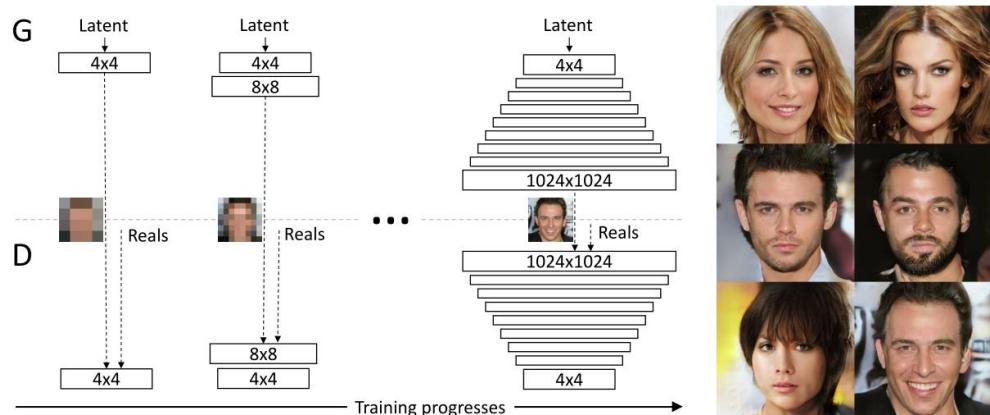
## 5. ProGAN (也称 PGGAN)

### 5.1 ProGAN 基本思路

如果现在我们想生成超高分辨率的图像，譬如  $1024 \times 1024$  图片，假设我们采用 StackGAN 或者是 LapGAN 的话，我们需要用到的 GANs 结构会非常多，这样会导致网络深度巨大，训练起来非常慢。为了解决这一问题，PGGAN（渐进式增长 GAN）提出的想法是，我们只需要一个 GANs 就能产生  $1024 \times 1024$  图片。但是一开始的时候 GANs 的网络非常浅，只能学习低分辨率 ( $4 \times 4$ ) 的图片生成，随着训练进行，我们会把 GANs 的网络层数逐渐加深，进而去学习更高分辨率的图片生成，最终不断的更新 GANs 从而能学习到  $1024 \times 1024$  分辨率的图片生成。

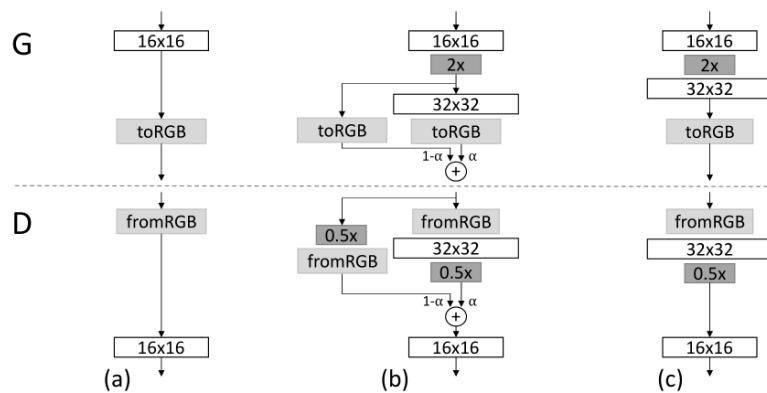
也就是说，PGGAN 与 StackGAN 和 LapGAN 的最大不同在于，后两者的网络结构是固定的，但是 PGGAN 随着训练进行网络会不断加深，网络结构是在不断改变的。这样做最大的好处就是，PGGAN 大部分的迭代都在较低分辨率下完成，训练速度比传统 GANs 提升了 2-6 倍。

### 5.2 ProGAN 的模型架构



训练开始于有着一个  $4 \times 4$  像素的低空间分辨率的生成器和判别器。随着训练的改善，我们逐渐向生成器和判别器网络中添加层，进而增加生成图片的空间分辨率。所有现存的层在过程中保持可训练性。这里  $N \times N$  是指卷积层在  $N \times N$  的空间分辨率上进行操作。这个方法使得在高分辨率上也能稳定合成并且加快了训练速度。右图我们展示了六张通过使用在  $1024 \times 1024$  空间分辨率上渐进增长的方法生成的样例图片。

但是上述这样的做法会有一个问题，就是从  $4 \times 4$  的输出变为  $8 \times 8$  的输出的过程中，网络层数的突变会造成 GANs 的急剧不稳定，使得 GANs 需要花费额外的时间从动荡状态收敛回平稳状态，这会影响模型训练的效率。为了解决这一问题，PGGAN 提出了平滑过渡技术。



如上图所示，当把生成器和判别器的分辨率加倍时，会平滑的增强新的层。我们以从  $16 \times 16$  像素的图片转换到  $32 \times 32$  像素的图片为例。在转换 (b) 过程中，把在更高分辨率上操作的层视为一个残缺块，权重  $\alpha$  从 0 到 1 线性增长。当  $\alpha$  为 0 的时候，相当于图 (a)，当  $\alpha$  为 1 的时候，相当于图(c)。所以，在转换过程中，生成样本的像素，是从  $16 \times 16$  到  $32 \times 32$  转换的。同理，对真实样本也做了类似的平滑过渡，也就是，在这个阶段的某个训练 batch，真实样本是:  $X = X_{16\text{pixel}} * (1-\alpha) + X_{32\text{pixel}} * \alpha$ 。

上图中的  $2\times$  和  $0.5\times$  指利用最近邻卷积和平均池化分别对图片分辨率加倍和折半。  
 $\text{toRGB}$  表示将一个层中的特征向量投射到 RGB 颜色空间中， $\text{fromRGB}$  正好是相反的过程；这两个过程都是利用  $1 \times 1$  卷积。当训练判别器时，插入下采样后的真实图片去匹配网络中的当前分辨率。在分辨率转换过程中，会在两张真实图片的分辨率之间插值，类似于将两个分辨率结合到一起用生成器输出。详细的过程可以参见 paper。

综上，便是 PGGAN 的主要思想，PGGAN 的主要优点就是能生成高质量的样本。日常生活中我们需要高清图片的时候还是蛮多的，因此我觉得 PGGAN 的应用价值还是挺大的。

## 6. StyleGAN

### 6.1 StyleGAN 解决的问题

我们先来反思一下上一节介绍的 ProGAN 有什么缺陷，由于 ProGAN 是逐级直接生成图片，我们没有对其增添控制，我们也就无法获知它在每一级上学到的特征是什么，这就导致了它控制所生成图像的特定特征的能力非常有限。换句话说，这些特性是互相关联的，因此尝试调整一下输入，即使是一点儿，通常也会同时影响多个特性。

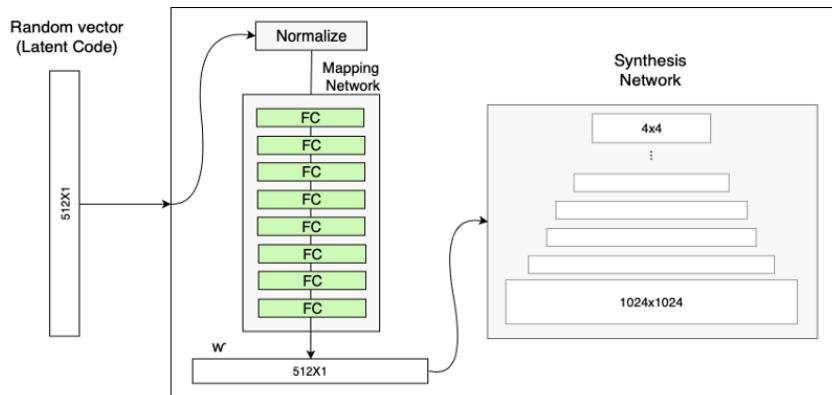
我们希望有一种更好的模型，能让我们控制住输出的图片是长什么样的，也就是在生成图片过程中每一级的特征，要能够特定决定生成图片某些方面的表象，并且相互间的影响尽可能小。于是，在 ProGAN 的基础上，StyleGAN 作出了进一步的改进与提升。

### 6.2. StyleGAN 模型架构

StyleGAN 首先重点关注了 ProGAN 的生成器网络，它发现，渐进层的一个潜在的好处是，如果使用得当，它们能够控制图像的不同视觉特征。层和分辨率越低，它所影响的特征就越粗糙。简要将这些特征分为三种类型：

- 1、粗糙的——分辨率不超过  $8^2$ ，影响姿势、一般发型、面部形状等；
  - 2、中等的——分辨率为  $16^2$  至  $32^2$ ，影响更精细的面部特征、发型、眼睛的睁开或是闭合等；
  - 3、高质的——分辨率为  $64^2$  到  $1024^2$ ，影响颜色（眼睛、头发和皮肤）和微观特征；
- 然后，StyleGAN 就在 ProGAN 的生成器的基础上增添了很多附加模块。

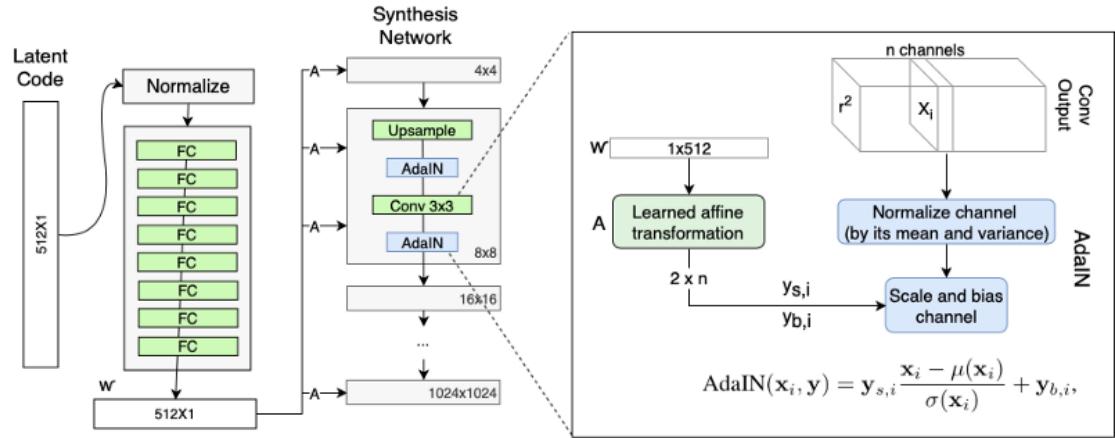
#### 1. 映射网络



映射网络的目标是将输入向量编码为中间向量，中间向量的不同元素控制不同的视觉特征。这是一个非常重要的过程，因为使用输入向量来控制视觉特征的能力是非常有限的，因为它必须遵循训练数据的概率密度。例如，如果黑头发的人的图像在数据集中更常见，那么更多的输入值将会被映射到该特征上。因此，该模型无法将部分输入（向量中的元素）映射到特征上，这一现象被称为特征纠缠。然而，通过使用另一个神经网络，该模型可以生成一个不必遵循训练数据分布的向量，并且可以减少特征之间的相关性。

映射网络由 8 个全连接层组成，它的输出  $w$  与输入层（ $512 \times 1$ ）的大小相同。

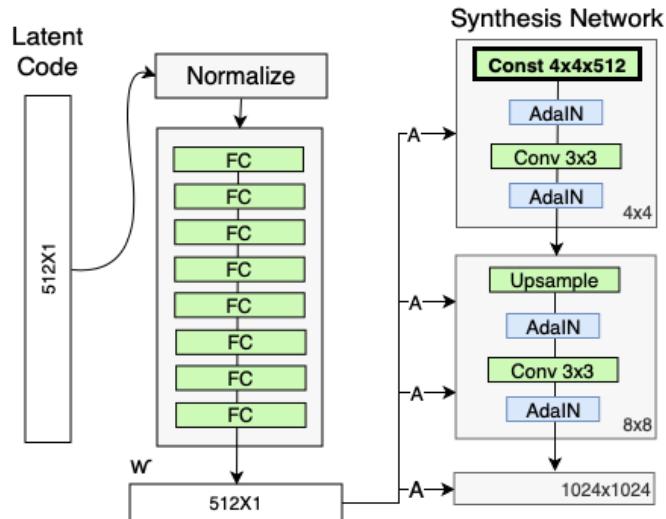
## 2. 样式模块 (AdaIN)



AdaIN (自适应实例标准化) 模块将映射网络创建的编码信息  $w$  传输到生成的图像中。该模块被添加到合成网络的每个分辨率级别中，并定义该级别中特征的可视化表达式：

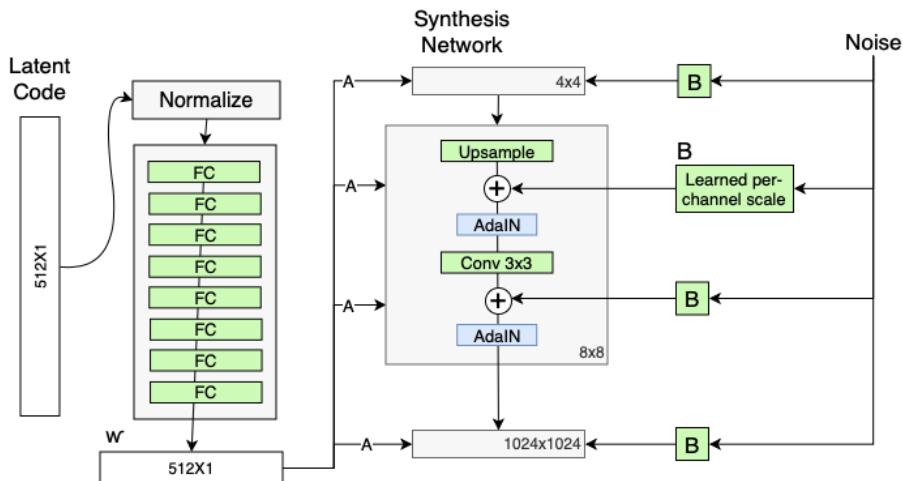
- 1、卷积层输出的每个通道首先进行标准化，以确保步骤 3 的缩放和切换具有预期的效果；
- 2、中间向量  $w$  使用另一个全连接的网络层 (标记为 A) 转换为每个通道的比例和偏差；
- 3、比例和偏差的向量切换卷积输出的每个通道，从而定义卷积中每个卷积核的重要性。这个调优操作将信息从  $w$  转换为可视的表达方式；

## 3. 删除传统输入



大多数的模型以及其中的 ProGAN 使用随机输入来创建生成器的初始图像 (即  $4 \times 4$  级别的输入)。StyleGAN 团队发现图像特征是由  $w$  和 AdaIN 控制的，因此可以忽略初始输入，并用常量值替代。虽然本文没有解释它为什么能提高性能，但一个保险的假设是它减少了特征纠缠，对于网络在只使用  $w$  而不依赖于纠缠输入向量的情况下更容易学习。

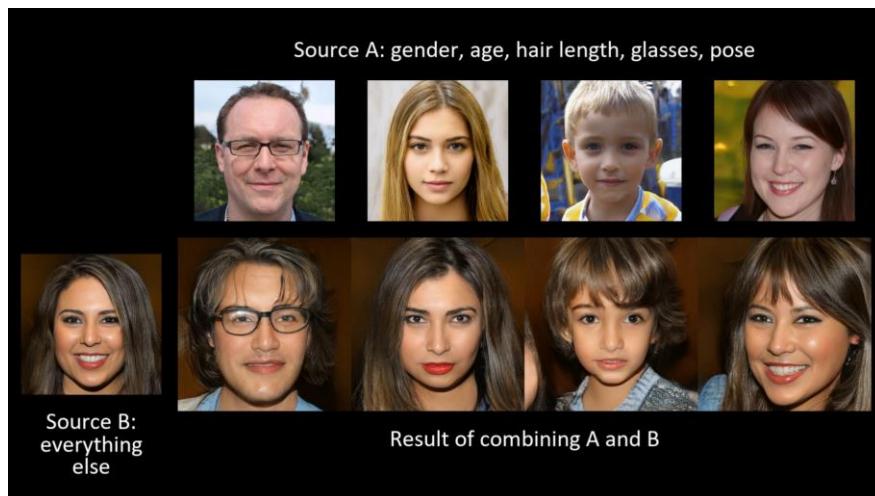
## 4. 随机变化



人们的脸上有许多小的特征，可以看作是随机的，例如：雀斑、发髻线的准确位置、皱纹、使图像更逼真的特征以及各种增加输出的变化。将这些小特征插入 GAN 图像的常用方法是在输入向量中添加随机噪声。然而，在许多情况下，由于上述特征的纠缠现象，控制噪声的影响是很复杂的，从而会导致图像的其它特征受到影响。

StyleGAN 中的噪声以类似于 AdaIN 机制的方式添加，在 AdaIN 模块之前向每个通道添加一个缩放过的噪声，并稍微改变其操作的分辨率级别特征的视觉表达方式。

## 5. 样式混合



StyleGAN 生成器在合成网络的每个级别中使用了中间向量，这有可能导致网络学习到这些级别是相关的。为了降低相关性，模型随机选择两个输入向量，并为它们生成了中间向量  $w$ 。然后，它用第一个输入向量来训练一些网络级别，然后（在一个随机点中）切换到另一个输入向量来训练其余的级别。随机的切换确保了网络不会学习并依赖于一个合成网络级别之间的相关性。

虽然它并不会提高所有数据集上的模型性能，但是这个概念有一个非常有趣的副作用 — 它能够以一种连贯的方式来组合多个图像（视频请查看原文）。该模型生成了两个图像 A 和 B，然后通过从 A 中提取低级别的特征并从 B 中提取其余特征再组合这两个图像。

## 6. 在 W 中的截取技巧



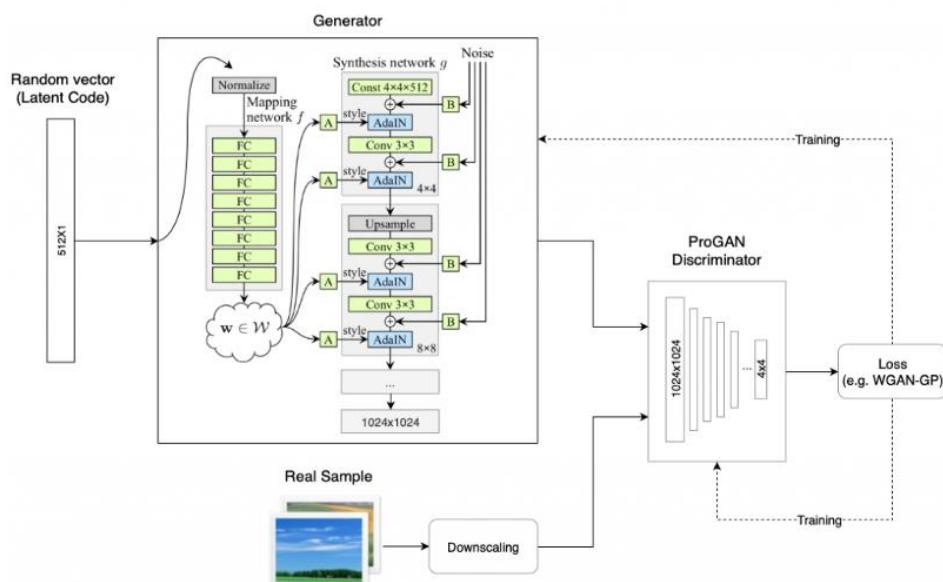
在生成模型中的一个挑战，是处理在训练数据中表现不佳的地方。这导致了生成器无法学习和创建与它们类似的图像（相反，它会创建效果不好的图像）。为了避免生成较差的图像，StyleGAN 截断了中间向量  $w$ ，迫使它保持接近“平均”的中间向量（上图左 4）。

对模型进行训练之后，通过选择多个随机的输入，用映射网络生成它们的中间向量，并计算这些向量的平均值，从而生成“平均”的平均值  $w$ 。当生成新的图像时，不用直接使用映射网络的输出，而是将值  $w$  转换为  $w_{\text{new}} = w_{\text{avg}} + \Psi(w - w_{\text{avg}})$ ，其中  $\Psi$  的值定义了图像与“平均”图像的差异量（以及输出的多样性）。有趣的是，在仿射转换块之前，通过对每个级别使用不同的  $\Psi$ ，模型可以控制每个特征集与平均值的差异量。

## 7. 微调超参数

StyleGAN 的另外一个改进措施是更新几个网络超参数，例如训练持续时间和损失函数，并将离得最近的放大或缩小尺度替换为双线性采样。

综上，加入了一系列附加模块后得到的 StyleGAN 最终网络模型结构图如下：

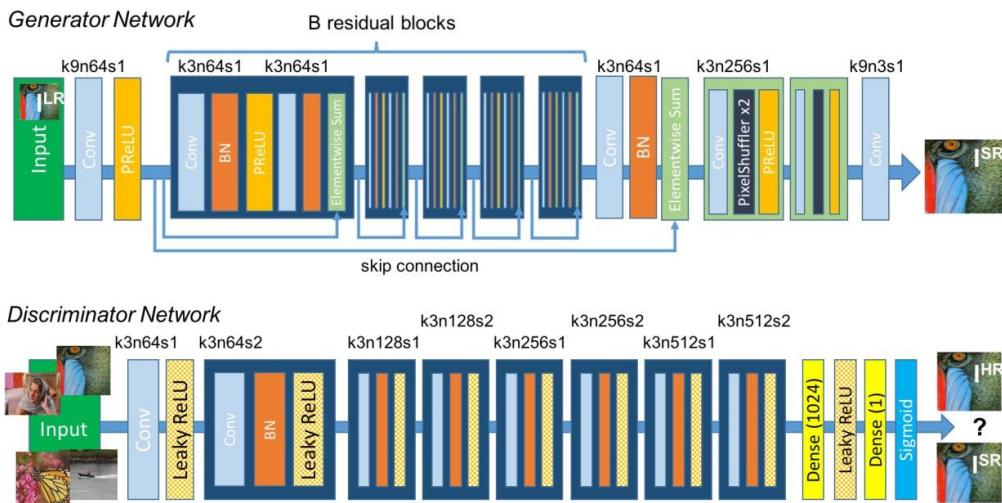


上述就是 StyleGAN 的完整模型的介绍了。不得不说，不论是在理论方法上，还是工程实践上，StyleGAN 都是一篇具有突破性的论文，它不仅可以生成高质量的和逼真的图像，而且还可以对生成的图像进行较好的控制和理解。

## 7.\* SRGAN

SRGAN (Super-Resolution GAN) 顾名思义,就是一个专门做图像超分辨率的 GANs。核心思路非常简单, G 网通过低分辨率的图像生成高分辨率图像,由 D 判断拿到的图像是由 G 网生成的,还是数据库中的原图像。当 G 网能成功骗过 D 网的时候,那这个 GANs 就可以完成超分辨率了。

网络架构如下:



具体的网络参数设置和损失函数的定义读者可以翻阅 paper 研究,因为没有什么亮点和创新点,在此就不多作介绍了。

至此, GANs 在图像生成上的介绍到此就告一段落了,可以看出 GANs 在实现高质量图像生成上的 paper 非常多,是 GANs 的主力军,但其实 GANs 不光可以实现高质量图像生成,它还能实现在不同图像之间的转化以及通过提取特征定向生成图片等,我们会在接下来的部分中进行介绍。

## Part2 GANs 在风格迁移上的应用

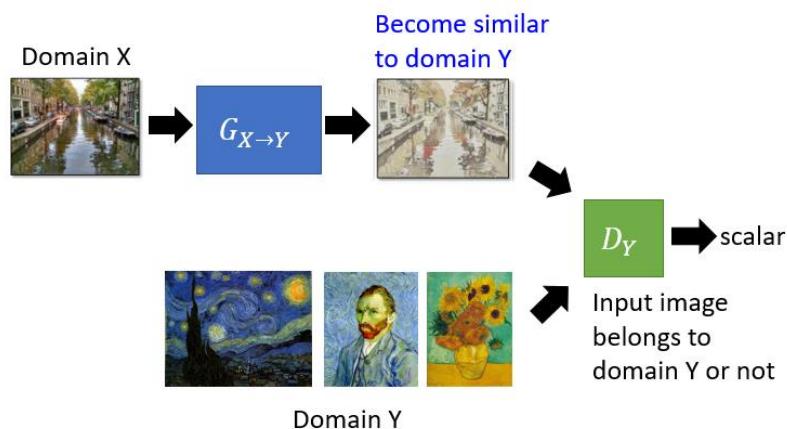
### 1. CycleGAN

#### 1.1 CycleGAN 解决的问题

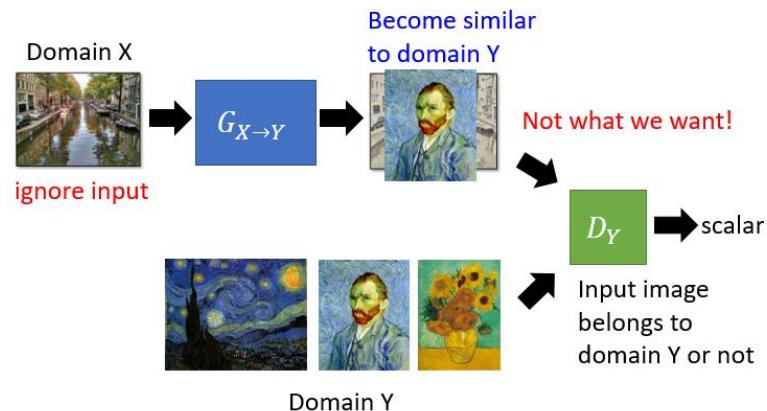
假设我们现在要训练一个风格迁移的神经网络，也就是说输入一张图片，输出一张它不同的风格的图片，比如说输出一张具有泛古抽象质感的图片。



那么我们考虑应用 GANs 技术。一个很自然的想法是给它增添一个判别器，这个判别器用来判别输入的图像是真实的还是 G 伪造的。



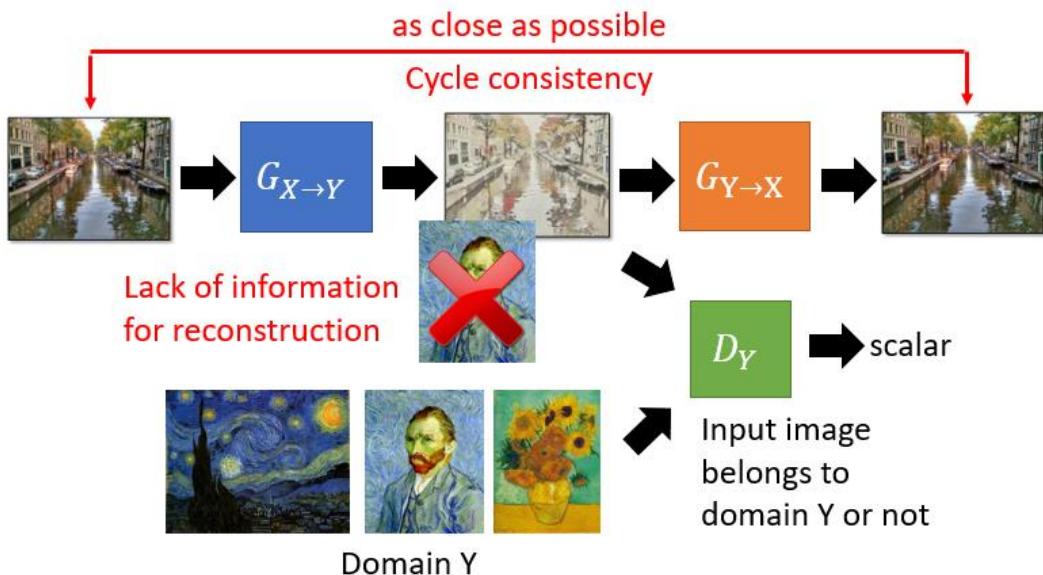
这个架构看似合理，但是会有一些潜在的危险。在生成器很深时，它的输出和输入差别是可能非常大的，存在一种情况是当输出图像靠近真实分布 Y 里的某一张图像时，生成器就发现了一个 BUG，只要它的输出越逼近这张真实图像，判别器给的评分就越高，于是生成器最终可以完全忽略输入长什么样，输出这张偷学到的真实图片，就能产生“高质量”图片。



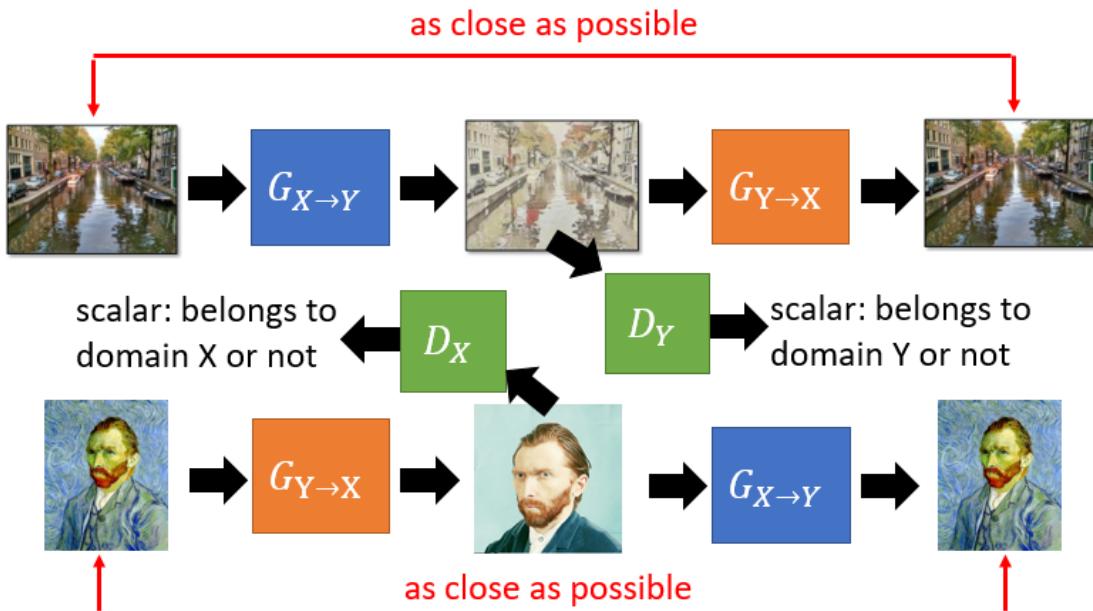
为了消除这种潜在危险，CycleGAN 诞生了。

## 1.2 CycleGAN 的原理

为了防止生成器学习到具有欺骗性的造假数据, 我们只需要保证生成器的输出和原图具有很高的相似性, 也就是不丢失原图的特征, 于是 CycleGAN 中加入了一个新的生成器, 把第一个生成器的输出当作输入丢进去, 希望能输出一个和原始输入尽可能相似的图片, 如果能够比较好的还原回原始图片, 证明第一个生成器的输出保留了大量原始图片的特征, 输出结果是较为可靠的; 而如果不能较好的还原回原始图片, 意味着第一个生成器可能使用了“造假”的输出结果。

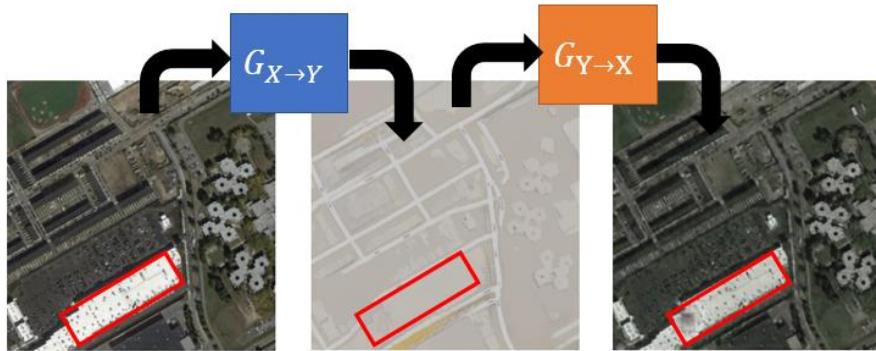


那 CycleGAN 还可以做成双向的, 除了从 X domain--(G1)-->Y domain--(G2)-->X domain 的训练, 同时还会有 Y domain--(G2)-->X domain--(G1)-->Y domain 这样的训练, 在第二种训练中会新引入一个判别器, 功能同样是保证整次训练的输入和输出尽可能相似。



### 1.3 CycleGAN 的讨论

CycleGAN 也不是没有问题。CycleGAN: a Master of Steganography (隐写术) [Casey Chu, et al., NIPS workshop, 2017]这篇论文就指出，CycleGAN 存在一种情况，是它能学会把输入的某些部分藏起来，然后在输出的时候再还原回来。比如下面这张图：



The information is hidden.

可以看到，在经过第一个生成器的时候，屋顶的黑色斑点不见了，但是在经过第二个生成器之后，屋顶的黑色斑点又被还原回来了。这其实意味着，第一个生成器并没有遗失掉屋顶有黑色斑点这一讯息，它只是用一种人眼看不出的方式将这一讯息隐藏在输出的图片中（例如黑点数值改得非常小），而第二个生成器在训练过程中也学习到了提取这种隐藏讯息的方式。那生成器隐藏讯息的目的是什么呢？其实很简单，隐藏掉一些破坏风格相似性的“坏点”会更容易获得判别器的高分，而从判别器那拿高分是生成器实际上的唯一目的。

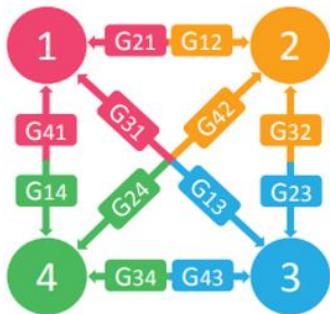
综上，CycleGAN 所宣称的 CycleConsistency 其实是不一定能完全保持的，毕竟生成器的学习能力非常强大，即便人为地赋予它诸多限制，它也有可能学到一些 trick 去产生一些其实并不太符合人们要求的输出结果。

## 2. StarGAN

### 2.1 StarGAN 解决的问题

有的时候我们可能希望图片能在 $n$ 个domain当中互转, 那依据CycleGAN的设计思路, 理论上我们需要训练 $2 * C_n^2$ 个生成器, 如下图所示:

(a) Cross-domain models

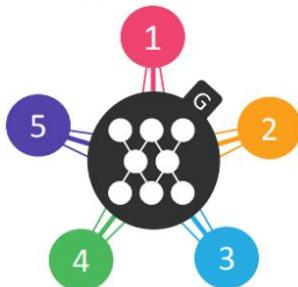


很明显这需要训练的生成器太多了。那为了用更少的生成器实现多个风格之间的互转, StarGAN 被提出了。

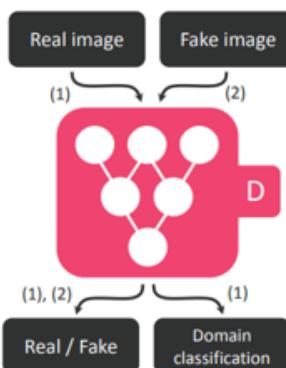
### 2.2 StarGAN 的原理

StarGAN 在设计的时候就希望只用一个生成器去实现所有 domain 之间的互转。

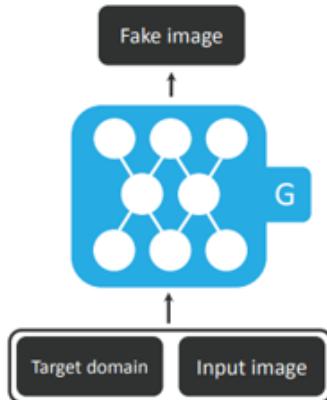
(b) StarGAN



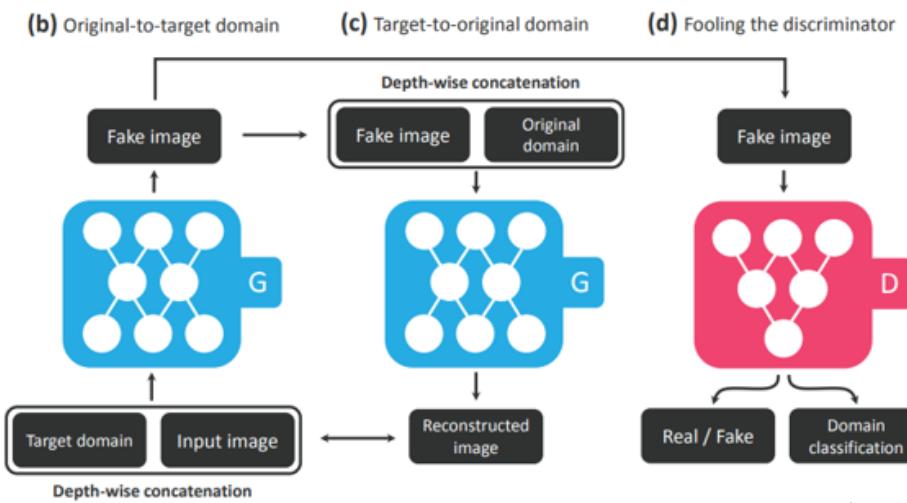
对于判别器, 它的输入是一张图片, 它需要去鉴别出这张图片是真实图片还是生成器产生的图片, 同时它还需要分辨出这张输入的图片来自于哪个 domain(哪种风格)。



对于生成器，它有两个输入，分别是目标 domain 和输入图片，它的输出是一张符合目标 domain 风格的图片。



整个训练架构如下图所示：



首先目标 domain 和输入图片会被输入到生成器当中，然后生成器产生的伪造图片一方面会被传给判别器（右图），判别器会判别这张图片的真假以及 domain 分类是哪，另一方面这个伪造图片会被再次传回给这个生成器（中图），不过目标 domain 改成了原始的来源 domain，那模型会希望第二次输出的图片能和最开始的输入图片尽可能相似。其实整套训练流程与 CycleGAN 是非常相似的，不同之处在于 CycleGAN 使用了两个生成器做风格的来回变换，而在 StarGAN 中仅使用了一个生成器实现这一变换。

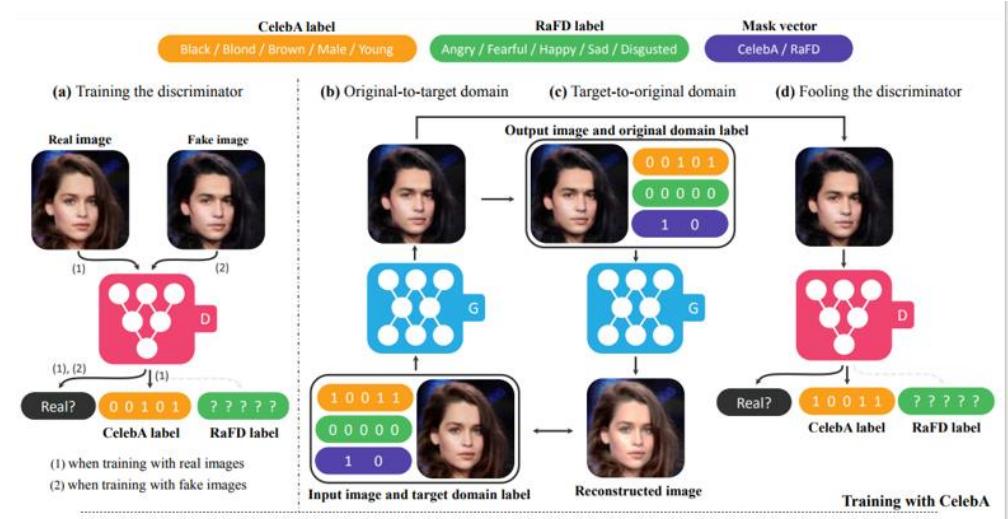
## 2.3 StarGAN 的实现举例

我们用一个实例来说明 StarGAN 的具体训练方式。

首先 domain 的定义在 StarGAN 中是用一个 vector 表示的，譬如：

CelebA label	RaFD label	Mask vector
Black / Blond / Brown / Male / Young	Angry / Fearful / Happy / Sad / Disgusted	CelebA / RaFD

那如果一个 domain 表示为 00101 00000 10, 就意味着这是一个 CelebA 的 label, 并且这个 domain 的风格是棕发年轻女性, 如果希望要转移到的风格是黑发年轻男性, 那么训练过程如下图所示:

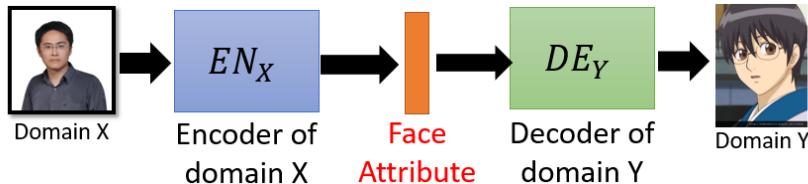


首先原始图片被丢入这个生成器中, target domain 是 10011 (黑发年轻男性), 然后生成器就会产生一个黑发年轻男性的图片, 接着这张图片又会再次被丢进这个生成器中, 但是 target domain 改成了 00101 (棕发年轻女性), 然后生成器就会产生一个棕发年轻女性的图片, StarGAN 希望这张图片和原始的输入的棕发年轻女性的图片尽可能相似, 此处用到的 loss 就是 CycleGAN 当中的 loss。另外一方面, 第一次生成器产生的黑发年轻男性图会传给判别器, 判别器一方面要判断这张图的真假, 另外一方面还要判断出这张图的 domain, 如果判断结果是 10011 的话就证明这张图的效果是好的。

综上就是 StarGAN 的实现过程, 它能实现多种风格之间的互转, 并且模型比较小巧且高效。

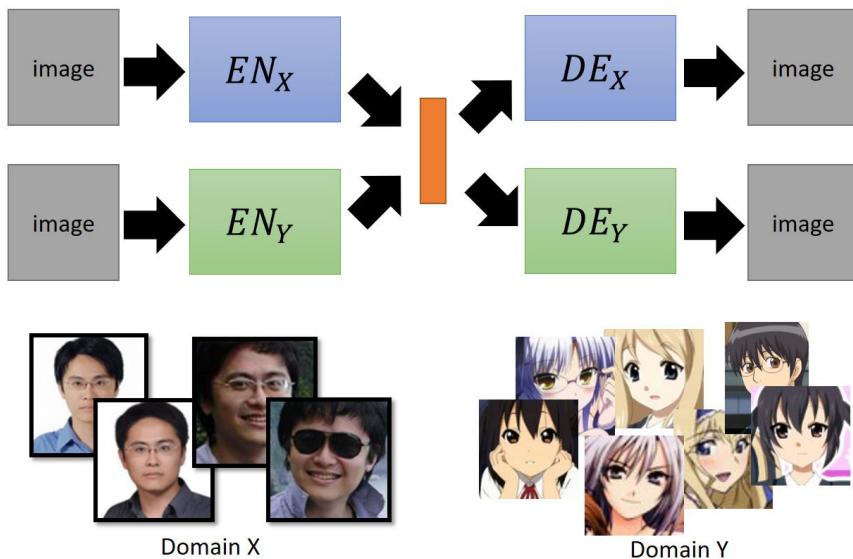
### 3. 如何解决 BIG TRANSFORMATION

我们前面提到的风格转换都是比较小幅度的风格转换,有的时候我们会需要做非常大的风格转换,譬如把真人照片转成动漫照片。大风格的转换我们容易想到的一个思路是 auto-encoder:



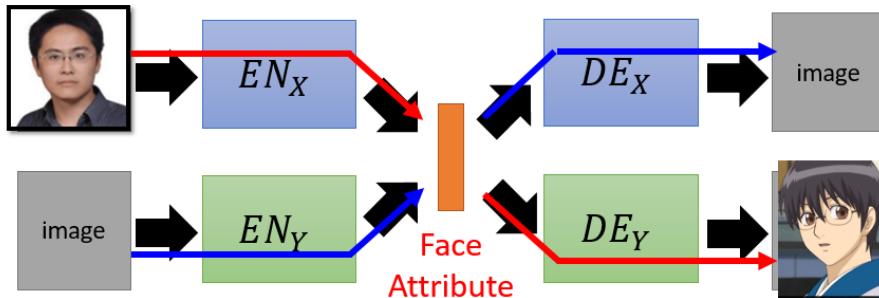
也就是说,输入一个真人的照片,通过一个 encoder 提取出它的人脸特征,再放入到一个 decoder 中输出一张动漫人物的照片,这两张照片的差异是可以非常大的。

现在我们进一步希望,  $x$  domain 和  $y$  domain 的照片能够相互转换,于是我们再增添一个 auto-encoder:

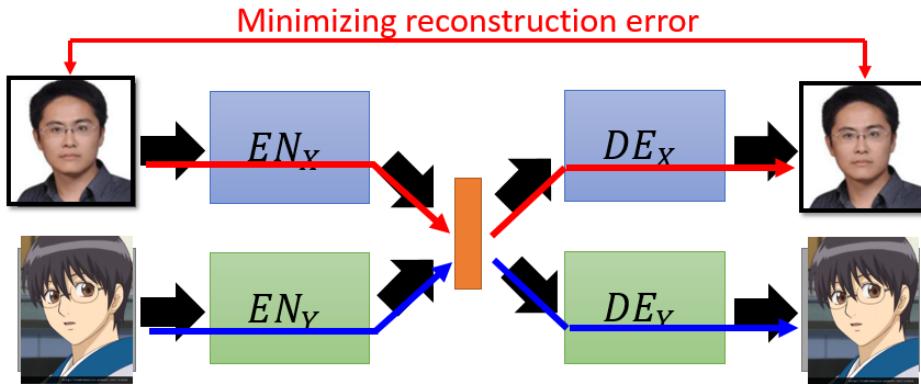


这意味着,输入  $x$  domain 的图片就放入  $x$  encoder 中, 输入  $y$  domain 的图片就放入  $y$  encoder 中,它们都能提取出对应的人脸特征,然后如果需要输出  $x$  domain 的图片就把特征放入  $x$  decoder 中,如果需要输出  $y$  domain 的图片就把特征放入  $y$  decoder 中,这样就能实现在两个 domain 间的相互转换。

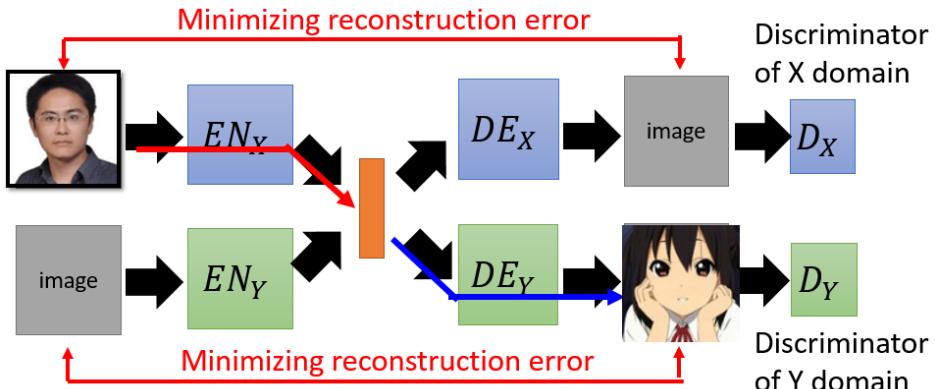
当然上述我们希望这个模型具有的功能是在  $x$  domain 和  $y$  domain 之间互换,也就是图片的训练路径是  $image_x \rightarrow encoder_x \rightarrow decoder_y \rightarrow image_y$  和  $image_y \rightarrow encoder_y \rightarrow decoder_x \rightarrow image_x$ 。



那如果仅仅是按照这样的路径训练的话,  $encoder_x$ 与 $encoder_y$ 完全没有关系,  $decoder_x$ 与 $decoder_y$ 也完全没有关系, 事实上它就等价于两个单独训练的 auto-encoder。为了改善这一模型, 我们可以增添两条训练路径, 如下图所示:



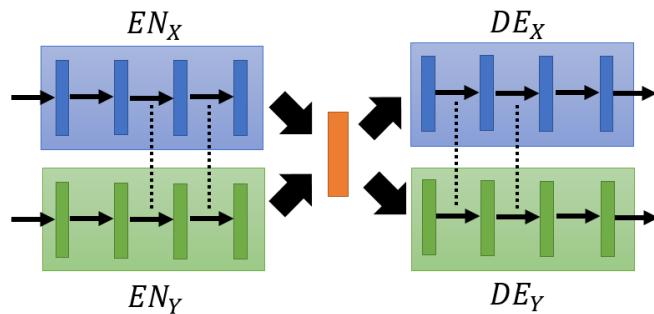
在这两种路径下, 一张图片会进入同一个 domain 的 encoder 和 decoder, 我们希望最终输出的图片能和原始图片越相似越好。但是仅仅只有 minimize reconstruction 是不够的, 我们还希望输出的图片尽可能清晰, 于是分别加入一个判别器。



这样在这种训练路径下, 模型产生的图片就既和原图比较相似也有着尽可能高的清晰度。但是现在还需要考虑一个新的问题, 就是模型目前只能保证一张图片  $image_x$  走  $encoder_x$  和  $decoder_x$  这条路, 或者一张图片  $image_y$  走  $encoder_y$  和  $decoder_y$  这条路, 能够产生和原图高度相似 (这儿的相似是要求像素无差) 的图片, 但是模型不能保证, 一张图片  $image_x$  走  $encoder_x$  和  $decoder_y$  这条路产生的图片和原图  $image_x$  是相似的 (这儿的相似是要求看着像同一个人, 通常输出会很清楚但明显不是同一个人)。这是因为, domain x 和 domain y 的差异很大,  $encoder_x$  与  $encoder_y$  学到的特征提取方法也会不一样, 譬如  $encoder_x$  用第 1,2 维提取头发和眼睛信息, 但是  $encoder_y$  用 3,4 维提取头发和眼镜信息, 这样  $decoder_y$  在解  $encoder_x$  提取的特征信息时就会提取出错误的特征信息。那在文献上为了解决这个问题, 出现了各式各样的解法, 下面会一一介绍。

### 3.1 CoGAN

CoGAN(又称 CoupledGAN)采用了一个最直接的解决思路,既然错误的出现是encoder<sub>x</sub>和encoder<sub>y</sub>的差异导致的,那我何不让encoder<sub>x</sub>和encoder<sub>y</sub>尽可能地相似?具体来说,就是让encoder<sub>x</sub>和encoder<sub>y</sub>的输出尽可能地相似。那为了让二者输出尽可能相似, CoupleGAN便让encoder<sub>x</sub>和encoder<sub>y</sub>的最后几层的参数是共享的,也就是encoder<sub>x</sub>和encoder<sub>y</sub>的最后几层hidden layer 是完全一样的,这样它们提取出的脸部特征(下称 face attribution)在结构上就非常相似。同理,既然 face attribution 差不多了, decoder<sub>x</sub>和decoder<sub>y</sub>的前面几层参数也是共享的。

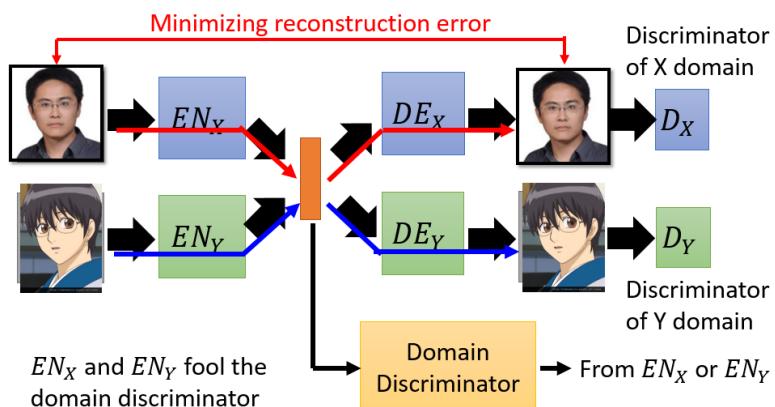


Sharing the parameters of encoders and decoders

CoGAN 有一种极端的情况,就是encoder<sub>x</sub>和encoder<sub>y</sub>的所有参数都共享,只是额外的增加一个输入变量 0 或 1.1 表示输入来自 x domain, 0 表示输入来自 y domain。在这种情况下, encoder<sub>x</sub>和encoder<sub>y</sub>可以合并为一个 encoder。

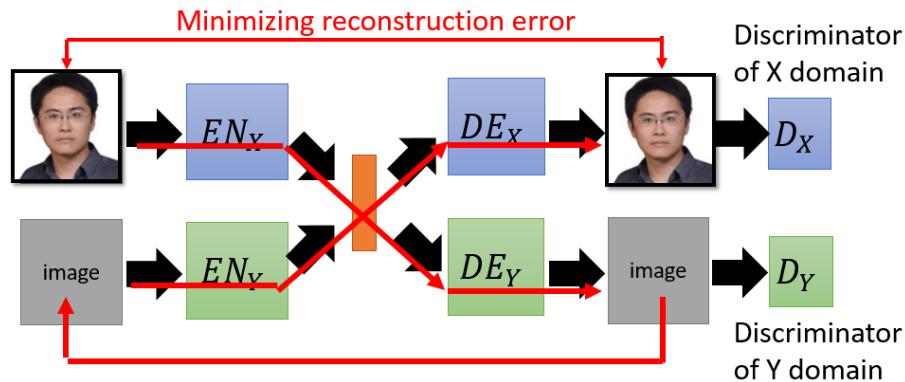
### 3.2 Guillaume Lample

我们的目的,实质上是希望encoder<sub>x</sub>和encoder<sub>y</sub>产生的 face attribution 尽可能一致,那Guillaume Lample 方法就是给 face attribution 增添一个判别器,去判断这个 face attribution 是来自于 x domain 还是 y domain, 那encoder<sub>x</sub>和encoder<sub>y</sub>为了骗过这个判别器,就会产生尽可能一致的 face attribution,这样判别器就无法判断出encoder<sub>x</sub>和encoder<sub>y</sub>输出结果的差异。



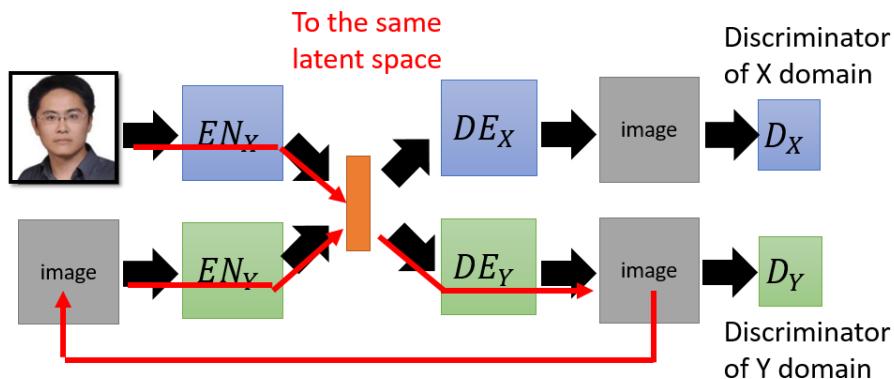
### 3.3 ComboGAN

实际上, 从encoder<sub>x</sub>和decoder<sub>y</sub>产生的图片或者从encoder<sub>y</sub>和decoder<sub>x</sub>产生的图片无法和原图相似的本质原因, 是特征信息的丢失。那我们现在把两条路拼在一起, 形成一个image<sub>x</sub>->encoder<sub>x</sub>->decoder<sub>y</sub>->image<sub>y</sub>->encoder<sub>y</sub>->decoder<sub>x</sub>->image<sub>x</sub>'的大回路, 如果我们增添一个 reconstruction loss, 要求image<sub>x</sub>和image<sub>x</sub>'尽可能地相似, 这样就会逼迫着encoder<sub>x</sub>->decoder<sub>y</sub>和encoder<sub>y</sub>->decoder<sub>x</sub>这两段路尽可能的不丢失原图的特征信息, 最终就能解决图片不相似的问题。这个方法就是利用了 Cycle consistency 的 ComboGAN。



### 3.4 XGAN

不同于 ComboGAN 利用的是 Cycle consistency, XGAN 利用的是 Semantic consistency (语义保持), 也就是说不同于 ComboGAN 希望图像到图像能保持一致, XGAN 希望的是特征到特征能保持一致。这样在计算相似性的时候不再是用 pixel-wise 的计算, 而是 latent 相似性的计算。



如上图所示, 在image<sub>x</sub>->encoder<sub>x</sub>之后会得到一个latent<sub>1</sub>, 然后继续走半个循环得到latent<sub>2</sub>, 即latent<sub>1</sub>->decoder<sub>y</sub>->image<sub>y</sub>->encoder<sub>y</sub>->latent<sub>2</sub>, XGAN 便是希望latent<sub>1</sub>与latent<sub>2</sub>能够越接近越好。

## Part3 GANs 在特征提取上的应用

### 1. InfoGAN

#### 1.1 InfoGAN 解决的问题

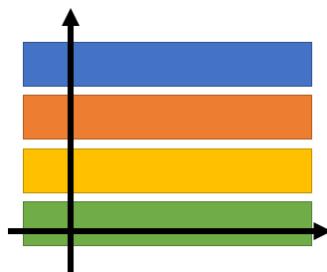
我们知道 GANs 能实现从先验分布到目标分布的生成过程，通常 GANs 的输入是先验分布的随机抽样，这意味着 GANs 的输出也是随机的。现在我们不想让 GANs 随机生成图片，我们希望通过控制输入的参数去生成特定的图片。我们以手写数字图片的生成为例，我们希望调整输入的参数，从而能控制输出的数字、高度以及形状等。

首先得研究输入的每一个维度对输出的影响，先看下面这张图：

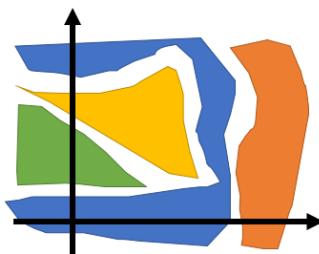


这是我们在传统 GANs 上面做的实验，调整输入向量某一维度上的取值，得到了上面的输出。我们发现，随着输入的逐渐变化，输出的结果是变来变去且非常跳跃的，比如一开始输出 7，随着调整又变成输出 0，接着又变成输出 7，似乎非常没有规律可言。

为什么会出现这样的情况？为了直观地解释，我们假设输入的向量只有二维，我们本来希望输入向量对应的特征分布是下面这样的：



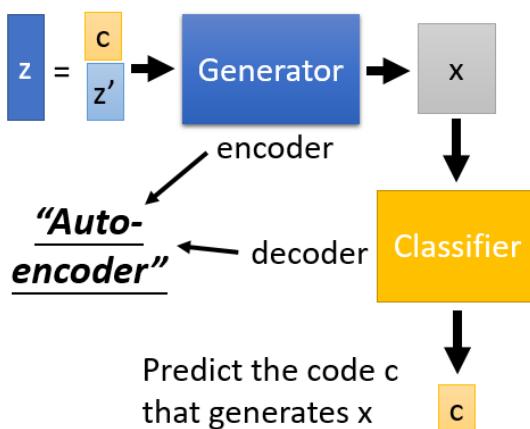
因为如果是这样分布的话，改变一个参数对应输出图像的变化是非常规律的，我们就能非常有效地控制输出。但是实际上，因为 GANs 强大的拟合能力，它学习到的输入向量对应的特征分布可以非常复杂，就像下图这样：



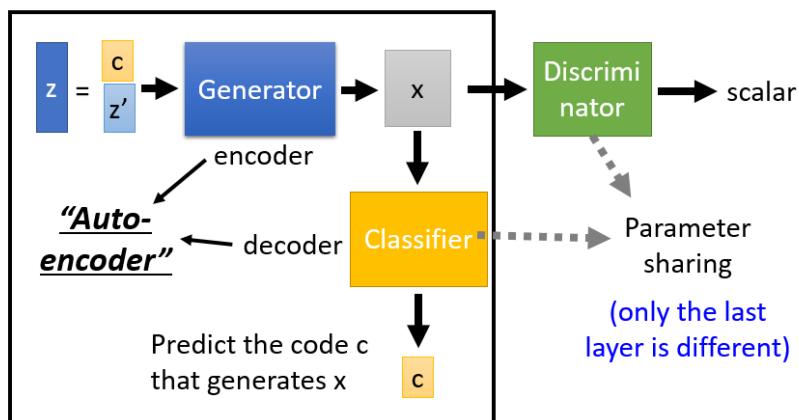
在这种情况下，输出图片的某个特征是由输入的很多维控制的，我们想要通过调整输入去控制这些特征就非常困难。为了解决这种问题，InfoGAN 被提出了。

## 1.2 InfoGAN 的原理

传统 GANs 的问题就是输入的向量对于输出的影响不明确, 那现在我们人为地要求输入的向量对于输出的影响必须明确。InfoGAN 的做法就是, 把输入的向量  $z$  拆分成子向量  $c$  和子向量  $z'$ , 其中子向量  $c$  就是明确地要对输出产生影响的部分。换言之, 所有的输出都要受到  $c$  的影响, 也就是所有的输出都含有  $c$  的信息, 这等价于所有的输出都能通过一个 classifier 提取出原始的  $c$  的信息, 如下图所示。



上图的这个结构, 类似于一个 auto-encoder 结构, 它保证了所有的输出  $x$  都直接受到向量  $c$  的影响。但是这样的结构, 不能够保证输出图像  $x$  的真实度, 因此必须还得增加一个判别器。这个判别器同时也能防止生成器的作弊行为: 直接原封不动地把向量  $c$  放在输出  $x$  当中, 这样 classifier 就能直接提取出信息  $c$ ; 因此增添判别器是必要的。最终 InfoGAN 的结构图如下图所示:



其中 classifier 与判别器因为是同一个输入, 最初的几层 layer 可以参数共享。另外值得一提的是, 这个向量  $c$  是一个很神奇的存在, 它的维数完全由人自己定, 这取决于我们希望最终图片  $x$  被划分为多少类, 那我们就用多少维的参数去决定。但是被划分的每一类具体长什么样, 这完全是由生成器和 classifier 共同学习的, 不过我们能够确定的是, 最终得到的分类, 一定是在特征差异最大的地方做的划分。

为了理解向量  $c$  的作用, 我们看一下 InfoGAN 在手写数字图片生成上的实验结果。

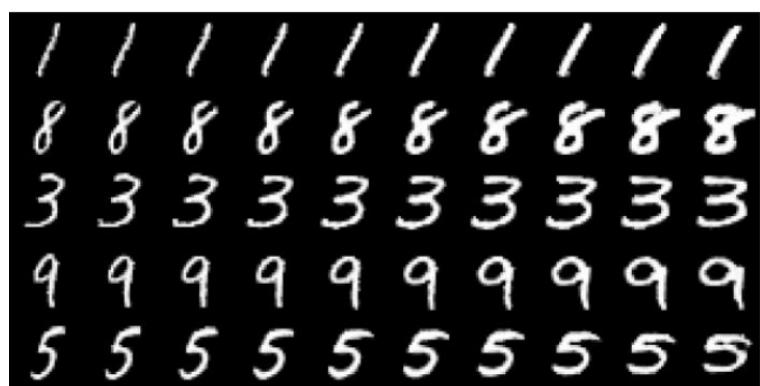
### 1.3 InfoGAN 的实验结果



这是改变向量  $c$  的第 1 维产生的生成结果，我们发现，第 1 维的功能就是直接决定生成的数字；同时也符合，第一维是在特征差异最大的地方做划分的结果



这是改变向量  $c$  的第 2 维产生的生成结果，第 2 维的功能是决定生成数字的倾斜角度。



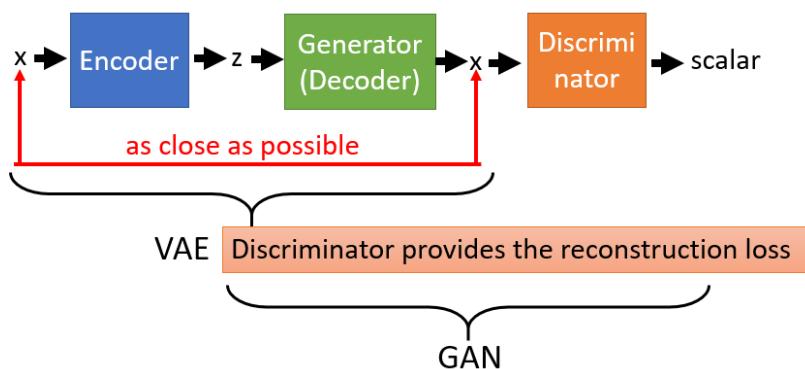
这是改变向量  $c$  的第 3 维产生的生成结果，第 3 维的功能是决定生成数字的粗细程度。

综上，InfoGAN 确实可以对生成结果作出稳定的划分，并且能通过控制输入去产生特定的输出结果。

## 2. VAEGAN

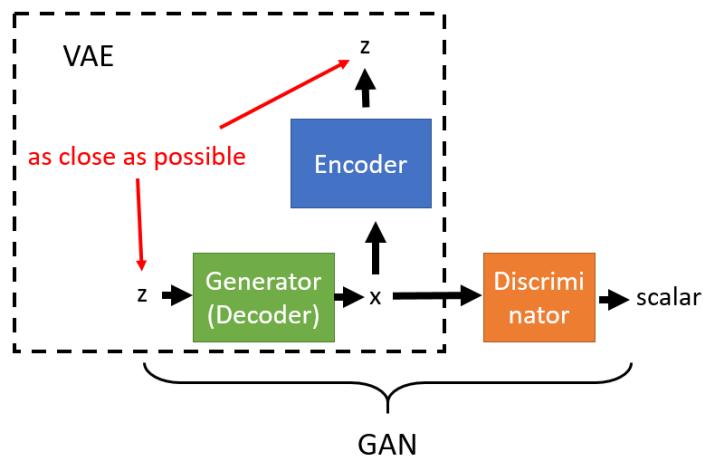
### 2.1 VAEGAN 的原理

我们知道 VAE 可以实现从原始分布编码到一个 normal 的分布，再从 normal 的分布解码回原始分布的功能，然后这个解码用到的 decoder 就可以单独被提取出来作为生成器使用。但是，VAE 有一个很大的问题就是，解码产生的图片往往都比较模糊。那为什么产生的图片会模糊？我会在之后学习 VAE 的过程中再去研究，目前我们只是知道，VAE 产生的图片是容易模糊的。于是就出现了 VAEGAN，它的作用，就是给 VAE 加上了 GANs 的架构，通过判别器使得 VAE 产生的图片变得清晰。因此我们可以理解为，VAEGAN 就是利用 GANs 去提升了 VAE 的图片生成质量。



我们来说明一下这个架构。首先输入一张真实图像  $x_{in}$ ，通过一个 encoder 变成一个 normal 的分布  $z$ ，然后  $z$  再输入到 decoder 中产生生成图像  $x_{out}$ ，这时 VAE 希望  $x_{out}$  与  $x_{in}$  之间的 loss 越小越好，但是 loss 小不见得图片就清晰，于是  $x_{out}$  又会被放入一个判别器中，去让判别器判断输入的图片属于  $P_{data}$  还是  $P_G$ ，这样最终  $x_{out}$  就会与  $x_{in}$  不仅相似还清晰。

而关于 VAEGAN，有趣的一点是，我们不仅可以用 GANs 来提升 VAE，也可以用 VAE 来提升 GANs。如果是被用作后者的话，“GANVAE”其实就等效为 CycleGAN 的一部分，如下图所示。

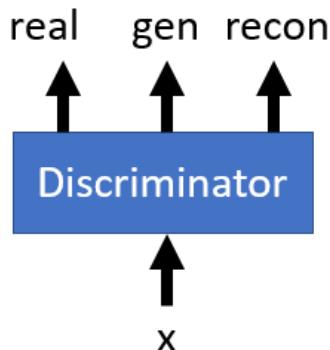


这两者的差别在于，把  $x$  encode 成  $z$  的过程是放在生成器的前面还是后面，如果放在前面，就是 VAEGAN；如果放在后面，就是 CycleGAN。

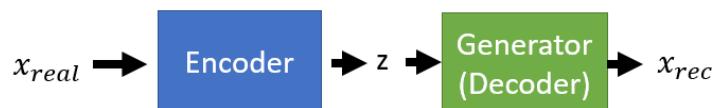
## 2.2 VAEGAN 的变种

上一节的 VAEGAN 存在一个问题，就是 Encoder 编码出的  $z$  不一定完全符合我们期望的 normal  $z$  的样式，也就是编码出的  $z$  和我们输入的  $z$  可以存在细微的不同，但是通过生成器的处理，它们产生的输出都能够骗过判别器。

为了解决这一问题 VAEGAN 提出了一个变种，它改变了判别器，使得判别器能更精细的鉴别输入图像的种类，共包括三个种类：真实图片，重构图片，与生成图片。



其中重构图片与生成图片的差异是，重构图片  $x_{rec}$  是真实图片通过 encoder 与 decoder 之后产生的图片，如下图所示。



而生成图片  $x_g$  仅仅是一个初始分布  $z$  通过 decoder 之后产生的图片，如下图所示：

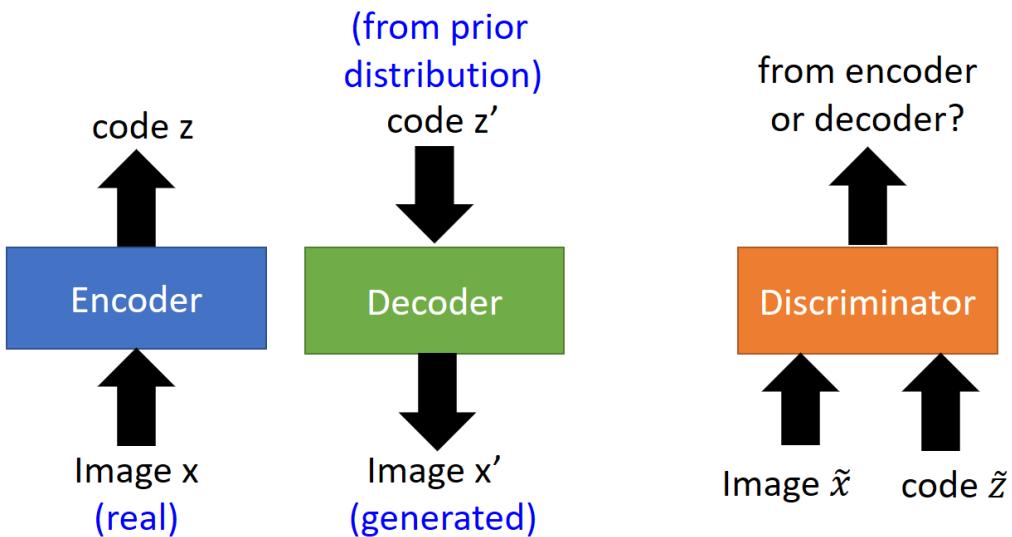


于是现在  $x_{rec}$  与  $x_g$  的差异能够被判别器学到，又因为生成器是共用的，那为了消除这种差异只能让输入  $z$  接近一致，也就是 encoder 编码出的  $z$  不断逼近我们给定的输入  $z$ ，最终几乎完全一样。结果证明，这种模型在实验中比 VAEGAN 有少量的提升。

### 3. BiGAN

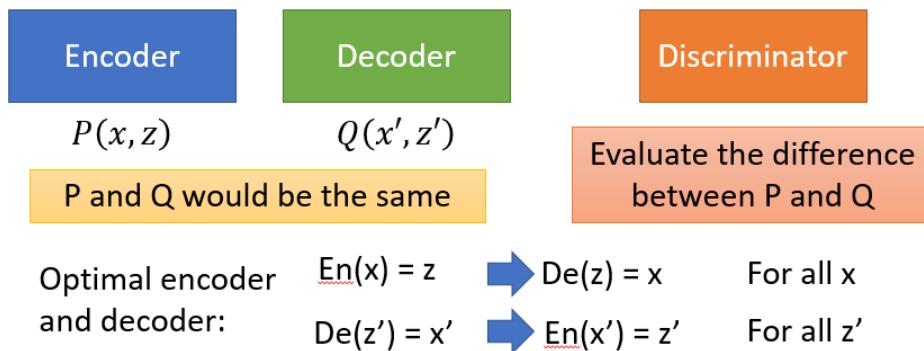
#### 3.1 BiGAN 的原理

GANs 的花样可以说五花八门, 上篇介绍的 VAE-GAN 是基于 GANs 训练的 auto-encoder, 现在 BiGAN 提出, encoder 与 decoder 的训练可以分开进行, 结果依然是可靠的。



BiGAN 的架构其实挺好理解。首先有一个 encoder, 输入真实图片, 产生一个编码; 另外有一个与 encoder 无关的 decoder, 输入一段给定编码, 产生一个图片; 最后有一个判别器, 它输入一段图片与编码的配对, 它需要去判断这个配对是来自 encoder 还是 decoder。

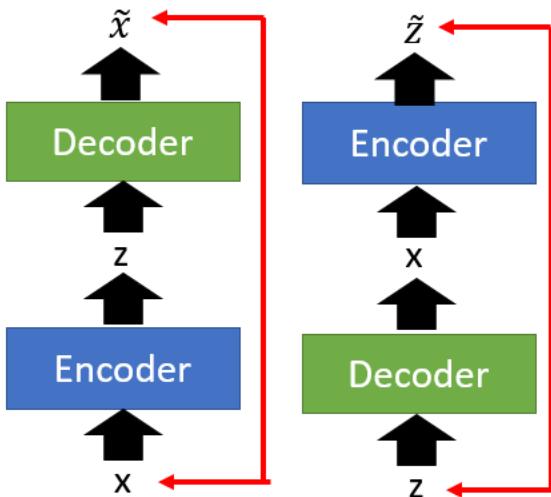
我们从理论上来简单分析一下这个模型产生的效果。假定 encoder 产生的配对的分布是  $P(x,z)$ , 其中  $x$  已知,  $z$  未知; 另一边 decoder 产生的配对的分布是  $Q(x',z')$ , 其中  $z'$  已知,  $x'$  未知。而判别器的作用是在消除  $P$  与  $Q$  之间的 Div, 也就是说最终  $P$  分布会与  $Q$  分布长得一模一样, 这意味着 encoder 学到的输出  $z$  就是已知的  $z'$ , 而 decoder 学到的输出  $x'$  就是已知的  $x$ 。因此, 最终的 encoder 与 decoder 就是互为逆运算的过程。



下面我们来讨论一下 BiGAN 的实际使用效果。

#### 3.2 BiGAN 的讨论

在讨论 BiGAN 之前, 我们先介绍一个与之对应的结构 BiVAE。



就如同 BiGAN 的本质是在训练两个 GANs, BiVAE 的本质就是在训练两个 VAE。现在我们已知的信息是真实图片  $x$  和先验编码  $z$ 。首先我们搭建一个 VAE, 实现  $x \rightarrow \text{encoder} \rightarrow z \rightarrow \text{decoder} \rightarrow \tilde{x}$  的过程, 我们希望  $\tilde{x}$  与  $x$  越接近越好; 然后我们把 encoder 和 decoder 反过来, 实现  $z \rightarrow \text{decoder} \rightarrow \tilde{x} \rightarrow \text{encoder} \rightarrow \tilde{z}$  的过程, 我们希望  $\tilde{z}$  与  $z$  越接近越好。

那这样的 BiVAE 的训练结果如何呢? 就是输出结果  $\tilde{x}$  与原图非常相似, 但是却比较模糊, 这与传统 VAE 的毛病几乎是一致的。我们再来看一下 BiGAN 的输出结果, 就是产生的图片会非常清晰, 但是往往与原图不相似。举一个例子, 比如说输入是一张鸟的图片, 通过 BiGAN 的输出也会是一张清晰的鸟的图片, 但是输出的鸟与输入的鸟不是同一种鸟。

于是, 现在我们可以知道, 下一步人们需要做的, 就是构造一个模型, 把 BiGAN 和 BiVAE 的优点结合起来, 实现一个稳定可靠的 auto-encoder。可惜截至目前, 还并没有一个模型能够解决这一问题。

## 第四章 附录

### 1.致谢及引用

声明：本书大部分内容是作者学习并撰写，但也有部分资料整理自互联网，  
本书仅用作学习。现注明转载出处如下：

致谢：

#### 李宏毅老师的基础 GANs 理论讲解

网址：<https://www.bilibili.com/video/av24011528/?p=1>

#### 第一章·理解 GANs

网址：<https://blog.csdn.net/sxf1061926959/article/details/54630462>

网址：<http://www.gwylab.com/paper-gans.html>

##### · 经典 GANs 整理

网址：<https://blog.csdn.net/omnispace/article/details/79071188>

网址：<https://blog.csdn.net/u013982164/article/details/79608258>

网址：

[https://mp.weixin.qq.com/s?\\_\\_biz=MjM5MTQzNzU2NA%3D%3D&idx=1&mid=2651662154&sn=44d09ee2beeca8f50415acfc06a12a1a](https://mp.weixin.qq.com/s?__biz=MjM5MTQzNzU2NA%3D%3D&idx=1&mid=2651662154&sn=44d09ee2beeca8f50415acfc06a12a1a)

网址：[http://speech.ee.ntu.edu.tw/~tlkagk/courses\\_MLDS18.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses_MLDS18.html)

#### 第二章·GANs 的提升

##### Part1

###### 3. fGAN

网址：<https://alberthg.github.io/2018/05/13/wgan/>

###### 7. SNGAN-7.2 奇异值分解

网址：<https://baike.baidu.com/item/%E5%A5%87%E5%BC%82%E5%80%BC/9975162>

网址：<https://blog.csdn.net/zhongkejingwang/article/details/43053513>

##### Part2

###### 2. ImprovedDCGAN

网址：[http://blog.sina.com.cn/s/blog\\_76d02ce90102xq4d.html](http://blog.sina.com.cn/s/blog_76d02ce90102xq4d.html)

###### 4. BigGAN

网址：<https://zhuanlan.zhihu.com/p/46581611>

##### Part3

###### 3. BEGAN

网址：

[https://blog.csdn.net/qq\\_25737169/article/details/77575617?locationNum=1&fps=1](https://blog.csdn.net/qq_25737169/article/details/77575617?locationNum=1&fps=1)

## 第三章·GANs 的应用

### Prat1

#### 3. StackGAN

网址: <https://blog.csdn.net/zlrai5895/article/details/81292167>

#### 6. StyleGAN

网 址 : <https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431?spm=a2c4e.11153940.blogcont686373.15.99143440XmYsTe>