

Lecture 6: 生成对抗网络 (GAN)

Lectured by HUNG-YI LEE (李宏毅)

Recorded by Yusheng zhao (yszhao0717@gmail.com)

Network as Generator:

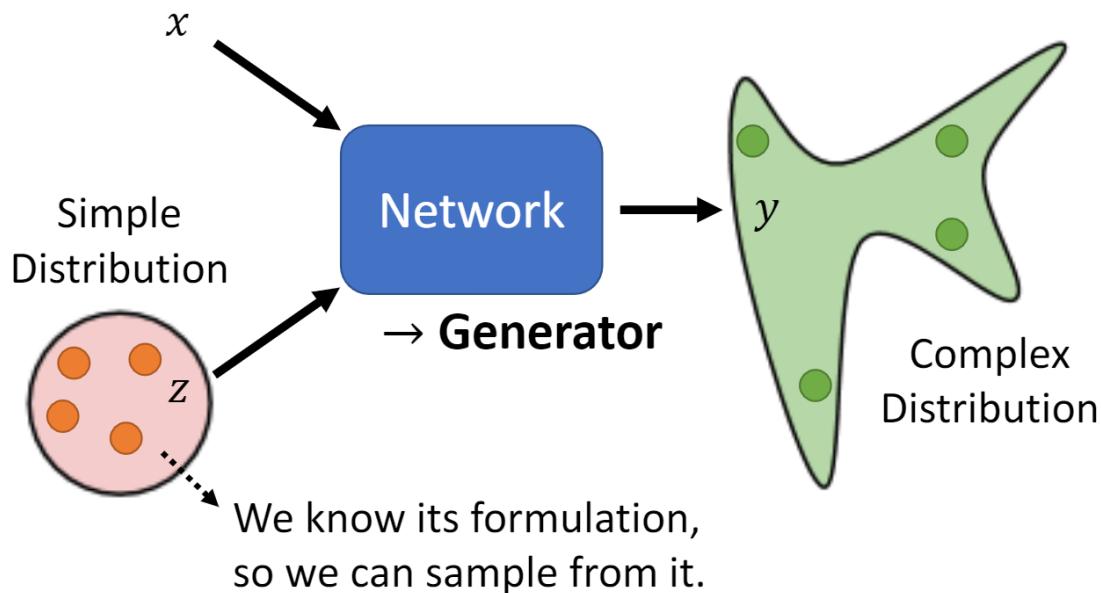
之前学到的Network都是一个function: $x \Rightarrow \text{Network} \Rightarrow y$

这个Lecture的主题是把Network看作是一个Generator，作深层使用，这个network特别的地方是输入会加上一个random的variable z : 网络的输入不仅有 x ，还有Simple Distribution的分布 z ， y 是同时看 x 和 z 的输出。

network怎么同时看 x 和 z 呢？有很多不同的做法，取决于network的架构。举个栗子： x 是个向量， z 是个向量，两者直接接起来变成比较长的向量作为一个输入；或者恰好 x 和 z 长度一样，那两者相加后当作输入。

z 特别的地方就是它是不固定的，所以每次都会不一样，它是从一个distribution中sample（采样）出来的，这个distribution（我们自己决定）必须够简单：我们知道这个分布的样子长什么样子，比方说高斯分布（Gaussian Distribution）、均匀分布（Uniform Distribution）。随着我们sample到的 z 不同，输出 y 也会不同。最后得到的输出构成一个复杂的分布（Complex Distribution）。

这种可以输出一个distribution的network我们称之为**Generator**。



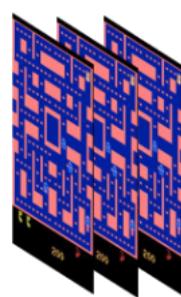
为什么我们需要Generator? (即为什么输出需要是个分布)

以video prediction为例，按照我们之前所学过的supervised learning方法，将过去的画面帧（部分）输入Network，输出一张预测的帧（部分）：会出现预测画面明显不合理的情形（network变成一个“老好人”，出现同一种输入得到多种输出的情形）——解决方法：让输出呈现一种概率的特征，或者说输出不固定（包含多种可能）。

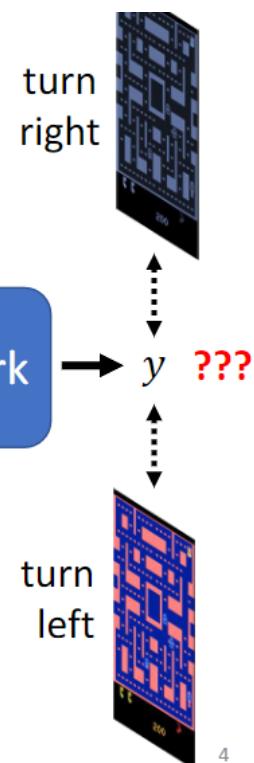


Prediction

Video Prediction

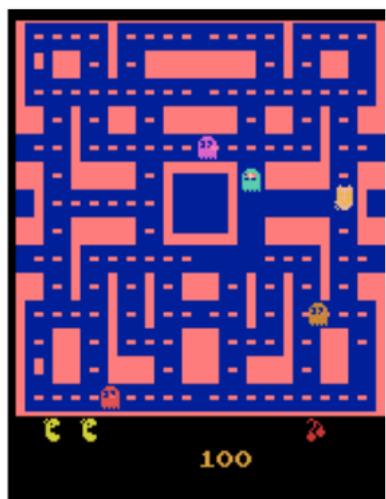


Previous frames



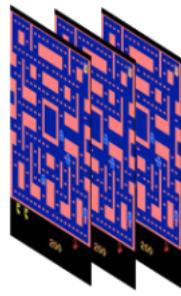
4

上图监督学习的状况，下图是generator的状况 ↗



Prediction

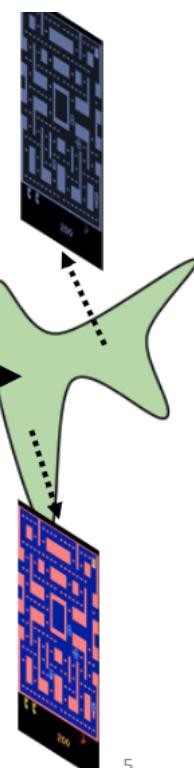
Video Prediction



Previous frames



Z
Simple
Distribution

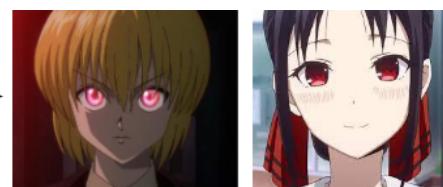
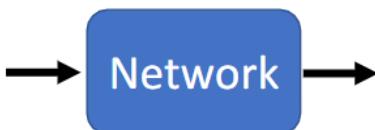


5

因此，**当我们的任务需要一些创造力 (creativity)** 的时候（或者说是the same input has different outputs），我们需要Generator

- Drawing

Character
with red eyes



同样红眼睛的特征，可能绘画出来的头像不太一样。

- Chatbot

你知道輝夜是
誰嗎？



她是秀知院學生會 ...

她開創了忍者時代 ...

一个Q&A相似的道理，回答也有不一样.....

上述两个例子二刺螈浓度过高.....老师一本正经的胡说八道：）

上面简单说明了我们为什么需要一个具有generator的model，且这个model是做怎样的任务的。

其中一个最著名的model称之为**Generative Adversarial Network (GAN)**

介绍Generative Adversarial Network (GAN)

GAN怎么念？——淦，完毕。

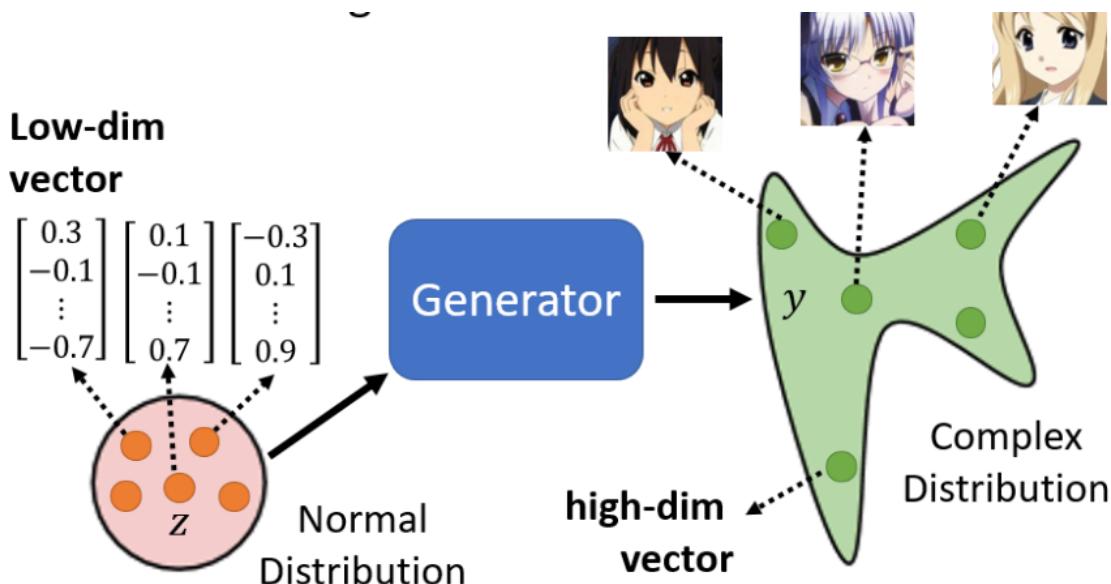
Generative Adversarial Networks (arxiv: <https://arxiv.org/abs/1406.2661>)

GAN的动物园——收集了超过五百种的变种GAN

以Anime Face Generation为例，

- 首先说明**Unconditional generation**（输入拿掉了 x , 只保留了分布 z ）

输入 z （一个从Normal Distribution采样出来的一个向量，通常是一个Low-dim vector，维数由自己决定的），把 z 丢进一个Generator里面，输出一个对应的向量 y （满足Complex Distribution，通常是一个high-dim vector，这样每一个高维的向量其实就对应一个生成的图像）

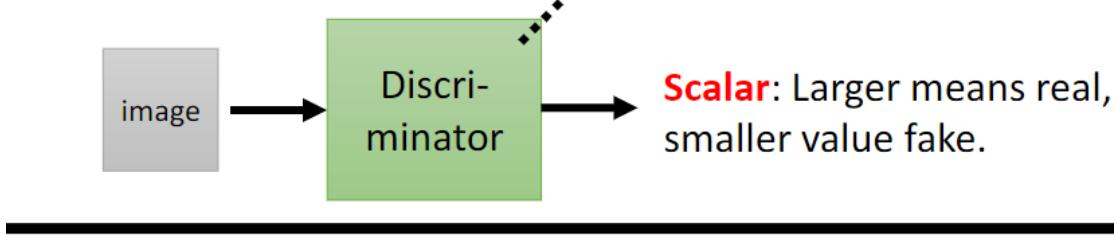


我们的目标是，不管左边输入的 z 是怎样的向量，都会输出一个二刺螈脸图像（向量）

- 为什么这边是normal distribution (正态分布) 呢？可以用别的吗？——当然可以用别的，实践证明不同的分布可能区别并没有真的非常大。可以探讨很多文献，有设计实验验证不同分布的区别。这个任务的要求只需要输入 z 的分布足够简单，选择distribution的任务可以交给generator实现。
- Discriminator**

作用：拿一张图像作为输入，本质上还是一个神经网络（或者说是一个function），输出是一个数字（标量scalar，这个数字越大就说明这个输入越像是真实的二刺螈人物的图像）

(that is, a function).

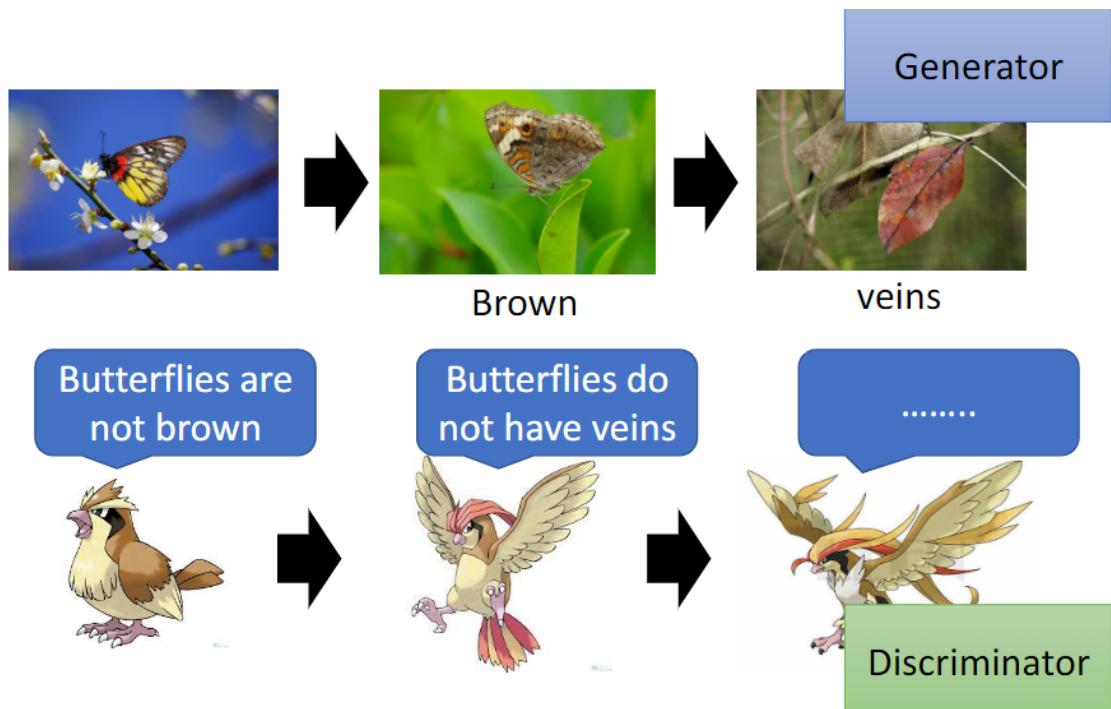


Generator是一个神经网络，同样的discriminator也是一个神经网络，涉及到的网络架构完全可以自己设计：可以是CNN，亦可以是transformer——只要能够产生我们所需要的输入输出都可以。

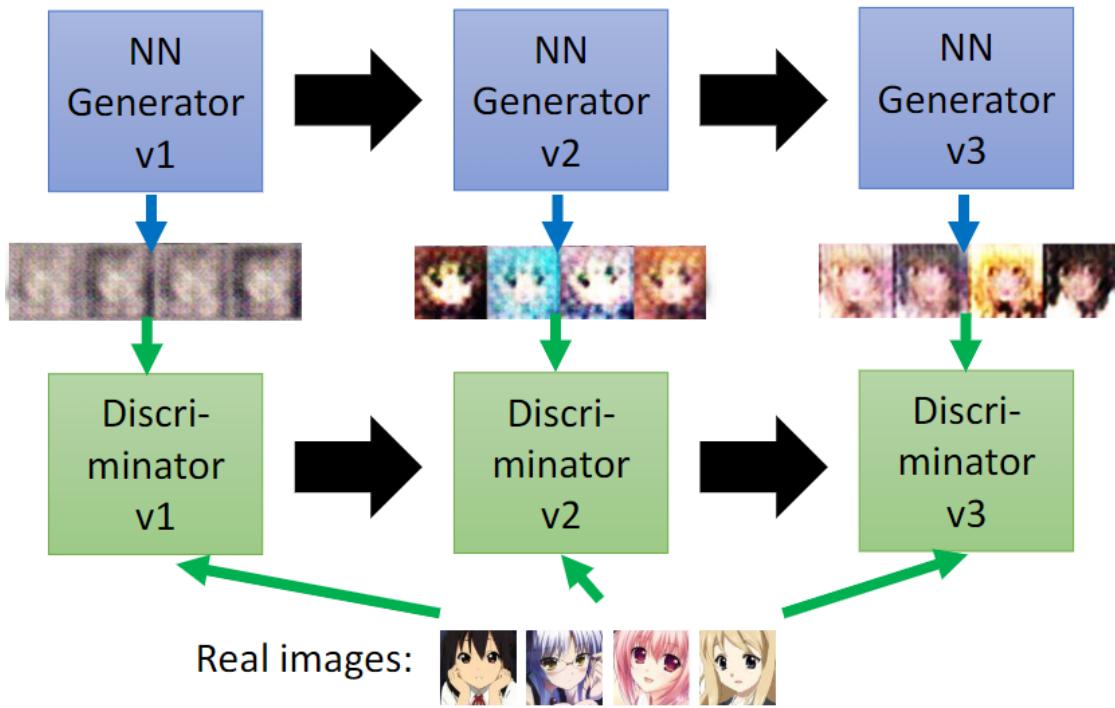
- **Basic Idea of GAN**

枯叶蝶的例子：天敌的存在导致，原来五彩斑斓的枯叶蝶祖先颜色就转变为枯叶般的棕色。同时，天敌也在进化了，辨别枯叶蝶的能力加强了；于是乎，枯叶蝶继续进化，生长出类似“叶脉”，更像枯叶；周而复始，天敌进化，猎物也再进化.....

对应到GAN上，猎物就是Generator，天敌就是Discriminator



我们的任务是generator画出二刺螈的人物，它是按照以下方式迭代的



第一代的generator的参数几乎是随机的，得到的图像是凌乱的杂讯，第一代discriminator的任务就是generator的输出和真实的图像（ground truth）有什么区别（举例说有没有两个圆球（眼睛））；第二代generator相比要进行“进化”——为了“骗”过第一代discriminator（比方说进化出眼睛），而第二代discriminator也随之进化，试图分辨第二代图片间的差异；周而复始，比学赶帮超.....generator产生出discriminator输出较大的二刺螈人物图像。

早期的论文，描述Generator和Discriminator的关系就是“**adversarial**”（对抗的）——亦敌亦友的关系（Naruto和Sasuke.....）

- 从算法角度看待**Generator**和**Discriminator**【Algorithm】

两个network（每个network有好几层）：

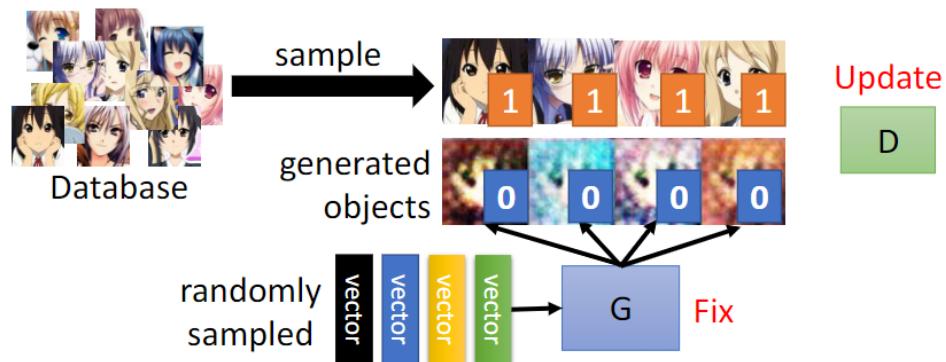
- 初始化Generator和Discriminator的参数：G和D
- 在每个iteration：

- STEP 1:** 固定住generator G，然后只训练（更新）discriminator D

初始化步骤中，从高斯分布randomly sample到low-dim的向量，输入generator中得到输出的high-dim的向量（图像）大概率非常不像二次元图像。

我们会有一个database（会有很多二刺螈图像），我们从这个database中sample出一些图像出来，拿真正的二次元头像和G的结果一起去训练我们的discriminator

discriminator的训练目标是要分别出generator和ground truth的差异。实际的操作可能是这样子的：训练一个分类器，ground truth为1，generator objects为0，做一个二分类的re任务；当然也可以当作回归问题来做。

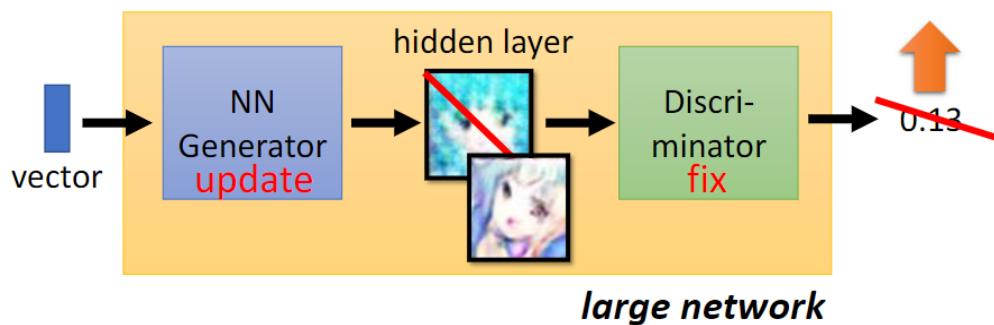


Discriminator learns to assign high scores to real objects and low scores to generated objects.

- **STEP 2:** 固定住discriminator D, 然后只训练(更新) generator G

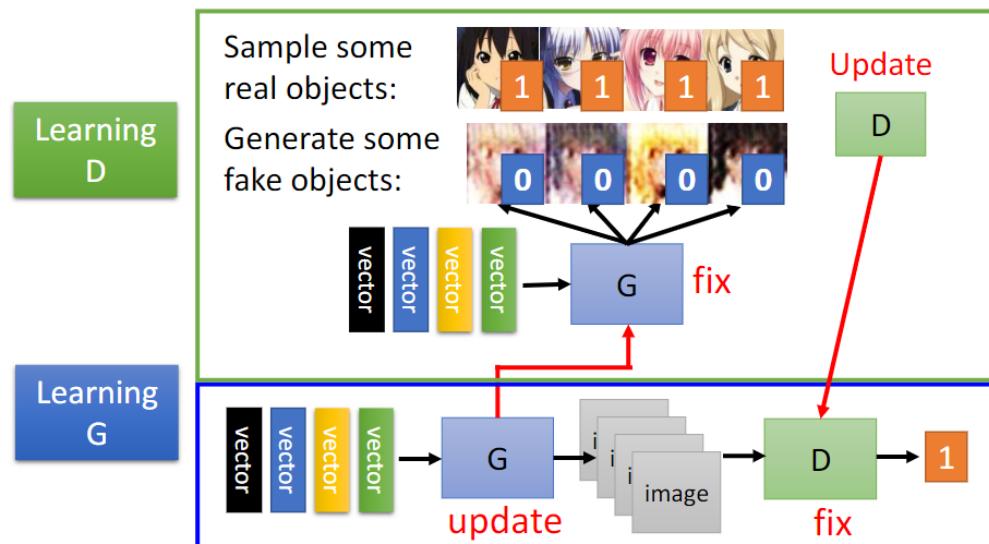
generator的训练目标就是要输出的向量(图像)能够以假乱真, 骗过discriminator

我们有一个generator, 从高斯分布randomly sample到low-dim的向量作为输入, 然后产生一个图片, 将图片丢进discriminator, 产生一个分数。discriminator的参数是固定的, 这时候需要调整generator的参数



实际上, 我们会把generator和discriminator接起来, 看作是一个很大的network (large network), 里面有很多层(前面更新调整参数, 后面几层参数固定), 其中有一层得到的输出很宽, 它的dimension就和图片里面的pixel数目乘三是一样的, 整理以下就是我们的目标输出(图片)。这个network的训练方法和我们之前讲的没有什么不同, 唯一的不同就是其目标discriminator的输出是越大越好。

- **STEP 3:** 接下来, 就是反复的训练discriminator和generator, 重复**STEP 1**和**STEP 2**



最后得到一个generator能够生成以假乱真的图片, 那么训练任务就结束了。

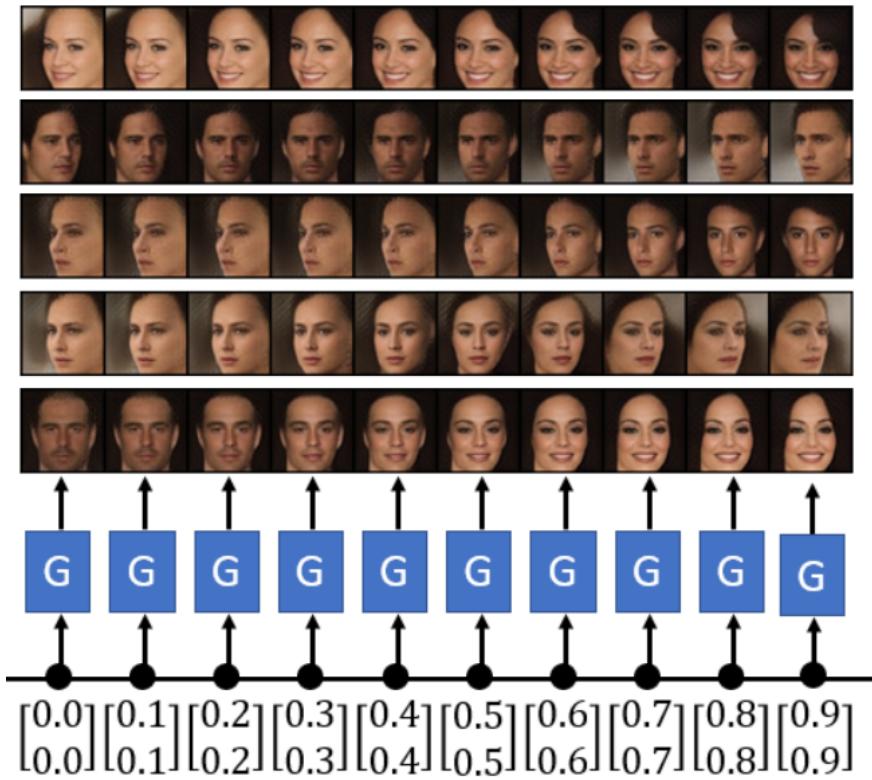
- 二次元人物头像生成实战

2019年，用styleGAN做的二次元人物头像，<https://www.gwern.net/Faces>

- 除了产生动画人脸，也可以用GAN产生真实人脸

Progressive GAN

对于两张生成的人脸图像，两个向量之间不断做内插（interpolation），可以得到两个人物之间的渐变

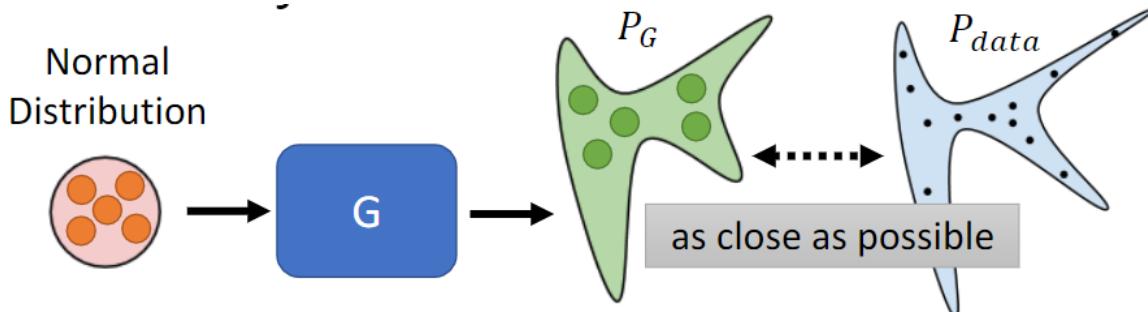


这个渐变可以表现为人物表情的变化，眼神方向的变化。

以上用实例说明了GAN的实际运作是如何的，以下从理论上说明为什么Generator和Discriminator的互动可以让Generator可以产生像是真正人脸的图片。

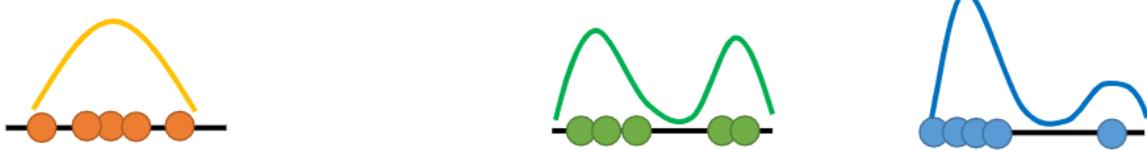
Theory behind GAN

我们训练的目标是什么？——以我们之前讲的神经网络为例，我们要定义一个loss function，用梯度下降法，不断地minimize这个loss function就行了。而在这个Generator地问题里，我们要minimize两个dscrriminator的差距，如下图↓



我们从Normal Distribution（正态分布）中sample出vector丢进Generator里面，产生出一个比较复杂的distribution，这个复杂的distribution被称之为 P_G ，而ground truth即真实的数据集形成另一个实在的一个distribution被称之为 P_{data} ，我们期待这两者越接近越好——这就是我们minimize的目标。

分布的布局假设如下↓



以上，我们总结这样一个式子：

$$G^* = \arg \min_G \text{Div}(P_G, P_{data}) \quad (1)$$

其中 $\text{Div}(P_G, P_{data})$ 代表 P_G 和 P_{data} 之间的散度 (Divergence) 【我们可以想成是两个distribution之间的某种距离，divergence本身就是distribution的一种measure】

我们实际工作的目标就是：找到一组generator的参数使得按上述方法经过generator后得到的 G^* 越小越好

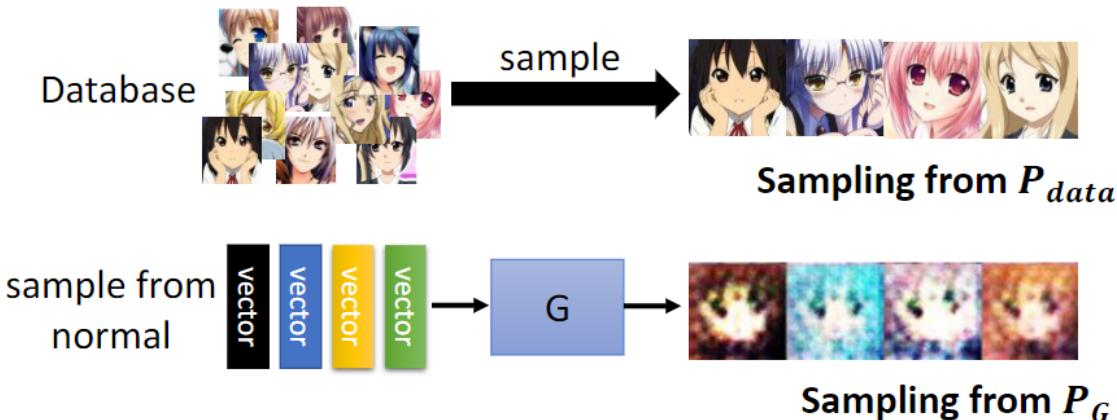
关于散度 (divergence)：这里不详细介绍，课外另外补充学习。有很多种实现方法：**KL divergence**、JS Divergence等等

但是实际上在计算散度时，会涉及到复杂的积分（实作上可能几乎不知道怎么计算），如果我们计算散度非常困难，那么怎么找到那个理想的 G^* 呢？——这个就是在train这种generator经常遇到的问题，而GAN以很神奇巧妙的方式，可以突破计算散度的限制。以下进行介绍

Sampling is good enough.....

GAN要求我们只需要知道怎么从 P_G 和 P_{data} 这两个分布sample东西出来，就有办法算divergence。

- 把真实的database拿出来，里面sample一些就可以组成 P_{data}
- 把normal distribution采样出来的vector输入进generator，产生一堆输出-->就是从 P_G 采样出来的结果



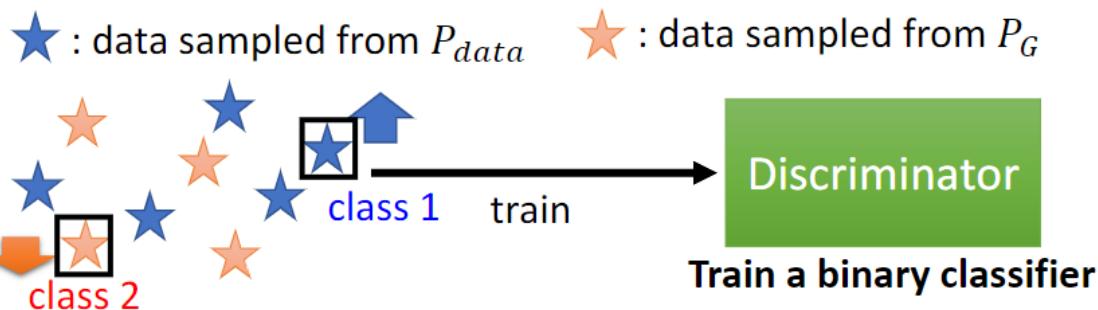
既然已经知道如何sample了，那么怎么计算（估算）divergence呢？——这需要Discriminator的力量，我们根据sample出来的data（分别来自 P_G 和 P_{data} ）来对Discriminator进行训练，训练的目标就是当看到Real data (from P_{data}) 时，给比较高的分数，而看到generative data时，就给比较低的分数。<https://arxiv.org/abs/1406.2661>

假如把它看作是Optimization的问题，总结为以下任务：训练一个Discriminator

需要maximize的称之为**Objective Function**；对应的，需要minimize的称之为**Loss Function**

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

Objective Function for D: $V(G, D) = E_{y \sim P_{data}} [\log D(y)] + E_{y \sim P_G} [\log(1 - D(y))]$

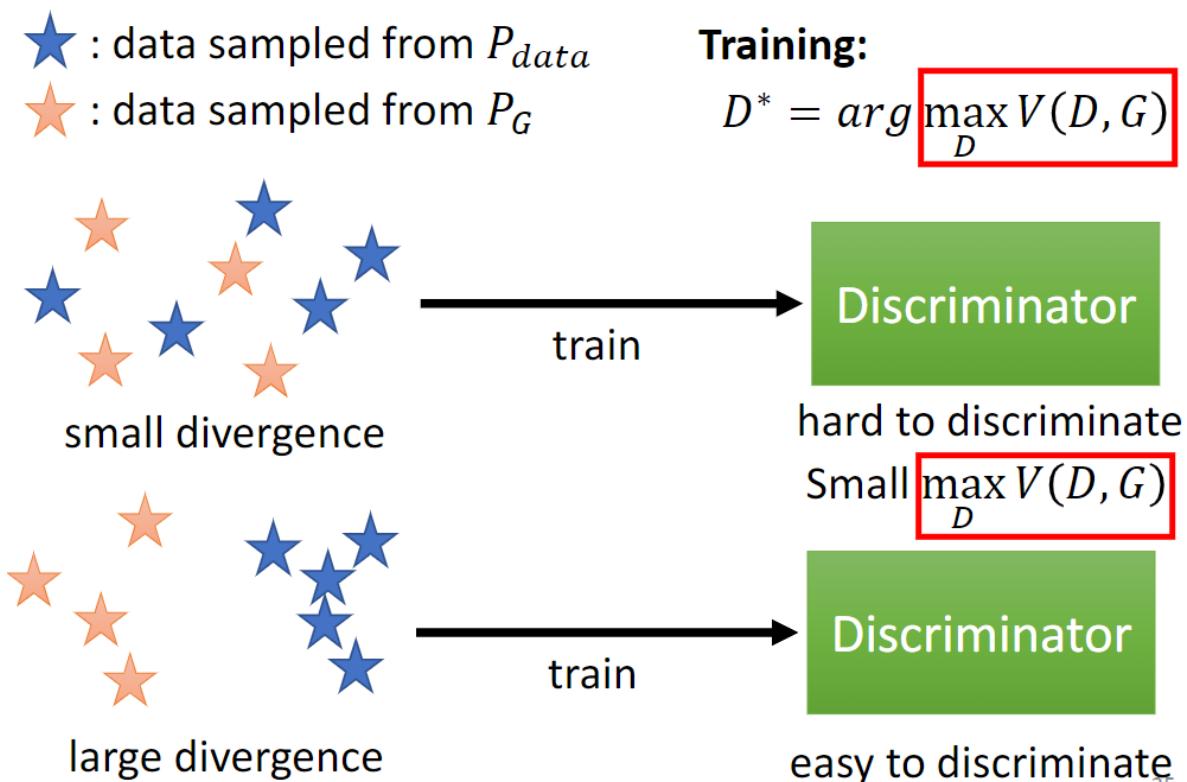


我们需要D越大越好，上面这件事情等同于一个 P_G 和 P_{data} 的二分类器 (binary classifier)

$$D^* = \arg \max_D V(D, G) \quad = \quad \text{Training classifier: minimize cross entropy}$$

negative cross entropy

直观上我们可以理解，Objective Function的值（在穷举为得到最大值）和divergence相关，详细的证明参见GAN的原始论文。



我们本来的目标是要找一个generator，去minimize两个分布的divergence，而由上述的关系，我们何不仿用训练目标**Objective Function**: $\max_D V(G, D)$ 来替换原有目标，求divergence的较大值，记为 $V(D, G)$ ，综上，我们转换任务，得：

$$\begin{aligned} \text{Generator : } G^* &= \arg \min_G \max_D V(G, D) \\ \text{Discriminator : } D^* &= \arg \max_D V(D, G) \end{aligned} \tag{2}$$

其中the maximum objective value ($\max_D V(G, D)$) 和JS divergence相关。

我们可以改变（自己定义）objective function，从而衡量各种各样得divergence（散度），在[文章中](#)（F GAN），总结了许多方法，using the divergence you like.

$$D \rightarrow G$$

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 χ^2	$\int \frac{(q(x)-p(x))^2}{p(x)} dx$	$(u - 1)^2$
Neyman χ^2	$\int \frac{(p(x)-q(x))^2}{q(x)} dx$	$\frac{(1-u)^2}{u}$
Squared Hellinger	$\int \left(\sqrt{p(x)} - \sqrt{q(x)} \right)^2 dx$	$(\sqrt{u} - 1)^2$
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	$\int p(x) \pi \log \frac{p(x)}{\pi p(x)+(1-\pi)q(x)} + (1-\pi)q(x) \log \frac{q(x)}{\pi p(x)+(1-\pi)q(x)} dx$	$\pi u \log u - (1-\pi+\pi u) \log(1-\pi+\pi u)$
GAN	$\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} dx - \log(4)$	$u \log u - (u+1) \log(u+1)$

Name	Conjugate $f^*(t)$
Total variation	t
Kullback-Leibler (KL)	$\exp(t - 1)$
Reverse KL	$-1 - \log(-t)$
Pearson χ^2	$\frac{1}{4}t^2 + t$
Neyman χ^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	$(1-\pi) \log \frac{1-\pi}{1-\pi e^{t/\pi}}$
GAN	$-\log(1 - \exp(t))$ <small>[37]</small>

不同的divergence，去怎样设计objective function，这篇文章都有详细的记载。

曾经有人觉得GAN没有很好的train是因为divergence没有去很好的计算，但是F GAN这篇文章说明了即使是精致的设计了divergence的近似的计算，GAN也依旧不好train.....

以下介绍GAN训练的小tips。

Tips for GAN

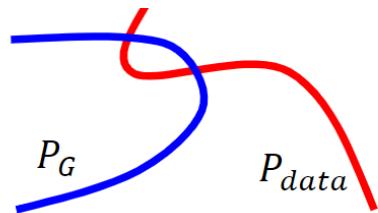
JS divergence有什么样的问题？

- 在绝大多数case里面， P_G 和 P_{data} 重叠的部分非常少 (are not overlapped)
 - 理由1：data本身的特性，由于图像本身是高维的，代表图像的向量就是处于高维空间中的低维的**manifold (流形)**。对于一个类别的图像本身（譬如二次元头像），分布的区间是非常狭窄的。

流形是一种空间，一个流形好比是一个 d 维的空间，在一个 m 维的空间中 ($m > d$) 被扭曲之后的结果（一般维度压缩的方法中都会提到这个词，谱聚类中就有涉及这个思想，稍后再说），可以类似于地球，地球的表面是一个球面

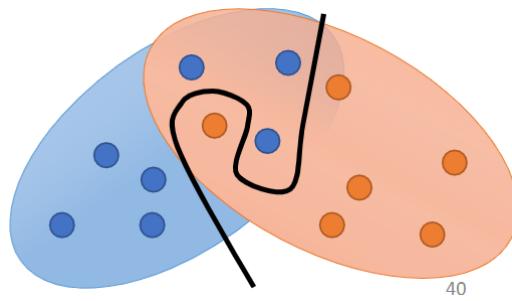
Both P_{data} and P_G are low-dim manifold in high-dim space.

The overlap can be ignored.



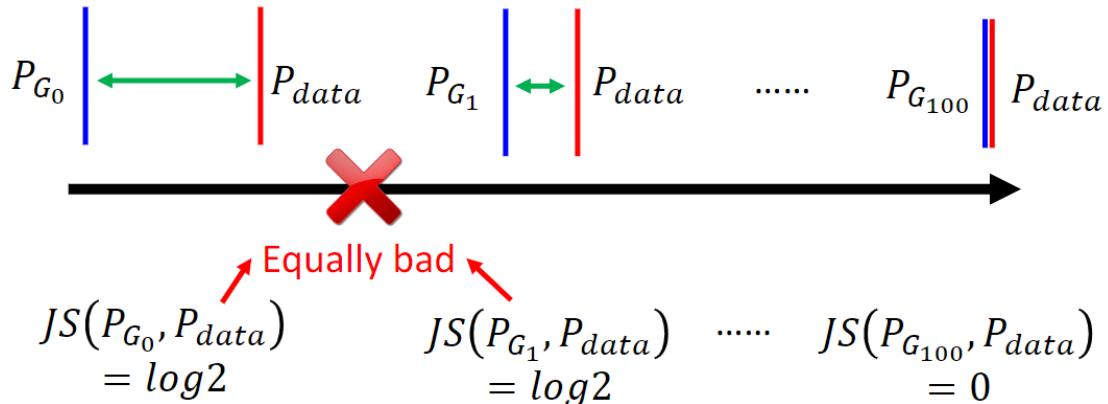
从二维空间的角度， P_G 和 P_{data} 都是二维空间上的两条直线，除非它俩重合，那么它俩相交的部分几乎可以忽略不记。

- 理由2：Sampling (采样的角度上看)，尽管 P_G 和 P_{data} 是有相交的，但是由于我们做初步处理的时候是先randomly sample的，那么在训练数据中相交的就更少了。



以上两个理由说明了 P_G 和 P_{data} 是几乎没有重叠的。

因此，如果这两个分布不重叠，那JS divergence会算出来就一直是 $\log 2$ ；若重合，则为0



JS divergence无法反映出 P_G 和 P_{data} 的差别优劣。

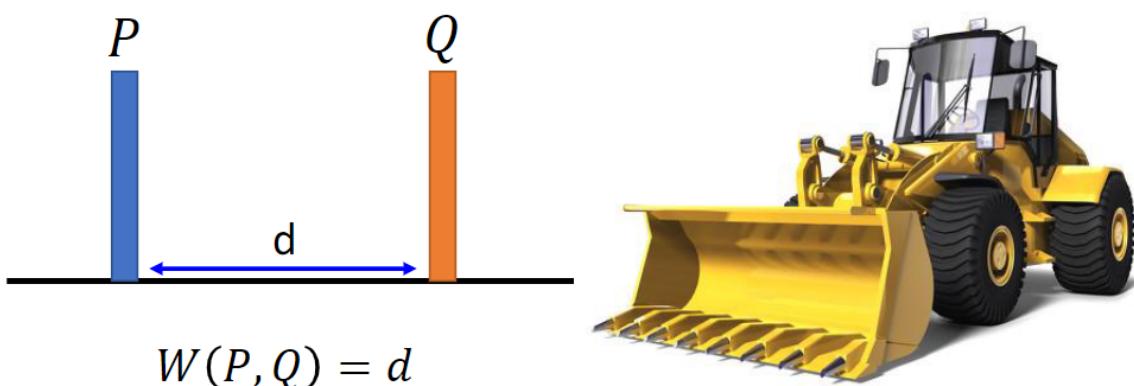
如果两个分布不重叠，那么在训练中二分类器正确率几乎会达到100%，这种训练近乎无效...

所以在早期的GAN训练仿佛是一种黑魔法，如果无法实时的去检查就无法验证train的效果如何。

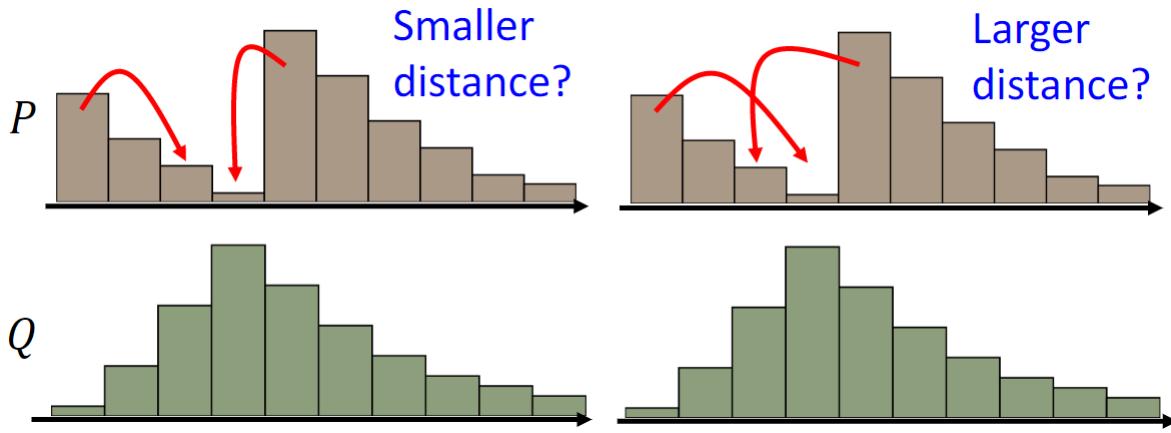
所以简单的而分类器无法提供有效的GAN的训练效果的反馈。以下介绍改良的替代品WGAN。

Wasserstein distance (earth mover distance)

假设两个分布： P 和 Q 。想象下 P 是土堆 (a pile of earth) 而 Q 是土堆放的目的地 (target) 。而**平均距离 (average distance)** 就是推土机 (earth mover) 把土从 P 到 Q 的距离



对于更复杂的分布情况，区间更宽些，把 P 变成 Q 有无穷多种，穷举所有的“moving plan”中**average distance最小的值才是Wasserstein distance**，参考博客：[Wasserstein GAN and the Kantorovich-Rubinstein Duality - Vincent Herrmann](#)



使用Wasserstein distance可以反映出训练过程中两个分布之间的距离变化（优劣），而二分类器无法做到。在训练过程中，它描述了一种渐渐变化的趋势（而非突变），这种趋势体现的差异能够让我们去train我们的generator去minimize我们定义的Wasserstein distance

WGAN

定义：就是用Wasserstein distance取代JS divergence的GAN，用Wasserstein distance来评估 P_G 和 P_{data} 之间的距离。

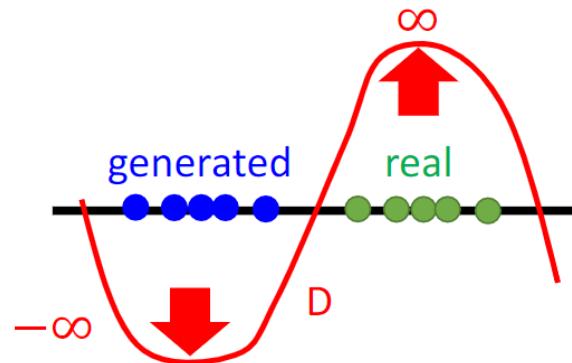
$$\max_{D \in 1 - Lipschitz} \{E_{y \sim P_{data}} [D(y)] - E_{y \sim P_G} [D(y)]\} \quad (3)$$

Wasserstein distance如何计算——去解答上 (3) 这个式子，解出来的值就是distance的数值解，注意到：这里的 y 指的是network的输出

$D \in 1 - Lipschitz$ ：表明 D 必须是一个足够平滑的function；如果没有这个限制，关于 D 的训练永远不会收敛，这些maximum就会趋于无穷大。文章：<https://arxiv.org/abs/1701.07875>

Without the constraint, the training of D will not converge.

Keeping the D smooth forces $D(y)$ become ∞ and $-\infty$



怎么去真的解 (3) 这个式子？又怎样去保证 D 的平滑性？

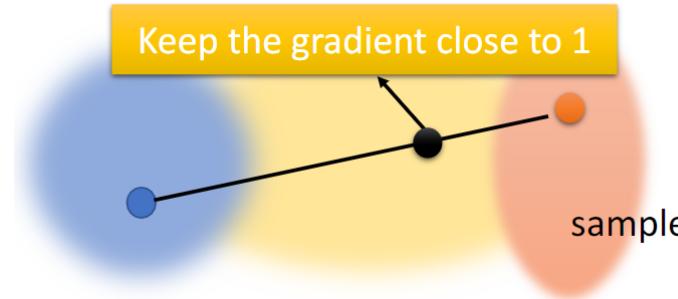
在最初的解答中研究者做了粗糙的处理，这个实际上没有解这个式子，而是做了粗线条的近似计算，当然缺陷很大。

Original WGAN → Weight

Force the parameters w between c and -c

After parameter update, if $w > c$, $w = c$; if $w < -c$, $w = -c$

后来另外一个想法是：Improve WGAN → Gradient Penalty，来自[一篇文章](#)

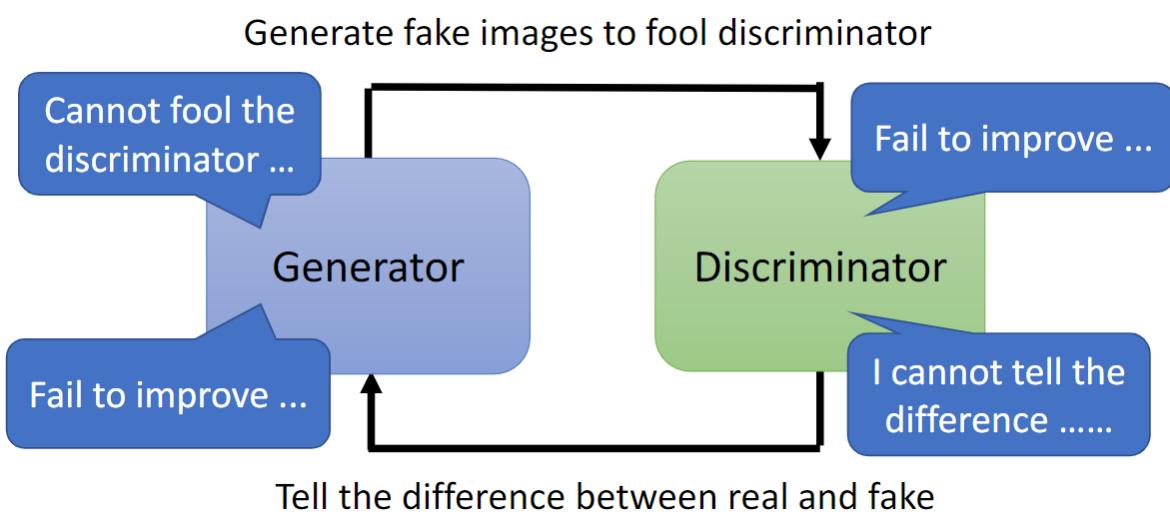


以及[Spectral Normalization for Generative Adversarial Networks](#)所介绍的**Spectral Normalization** ⇒ Keep gradient norm smaller than 1 everywhere (为了得到更好的GAN，应当参考下这篇文章)

课后问题解答

generator和discriminator通常都不会一直迭代到收敛，只迭代几次，就交换给另外一个生成器（因为时间成本）；所以在实际做的时候，许多参数就会“继承”前一个generator或是对应的discriminator（这个就是理论和实做的区别），而不会重头开始训练

WGAN有很多种不同的做法，其中效果最好的就是上述提到的**Spectral Normalization (SNGAN)**。但是即使有WGAN，GAN依然很难train起来，GAN是以不好train而闻名的。所以我们亦可以说GAN is still challenging...



Generator和Discriminator必须“棋逢对手”，相互“对抗”，相互促进。如果其中一方没有train好，另一方也会出现停滞——双成员的正反馈。

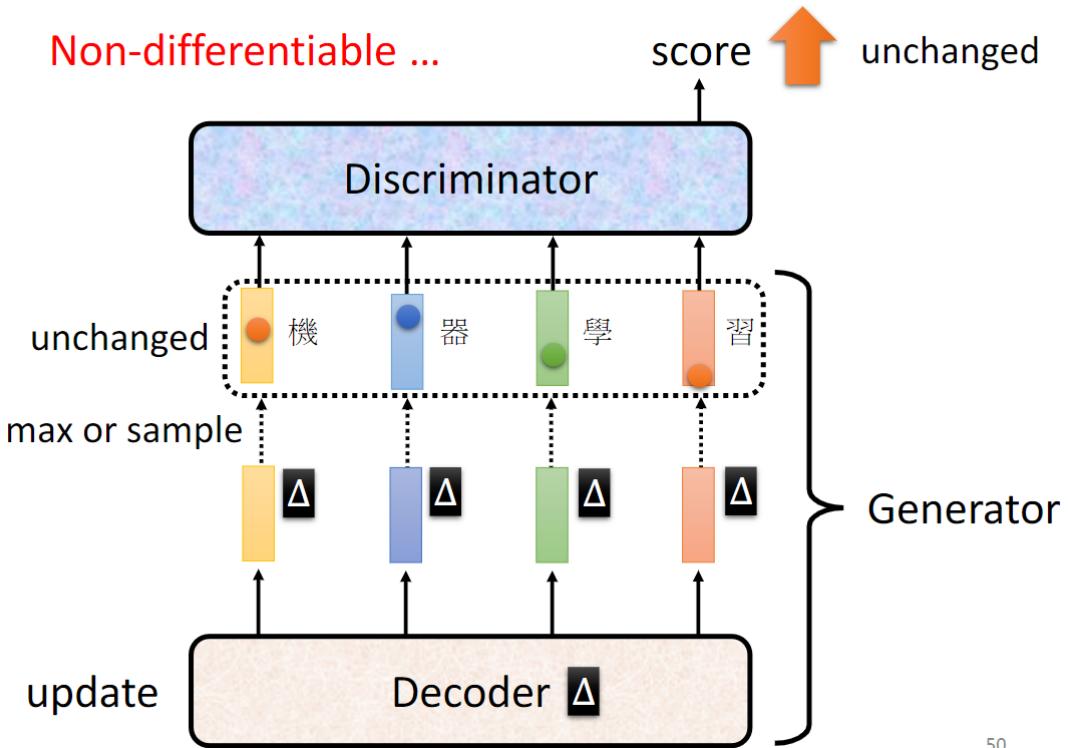
为了能够train起来，也是需要调超参数 (hyper parameter) 的。

网络上更多的Tips

- [Tips from Soumith](#)
- [Tips in DCGAN: Guideline for network architecture design for image generation](#)
- [Improved techniques for training GANs](#)
- [Tips from BigGAN](#)

GAN for Sequence Generation

课程前面部分使用GAN的样例是生成（二次元）图像，而训练GAN最难的地方是要拿GAN来生成文字。



50

同样是Generator要去生成“以假乱真”的文字可以fool到Discriminator的程度，以调整Generator (Decoder) 的参数。

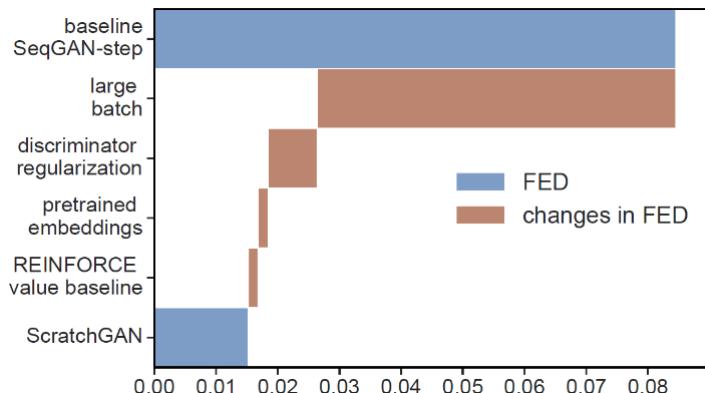
而难点在于如果要用梯度下降法去train这个Decoder (目标：让Discriminator分数越大越好)，实战中会发现我们根本做不到——原因在于Decoder某个参数的小小变化 (偏微分)，在attention机制下，分数最大的token (token取决于任务和我们的设计) 没有改变，以至于Discriminator的分数也没有改变。

attention机制下的**max or sample**下，尽管数值有在变化而取出来的token (可能是character或字符) 往往不变，导致gradient decent几乎失效；而CNN中的**max Pooling**层下，数值是一直随着变化的，故gradient decent有效。

遇到不能用gradient decent来train的问题，我们就要当作Reinforcement Learning (RL) 的问题来硬做！

恰好地，RL非常难train，GAN非常难train，导致GAN做文字生成相当困难。很长一段时间没有人能把Generator训练起来；在面对这个问题的时候，通常我们先做**Pretrain (预训练)** ——之后会讲到。

过去没有正常的方法来train这样子的GAN，直到提出[Scratch GAN](#)，from Scratch就是不需要pretrain的意思。所以它用了什么魔法？——爆调了超参数，以及各种tips (来自google的文章enmmm)
[enough hyperparameter-tuning and tips]



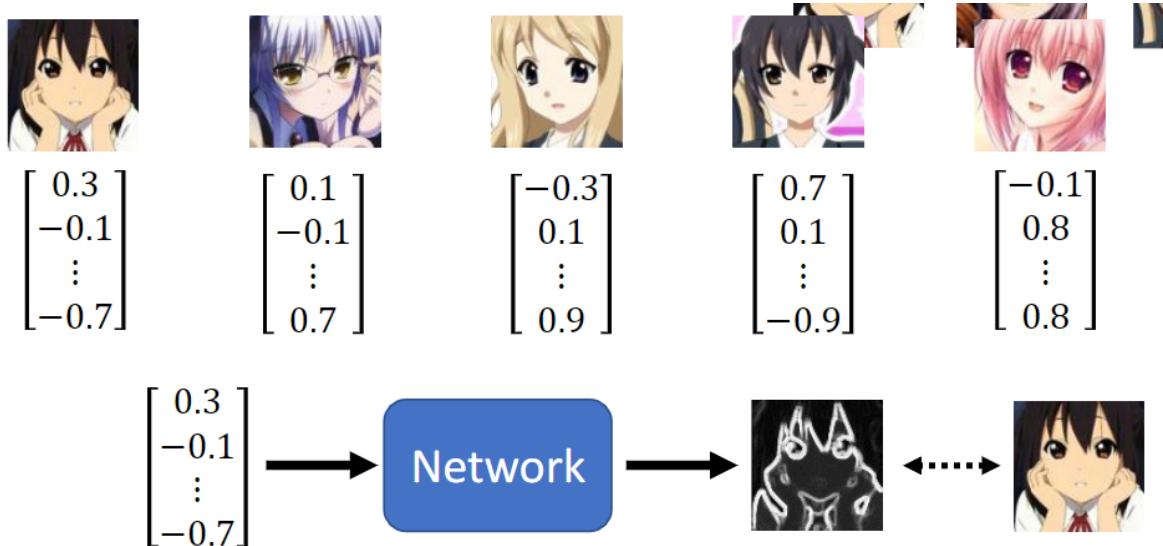
More Generative Models

GAN是最广泛使用而且performance也很好；以下的可以自行了解，train的难度也不遑多让。

- Variational Autoencoder (VAE)
- FLOW-based Model

用supervised learning来做图像生成

我们有一堆图片，每张图片用一个vector来表示，然后“硬train一发”，当然需要用到特殊的方法来安排这些训练的vectors。可参考的研究：[Generative Latent Optimization \(GLO\)](#)、[Gradient Origin Networks](#)

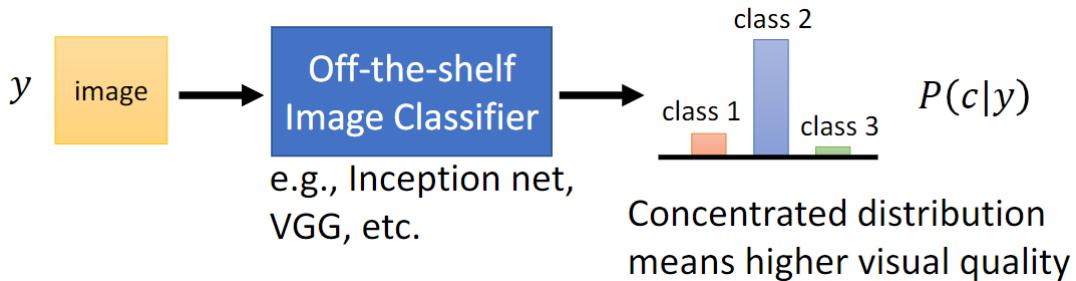


Evaluation of Generation

生成器效能评估与条件式生成。

Quality of Image

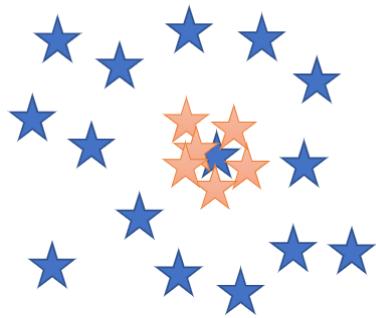
- 早期：找人来看——human evaluation is expensive(而且不客观，不稳定)
- 一个类别跑一个图像的分类系统



检验生成出来图像是否合理，用机器来验证机器。不过光用这个做法是不够的，我们需要对生成的结果的**diversity**进行考量。如果只用这个方法我们会被一个称作**Mode Collapse**的东西fool过去。

Mode Collapse (模型坍缩)：生成数量巨大后，产生效果单一

★ : real data
★ : generated data



我们可以理解为discriminator的一个盲点，generator硬打一发，就混过去了。目前有效彻底的解决方法还没有，即使是强如Google爆搜参数，也仅仅是在察觉到mode collapse后就停止train，收掉model。

还有一个问题称之为**Mode Dropping**，不太容易被侦测出来。

Mode Dropping:



依然是生成data范围太窄，出现了些模式类似的现象。

Generator
at iteration t

譬如

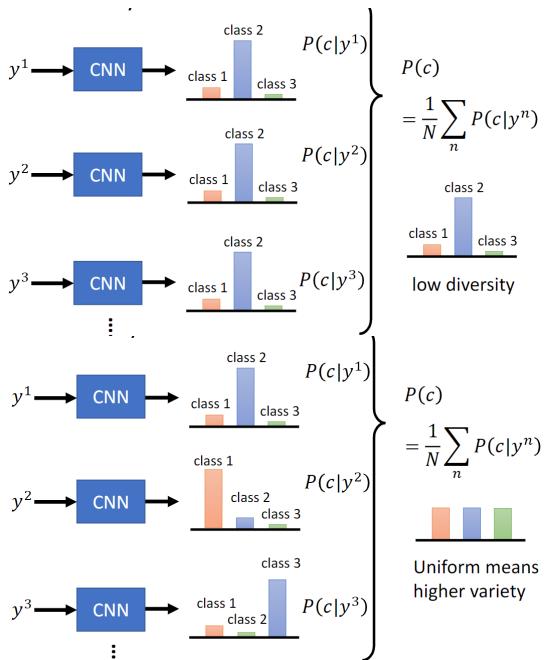
Generator
at iteration t+1



(肤色差异，同一个iteration肤色相似)

这个问题没有获得本质性解决。

这时候，我们就需要Image Classifier，验证过程：把生成所有图片丢进image Classify里面，得到各个图片的distribution（分布），把所有的distribution平均起来，看看平均的distribution是否集中起来，若是，说明多样性不足，每张图片都很类似。



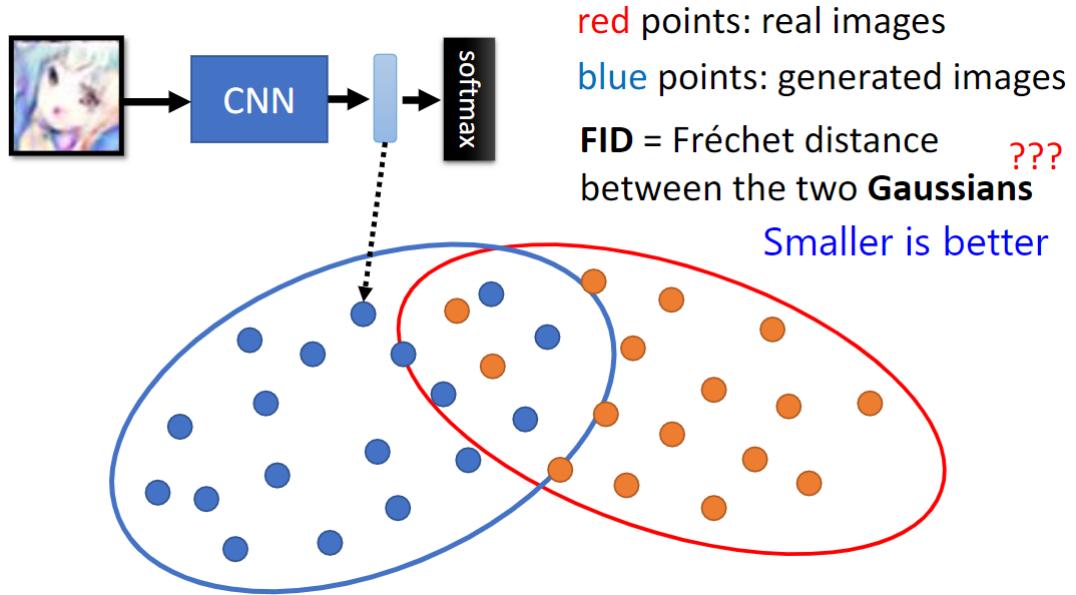
- Diversity和quality好像是有带点互斥的；分布越集中， quality越好；分布越平坦， diversity越好。但是两者范围不一样， quality看一张图片， diversity看生成的一批图片。

过去有一个常被使用的指标：**Inception Score (IS)** ——Good quality, large diversity → Large IS。使用方法：用Inception Network量一下quality，如果quality高，diversity大，那IS就高。IS不适用于衡量diversity。（每张二次元人物都是一个“人脸”）

Fréchet Inception Distance (FID)

来自<https://arxiv.org/pdf/1706.08500.pdf>

在把一张图像丢进CNN的网络的pipeline里面，可以在softmax之前截下来（最后一层Hidden Layer的输出代表图片），画出来假设（在二维平面）长这副样子



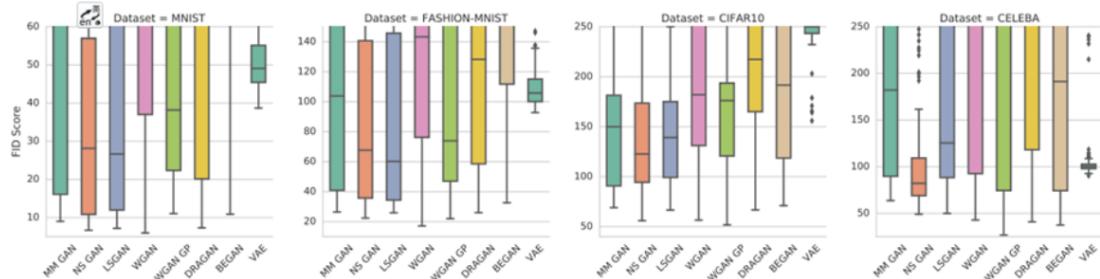
把分布当作是高斯分布 (Gaussians Distribution)，但是如果要得到准确的分布——需要大量的samples (样本) 也需要一点运算量

在作业里面，我们不仅看FID (smaller is better)，而且还要看二次元人脸侦测出来的数目 (当然越大越好) 【两个指标】

[Are GANs Created Equal? A Large-Scale Study.](#)

Google爆做，各式各样的GAN

GAN	DISCRIMINATOR LOSS	GENERATOR LOSS
MM GAN	$\mathcal{L}_D^{GAN} = -\mathbb{E}_{x \sim p_d} [\log(D(x))] + \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$	$\mathcal{L}_G^{GAN} = -\mathcal{L}_D^{GAN}$
NS GAN	$\mathcal{L}_D^{NSGAN} = \mathcal{L}_D^{GAN}$	$\mathcal{L}_G^{NSGAN} = \mathbb{E}_{\hat{x} \sim p_g} [\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_D^{WGAN} = -\mathbb{E}_{x \sim p_d} [D(x)] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$	$\mathcal{L}_G^{WGAN} = -\mathcal{L}_D^{WGAN}$
WGAN GP	$\mathcal{L}_D^{WGAN} = \mathcal{L}_D^{WGAN} + \lambda \mathbb{E}_{\hat{x} \sim p_g} [(\nabla D(\alpha x + (1 - \alpha)\hat{x}) _2 - 1)^2]$	$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$
LS GAN	$\mathcal{L}_D^{LSGAN} = -\mathbb{E}_{x \sim p_d} [(D(x) - 1)^2] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})^2]$	$\mathcal{L}_G^{LSGAN} = -\mathbb{E}_{\hat{x} \sim p_g} [(D(\hat{x}) - 1)^2]$
DRAGAN	$\mathcal{L}_D^{DRAGAN} = \mathcal{L}_D^{GAN} + \lambda \mathbb{E}_{\hat{x} \sim p_d + \mathcal{N}(0, c)} [(\nabla D(\hat{x}) _2 - 1)^2]$	$\mathcal{L}_G^{DRAGAN} = -\mathcal{L}_D^{NSGAN}$
BEGAN	$\mathcal{L}_D^{BEGAN} = \mathbb{E}_{x \sim p_d} [x - AE(x) _1] - k_t \mathbb{E}_{\hat{x} \sim p_g} [\hat{x} - AE(\hat{x}) _1]$	$\mathcal{L}_G^{BEGAN} = \mathbb{E}_{\hat{x} \sim p_g} [\hat{x} - AE(\hat{x}) _1]$



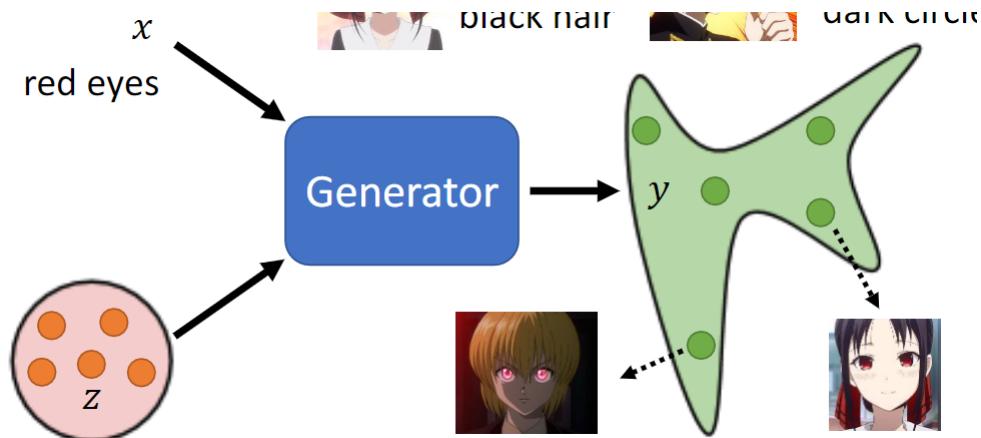
某些种类的GAN在某些架构的network上表现较好。

也会有一种现象（问题），GAN生成的图片和训练的图片几乎一致，这不是我们的目的，所以需要侦测下real data和Generated data的相似度（复制、翻转、放大）<https://arxiv.org/pdf/1511.01844.pdf>

在Pros and cons of GAN evaluation measures中给出了关于GAN的许多评价指标。

Conditional Generation

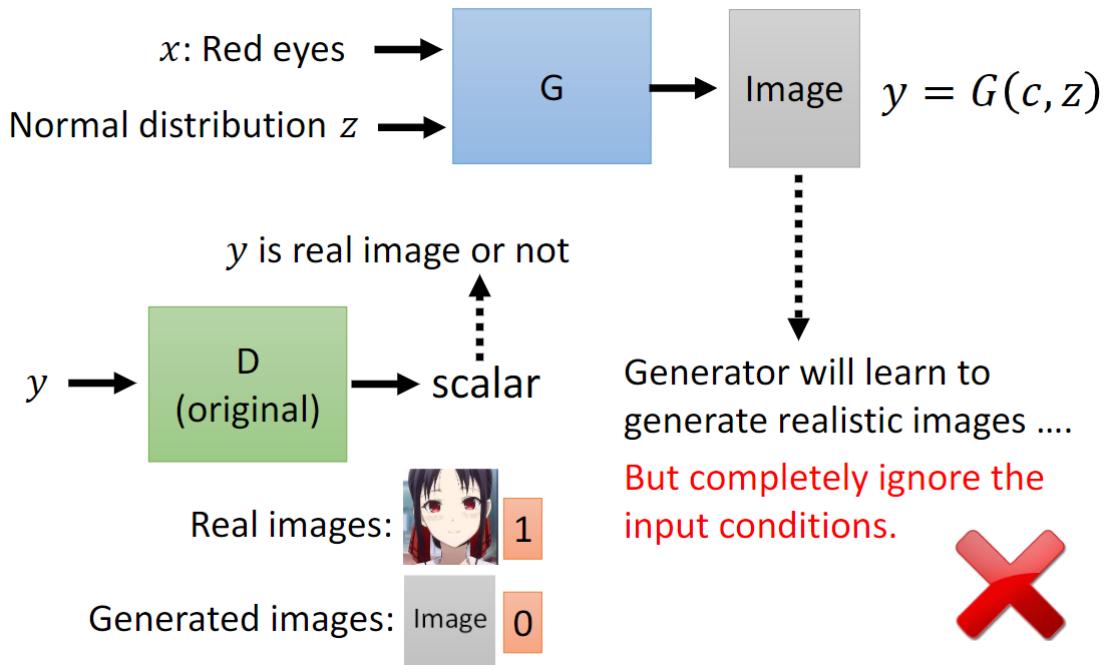
之前我们以生成二次元头像为例的GAN用的是unconditional generation。



- 可以做**Text-to-image**: 从文本中生成图片。这是一个supervised learning问题，在训练集上，我们需要搜集一些图片然后加上一些label（特征，文字的描述）
- 输入的 x 应该是一段文本（特征描述），有很多方法，可以用transformer
- 输入的 z 就是正态分布（Normal Distribution）

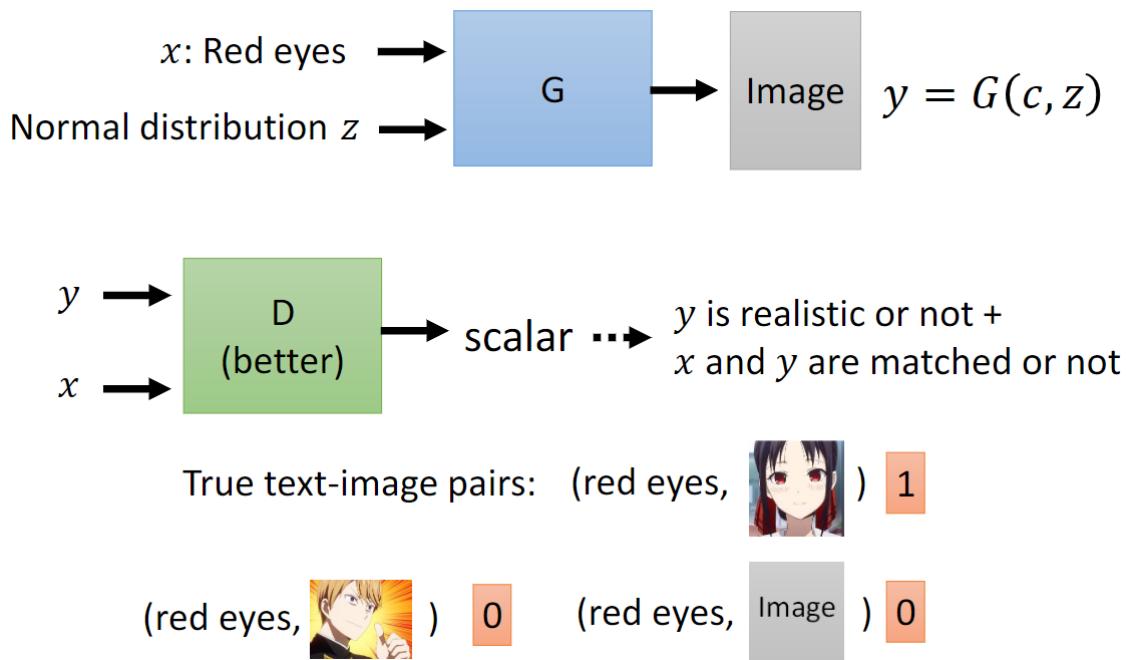
Conditional GAN

以下给出的方法是错误的，为什么？这个架构完全类似于unconditional GAN，事实上在训练当中，只要generator产生图像可以fool掉discriminator就好了，那么输入的 x 的作用完全没有体现出来。所以说，这个架构是错误的（不是我们想要的）。



以下给出正确的设计，来自<https://arxiv.org/abs/1605.05396>

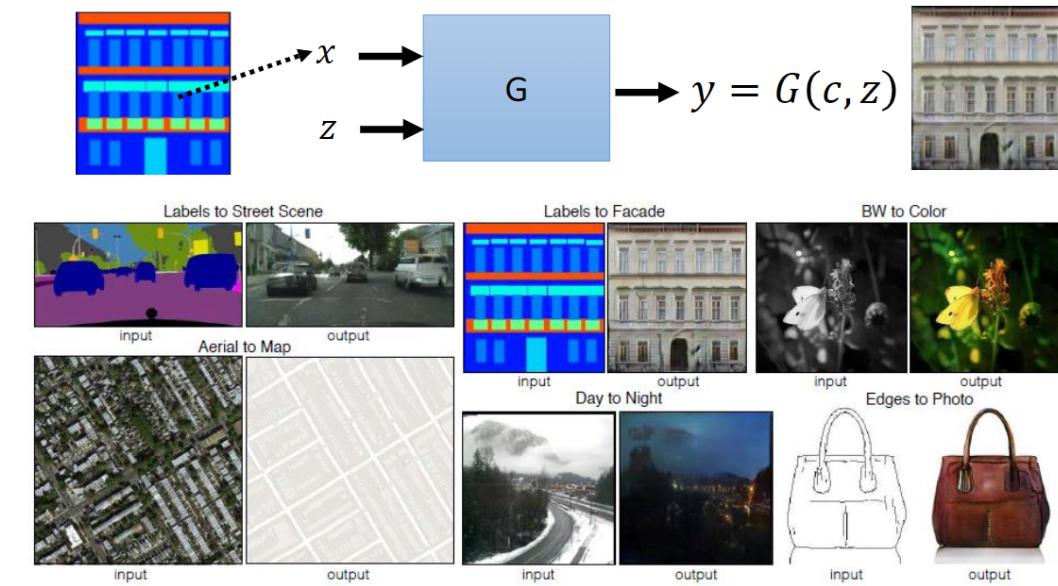
为了在训练中得到高分，不仅能生产出图像可以fool掉discriminator，而且文字label和图像需要匹配上。因此我们训练所需要的数据必须是成对出现的



不过，仅仅以上面这种positive Sample或是negative sample往往是不够的，我们还需要在训练集上增加已经这样的数据对（已经产生好的数据对，但和文字label不符）【左下】，类似是penalty Item（惩罚因子）

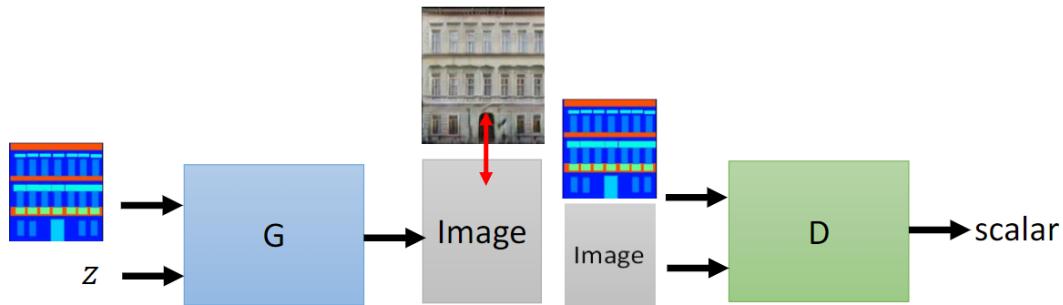
- 看一张图片生成一张图片，Image translation or pix2pix

来自<https://arxiv.org/abs/1611.07004>



具体的实现方法：

- 当然可以用supervised learning，但是前人实践证明可能效果不太好
- 用GAN来train（由于GAN创造力过于丰富了，加supervised learning【限制】可能会更好）

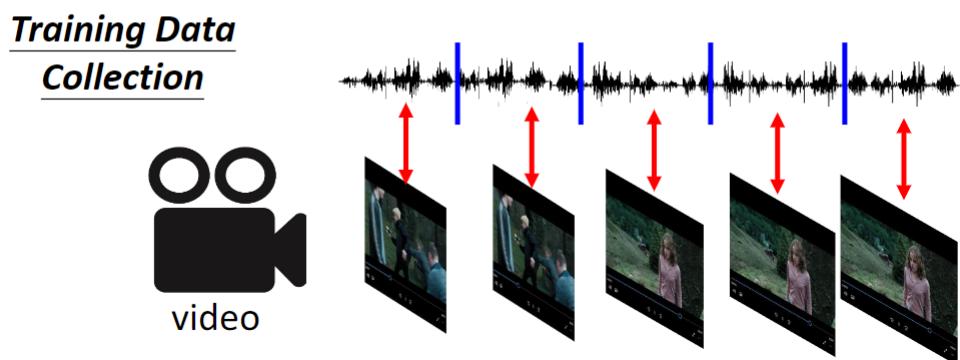


Testing:



- 一段声音生成一张图片

来自<https://arxiv.org/abs/1808.04108>



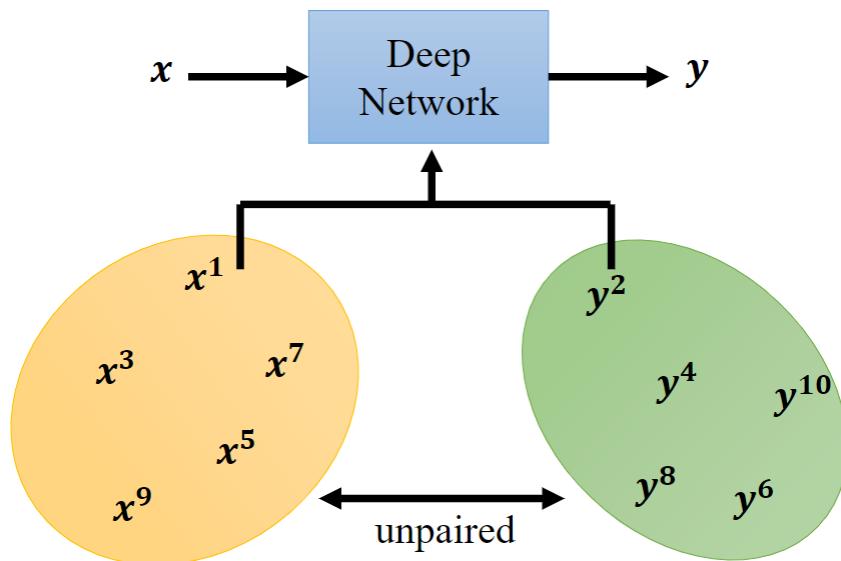
有意思的是，当声音越来越大，生成的图片似乎也会对其做出反应，仿佛机器真的学习到了声音的意味...

- 生成会动的图像，譬如Talking Head Generation

来自<https://arxiv.org/abs/1905.08233>

Learning from Unpaired Data

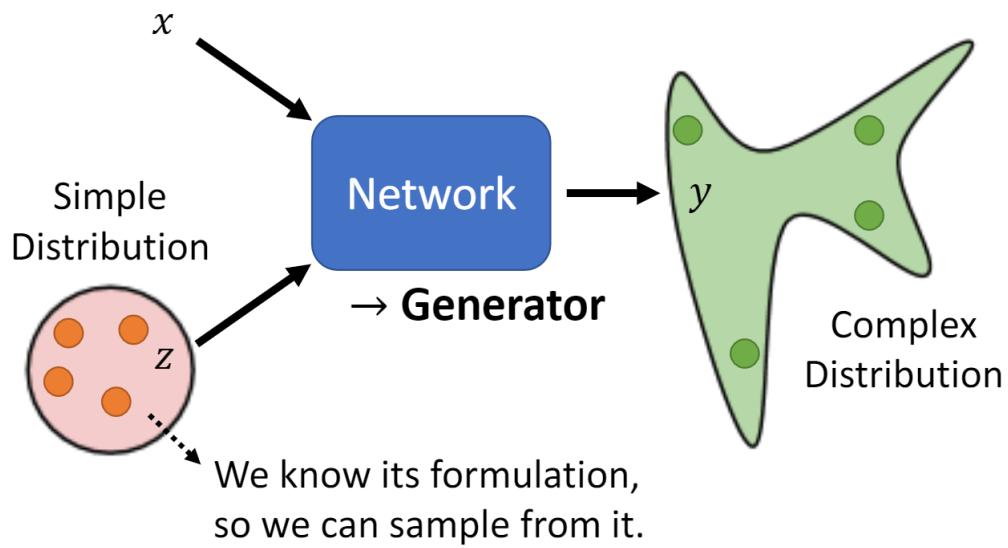
我们可能要训练这一种网络，训练需要的数据往往成对出现 (x, y) ，但是事实上我们的确有一堆 x ，也有一堆 y ，但是两者并不各个匹配成对（unpaired）



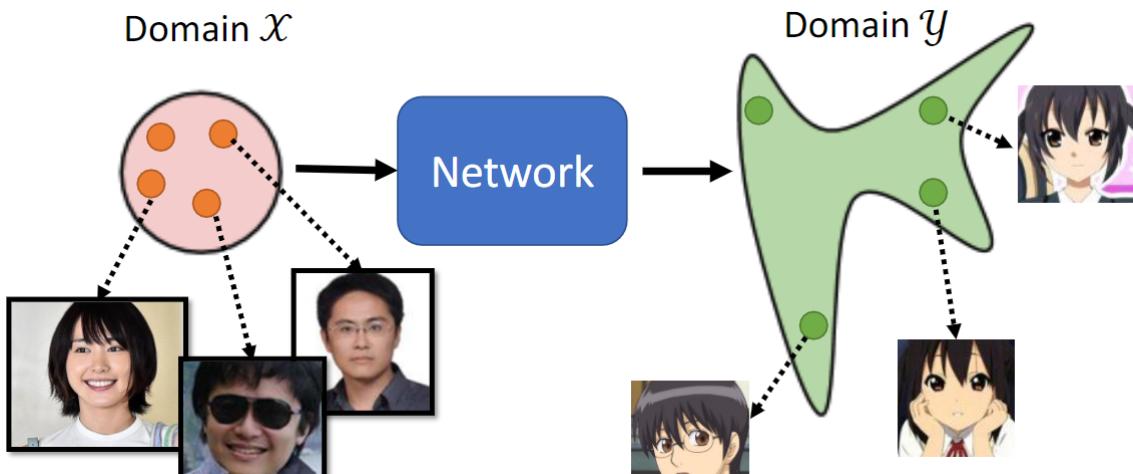
我们有没有办法在这种情况下train呢？ (without labels)

风格迁移任务：无监督学习的典型，没有任何成对的数据（资料）

之前我们讲过unconditional GAN的流程，如下



将上述流程中的 z 和 y 替换成Domain x 图片的分布和Domain y 的图片的分布



这里指出GAN可以用来寻找域的映射关系： $x \rightarrow y$; 其中 x, y 为一对 (pair) 数据，这种方法称之为
Unsupervised Conditional Generation

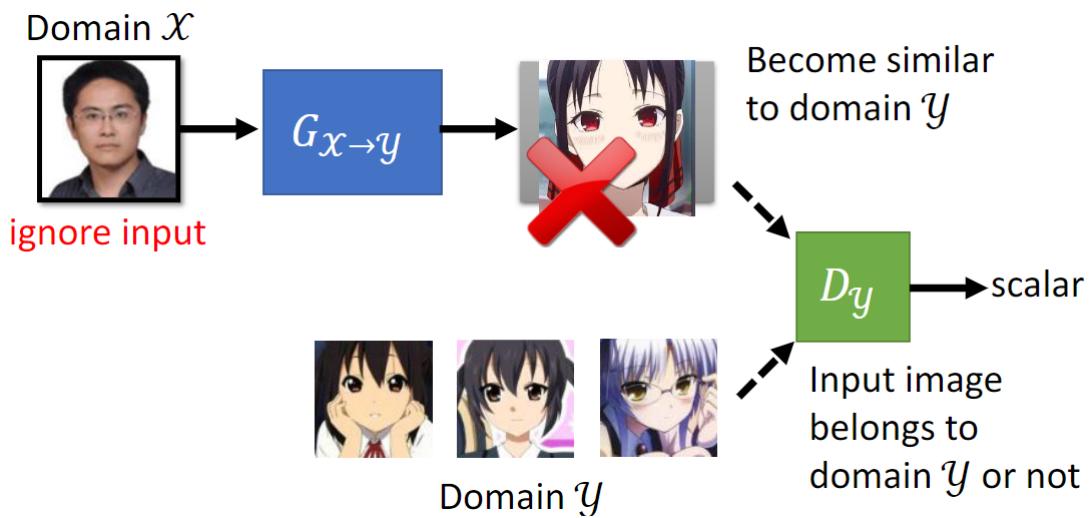
对应在数据为图像上人物通常是图像风格迁移任务，譬如真人头像-->动漫头像

Cycle GAN

Cycle GAN，即**循环生成对抗网络**，出自发表于 ICCV17 的论文《Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks》，和它的兄长Pix2Pix（均为朱大神作品）一样，用于图像风格迁移任务。

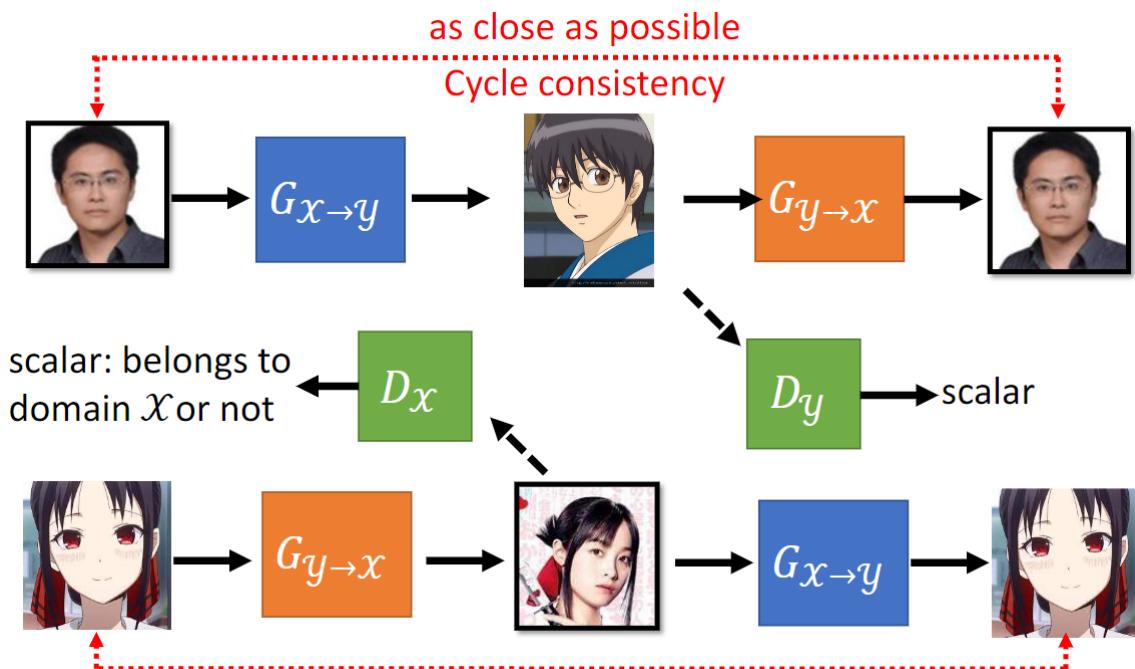
在原来的GAN里面：可以被sample的一个distribution（不一定是高斯分布）当作 x 丢尽generator，在这个任务中，我们认为可以丢尽一张需要被风格迁移的真人人脸图像，当然还要设计几个discriminator，看到是 Y domain的图就给高分，不是就给低分。

但是一般的GAN的做法显然是不够的，可能输出会产生二次元头像（discriminator会给高分）但是和输入毫无关系，这不是我们想要的。



Cycle GAN就是为了解决这个问题的一个巧妙的想法。

它要求训练两个Generator: 第一个 $G_{x \rightarrow y}$ 是把一张 X domain的图转换成一张 Y domain的图; 第二个 $G_{y \rightarrow x}$ 的作用是把 Y domain上的图还原回 X domain上的图。经过两次转换后, 输入和输出越接近越好 (两张图像流形向量的距离来衡量)



两个discriminator: D_x 要看想不想真实的人脸

相较一般的GAN的优点:

- 为了考虑到第二次的还原任务的完成——第一次转换到二次元头像不会乱做了(*^_^*)，而会考虑到输入真人头像
- 这个二次元图像输出(任务的最终目标输出)必须关联input，从而即使没有labels也有了reconstruct的可能
- 无法保证输入和输出会真的很像，因为机器可能会学到一些奇怪的mode(它唯一的限制不过就是第二次还原度较高就行)
- 事实上，GAN硬做也能做，by default会输出很像的东西

其他GAN们

类似于Cycle GAN，用来完成风格迁移任务

- [Disco GAN](#)
- [Dual GAN](#)

- Cycle GAN

以上几个都是不同的团队几乎在一样的时间（17年）提出的几乎一样的idea来完成这个任务

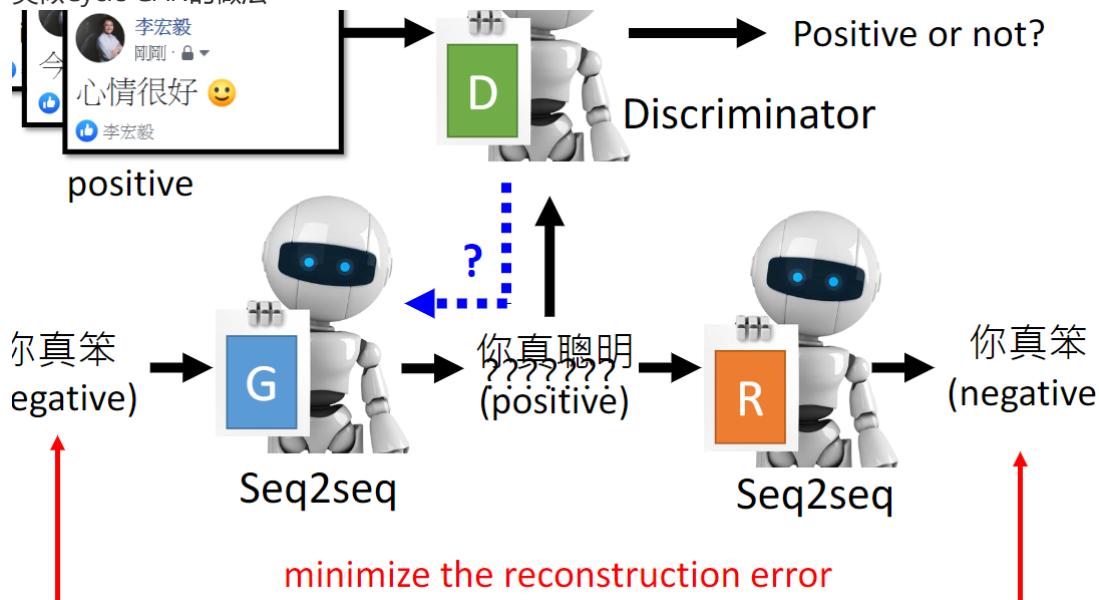
另外有一种GAN可以在多种风格间做转换：

- Star GAN

关于文字风格迁移任务 (Text Style Transfer)

seq2seq任务，具体实现和Cycle GAN几乎是一模一样的

- 首先收集一些sequence训练资料
- 类似Cycle GAN的做法



其中有几个问题：

- 1、怎么衡量两个句子或sequence的相似度？——余弦相似度
- 2、关于字符的任务用Discriminator会用问题（之前提到过），怎么办？——RL硬做

一些有趣的工作：

- Document → summary: Unsupervised Abstractive Summarization (<https://arxiv.org/abs/1810.02851>)
- Language 1 → Language 2: Unsupervised Translation (<https://arxiv.org/abs/1710.04087>、<https://arxiv.org/abs/1710.11041>)
- Audio → Text: Unsupervised ASR (<https://arxiv.org/abs/1804.00316>、<https://arxiv.org/abs/1812.09323>、<https://arxiv.org/abs/1904.04100>)

GAN完结撒花★;°☆(一▽一)/\$:.°★。