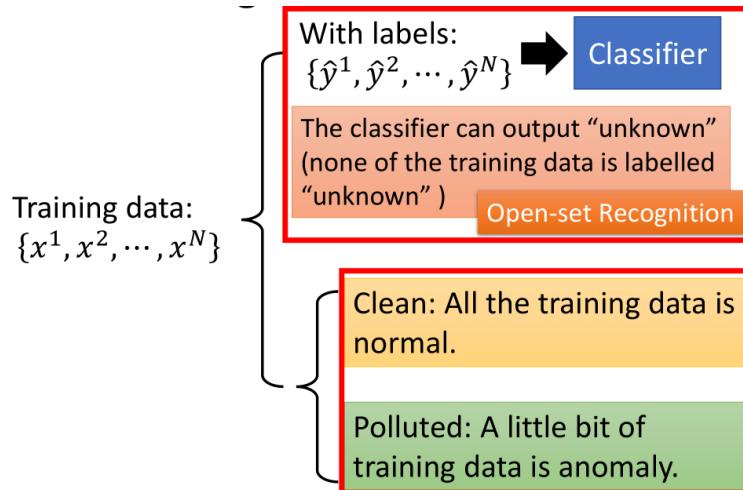


More ...

machine learning中也有其它做异常侦测的方法，比如One-class SVM，只需要正常的资料就可以训练SVM，然后就可以区分正常的还是异常的资料。

Isolated Forest，它所做的事情跟One-class SVM做的事情很像（给出正常的训练进行训练，模型会告诉你异常的资料是什么模样）。

Concluding Remarks



Generative Adversarial Network

Generative Adversarial Network

Three Categories of GAN

Typical GAN

Typical GAN要做的事情是找到一个Generator，Generator就是一个function。这个function的输入是random vector，输出是我们要这个Generator生成的图片。

举例来说：假设要机器做一个动画的Generator，那么就要收集很多动画人物的头像，然后将这些动画人物的头像输入至Generator去训练，Generator就会学会生成二次元人脸的图像。

那怎么训练这个Generator 呢，模型的架构是什么样子的呢？

最基本的GAN就是对Generator输入vector，输出就是我们要它生成的东西。我们以二次元人物为例，假设我们要机器画二次元人物，那么输出就是一张图片（高维度的向量）。Generator就像吃一个低维度的向量，然后输出一个高维度的向量。

GAN有趣的地方是：我们不只训练了Generator，同时在训练的过程中还会需要一个Discriminator。Discriminator的输入是一张图片，输出是一个分数。这个分数代表的含义是：这张输入的图片像不像我们要Generative产生的二次元人物的图像，如果像就给高分，如果不像就给低分。

Algorithm

接下来要讲的是Generator和Discriminator是咋样训练出来的，Generator和Discriminator都是neuron network，而它们的架构是什么样的，这取决于想要做的任务。举例来说：你要generator产生一张图片，那显然generator里面有很多的deconvolution layer。若要generator产生一篇文章或者句子，显然要使用RNN。

我们今天不讲generator跟discriminator的架构，这应该是取决于你想要做什么样的事情。

generator跟discriminator是某种network，训练neuron network时要随机初始化generator和discriminator的参数。

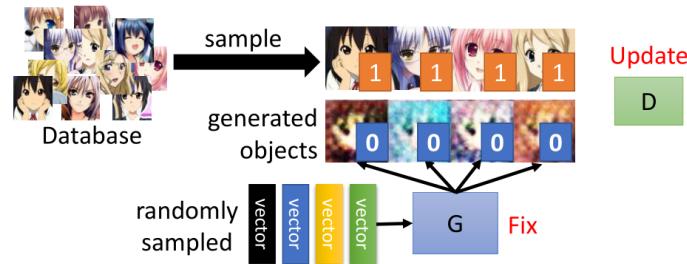
在初始化参数以后，在每一个training iteration要做两件事

step1: 固定generator，然后只训练discriminator（看到好的图片给高分，看到不好的图片给低分）。从一个资料库（都是二次元的图片）随机取样一些图片输入 discriminator，对于discriminator来说这些都是好的图片。因为generator的参数都是随机给定的，所以给generator一些向量，输出一些根本不像二次元的图像，这些对于discriminator来说就是不好的图片。接下来就可以教discriminator若看到上面的图片就输出1，看到下面的图片就输出0。训练discriminato的方式跟我们一般训练neuron network的方式是一样的。

Algorithm

- Initialize generator and discriminator G D
- In each training iteration:

Step 1: Fix generator G, and update discriminator D



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

有人会把discriminator当做回归问题来做，看到上面的图片输出1，看到下面的图片就输出0也可以。有人把discriminator当做分类的问题来做，把上面好的图片当做一类，把下面不好的图片当做另一类也可以。训练discriminator没有什么特别之处，跟我们训练neuron network或者binary classifier是一样的。唯一不同之处就是：假设我们用训练binary classifier的方式来训练discriminator时，不一样的就是binary classifier其中class的资料不是人为标注好的，而是机器自己生成。

step2: 固定住discriminator，只更新generator。

一般我们训练network是minimize 人为定义的loss function，在训练generator时，generator学习的对象不是人定的loss function/objective function，而是discriminator。你可以认为discriminator就是定义了某一种loss function，等于机器自己学习的loss function。

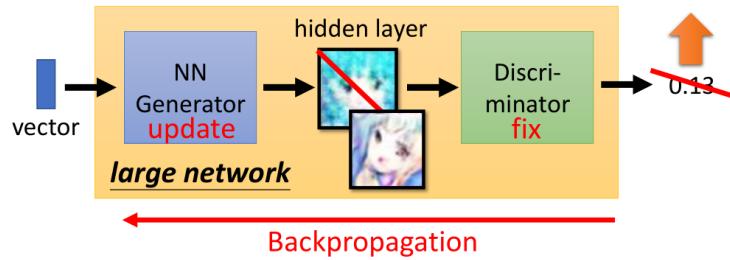
generator学习的目标就是为了骗过discriminator，我们让generator产生一些图片，在将这些图片输入进discriminator，然后discrimination就会给出这些图片一些分数。

接下来我们把generator和discriminator串在一起视为一个巨大的network。这个巨大的network输入是一个向量，输出是一个分数，在这个network中间hidden layer的输出可以看做是一张图片。

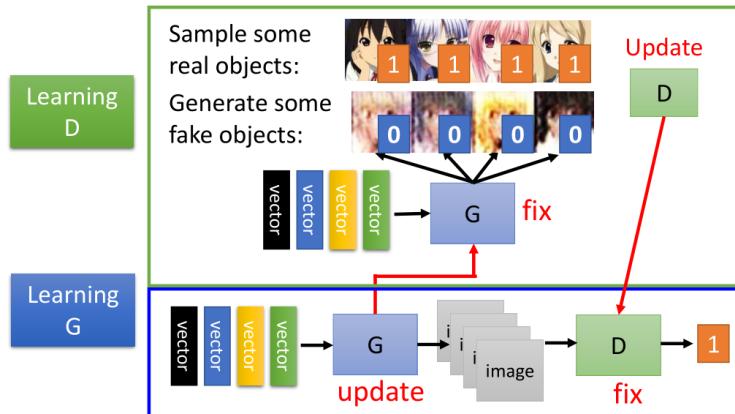
我们训练的目标是让输出越大越好，训练时依然会做backpropagation，只是要固定住discriminator，只是调整generator。调整完generator后输出会发生改变，generator新的输出会让discriminator给出高的分数。

Step 2: Fix discriminator D, and update generator G

Generator learns to “fool” the discriminator



实际上在训练时，训练discriminator一次，然后训练一次generator（固定discriminator），接下来由新的generator产生更多需要被标为0的图片，再去调discriminator，再调generator.....generator和discriminator交替训练，就做二次元人物的生成。网络上最好的二次元人物生成的结果。



GAN is hard to train...

众所周知，GAN这个技术是比较难train起来的，所以有很多人提出了更好的训练GAN的方法，WGAN, improve WGAN...

Conditional GAN

刚才是让机器随机的产生一些东西，这些不见得是我们想要的。我们更多的想要控制机器产生出来的东西。

我们可以训练一个Generator，这个generator的输入是一段文字，输出是这段文字对应的图像。举例来说：我们现在输入“Girl with red hair”，它就输出一个。根据某一个输入产生对应输出的generator被叫做Conditional GAN。

Text-to-Image

Traditional supervised approach

现在用文字产生影像作为示例，如何可以训练一个network根据文字来产生图像，最直觉的方法就是使用supervised learning。假设可以收集到文字跟影像之间的对应关系（这些图片有人标识每张图片对应的文字是什么），接下来就可以完全的套用传统的supervised learning的方法。直接训练一个network，它的输入是一段文字，输出是一张图像，希望这个输出跟原始的图像越接近越好。可以用这种方法直接训练，看到文字产生图像。

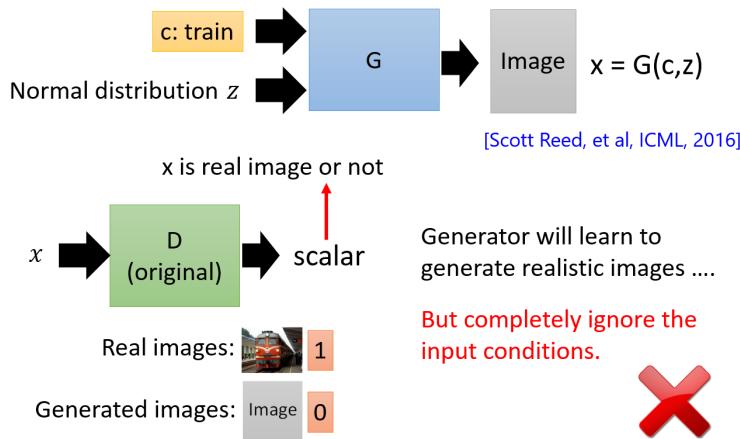
过去用这种方法（看到文字产生图像）来训练，训练出来的结果并不太好。为什么呢？举例说明：假设要机器学会画火车，但是训练资料里面有很多不同形态的火车，当network输入火车时它的正确答案有好几个，对于network来说会产生它们的平均作为输出。如果用supervised learning的方法产生出来的图像往往是非常模糊。

Conditional GAN

所以我们需要有新的技术（Conditional GAN）来做根据文字产生图像，对于Conditional GAN，我们也需要文字与图像的对应关系（supervised learning），但是它跟一般的supervised learning的训练目标是不一样的。事实上Conditional GAN也可以在unsupervised learning的情况下进行训练，后面的内容会涉及到。

Conditional GAN是跟着discriminator来进行学习的，要注意的是：在做Conditional GAN时跟一般的GAN是不一样的。

在第一部分讲一般的GAN时，discriminator是输入一张图片，然后判断它好还是不好。现在在Conditional GAN的情况下，discriminator只输入一张图片会遇到的问题是：对于discriminator想要骗过generator太容易了，它只要永远产生好的图像就行了。举例来说：永远产生一只很可爱的猫就可以骗过discriminator（对于discriminator来说那只猫是好的图片），然后就结束了。Generator就会学到不管现在输入什么样的文字，就一律忽视，都产生猫就好了，这显然不是我们想要的。



在进行Conditional GAN时往往有两个输入，discriminator应该同时看generator的输入（文字）和输出（图像），然后输出一个分数，这个分数同时代表两个含义。第一个含义是两个输入有多匹配，第二个含义是输入图像有多好。

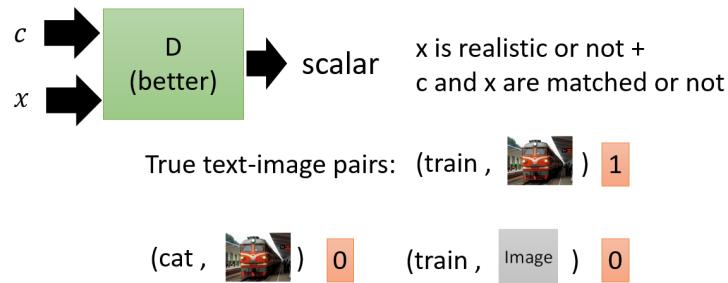
训练discriminator时要给它一些好的输入，这些是要给高分的。可以从datasets里面sample出文字与图像，告诉discriminator看到这些文字和图像应该给出高分。

对于另外一个case而言，什么情况会给出低分呢？

按照我们在第一部分讲一般GAN的想法，我们可能就是把文字输入generator，让他产生一些图片，这些文字和generator产生的图像要给予低分。

光是这么做discriminator会学到判断现在的输入是好还是不好，不管文字的部分只管图像的部分，这显然不是我们想要的。

所以在做Conditional GAN时，要给低分的case是要有两种。一个是跟一般的GAN一样是用generator生成图片；另外一个是从资料库里面sample出一些好的图片，但是给这些好的图片一些错误的文字。这时discriminator就会学到：并不是所有好的图片都是对的，如果好的图片对应都错误的文字它也是不好的。这样discriminator就会学懂文字与图像之间应该要有什么样的关系。



Sound-to-image

其实上述使用Conditional GAN根据文字生成影像的应用已经满坑满谷了，其实只要你有pair data你都可以尝试使用Conditional GAN，这里作了一个例子，训练了一个Generator，输入声音，可以输出对应的画面。

训练Conditional GAN必须要有pair data，影像跟声音的对应关系其实并不难收集，我们可以收集到很多的video，将video中的audio部分拿出来就是声音，将image frame部分拿出来就是image，这样就有了pair data，就可以训练network。举例来说：听到狗叫声，就画一只狗出来。

当听到第一行第一列声音时，机器觉得它是一条小溪；听到第二行第一列的声音时，机器觉得它是在海上奔驰的快艇；我现在担心机器是不是背了一些资料库里面的图片，并没有学会声音跟影像之间的关系，所以我们把声音调大，然后就产生了一条越来越大的瀑布。我们把快艇的声音调大，产生的就是快艇在海上快速的奔驰，水花就越来越多。机器有时候产生的结果也会非常的差。

Image-to-label

我们刚才尝试了用文字产生影像，现在反过来想，用影像来产生文字。我们将影像用在Multi-label image Classifier上，给机器看一张图片，让机器告诉我们图片有哪些物件，比如有球，球棒等等，正确答案不只有一个。我们在课堂里讲的分类问题，每一个输入都只属于每一个类别。但是在Multi-label image classifier的情况下，同一张图片可以同时属于不同的类别。

一张图片可以同时属于不同类别这件事，我们可以想成是一个生成的问题，现在Multi-label image Classifier是一个Conditional Generator，它的输入是一张图片（图片是condition），label是Generator输出。接下来就当做Conditional GAN进行训练。

	F1	MS-COCO	NUS-WIDE
The classifiers can have different architectures.	VGG-16	56.0	33.9
The classifiers are trained as conditional GAN.	+GAN	60.4	41.2
Conditional GAN outperforms other models designed for multi-label.	Inception	62.4	53.5
	+GAN	63.8	55.8
	Resnet-101	62.8	53.1
	+GAN	64.0	55.4
	Resnet-152	63.3	52.1
	+GAN	63.9	54.1
[Tsai, et al., submitted to ICASSP 2019]	Att-RNN	62.1	54.7
	RLSD	62.0	46.9

这些是一些实验的结果，其中使用F1 score来当做评价指标（分数越高代表分类越正确）。我们试了不同的architecture（从VGG-16到Resnet-152），假设现在不是用一般的训练法（nn输出和ground truth越接近越好），而是用Conditional GAN的方法时，你会发现在不同的network架构下，结果都是比较好的。为什么加上GAN的方法会比较好呢？因为加上GAN以后会考虑label和label之间的dependence。

Talking Head

根据一张图片跟人脸landmarks去产生另外一张人脸，也就是说你现在可以做：给它一张蒙娜丽莎的脸，然后在画一些人脸的landmarks，这样你就可以让蒙娜丽莎开始讲话。

<https://arxiv.org/abs/1905.08233>

Unsupervised Conditional GAN

刚才在讲Conditional GAN时我们需要输入和输出之间的对应关系，但是事实上有机会在不知道输入和输出之间的对应关系的情况下，可以教机器怎样将输入的内容转化为输出。这个技术最常见到的应用是风格转化。

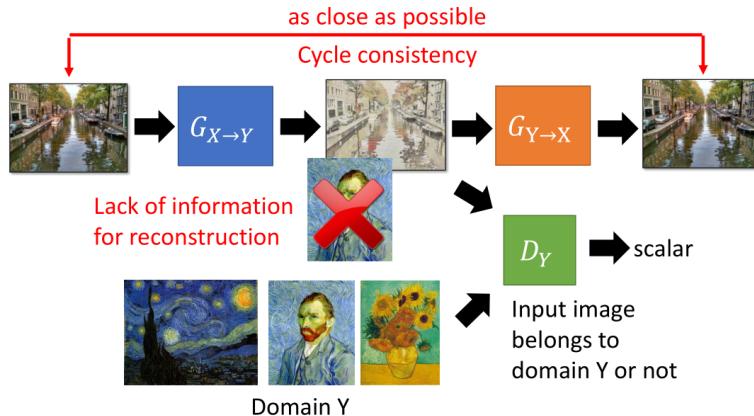
Cycle GAN

Cycle GAN想要做的事情是：训练一个generator，输入domain X（真实的画作），输出是梵高的画作。除了训练一个generator还需要训练一个discriminator，discriminator做的事情是：看很多梵高的画作，看到梵高的画作就给高分，看到不是梵高的画作就给低分。

generator为了要骗过discriminator，那我们期待产生出来的照片像是梵高的画作，光是这样做是不够的，因为generator会很快发现discriminator看的就是它的输出，那么generator就直接产生梵高的画作，完全无视输入的画作，只要骗过discriminator整个训练就结束了。

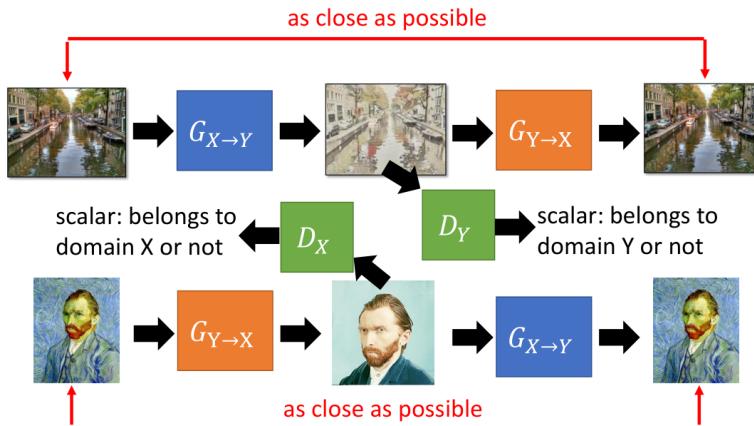
为了解决这个问题我们还需要再加上一个generator，这个generator要做的事情是根据第一个generator的输出还原原来的输入（输入一张Domain X，第一个generator将其转化为Domain Y的图，第二个generator在将其还原为Domain X，两者越接近越好）

Cycle GAN



现在加上了这个限制，第一个generator就不能够尽情的骗过discriminator，不能够直接产生梵高的自画像。如果直接转换为梵高的自画像，那么第二个generator就无法将梵高的自画像还原。所以第一个generator想办法将Domain Y，但是原来图片最重要的资讯仍然被保留了下来，那么第二个generator才会有办法将其还原。输出跟输入越接近越好，这件事叫做Cycle consistency。

Cycle GAN其实是可以双向的，我们刚才讲的是先将Domain X转换为Domain Y，再将Domain Y转换为Domain X。现在我们可以将Domain Y转换为Domain X，然后再用一个蓝色的generator将Domain X转换为Domain Y，让输出和输入越接近越好。



同样的技术不只是用在影像上，可以应用在其它领域上。举例来说：假设Domain X和Domain Y分别是不同人讲的话（语音），那就可以做语音的风格转换。假设Domain X和Domain Y是两种不同的风格文字，那就可以做文字的风格转化。假设我们把Domain X替换成负面的句子，将Domain Y换成正面的句子，我们就可以训练一个generator，可以将负面的句子转换为正面的句子。

如果直接将影像的技术套用到文字上是会有些问题，如果generator在做文字风格转换的时候，输入是一个句子，输出也是一个句子。如果输入和输出分别是句子的话，这时应该使用Seq2seq model作为网络架构。在训练时仍然很期待使用Backpropagation将Seq2seq和discriminator串在一起，然后使用backpropagation固定discriminator，只训练generator，希望输出的分数越接近越好。

但在文字上没有办法直接使用bachpropagation，因为Seq2seq model输出的是离散的seq，是一串token。原来我们将generator的输看做一个巨大network的hidden layer，那是因为在影像上generator的输出是连续的，但是在文字上generator的输出是离散的（不能微分）。

如何解决呢，在文献上有各式各样的解决办法。

Three Categories of Solutions

Gumbel-softmax: [Matt J. Kusner, et al, arXiv, 2016]

Continuous Input for Discriminator: [Sai Rajeswar, et al., arXiv, 2017][Ofir Press, et al., ICML workshop, 2017][Zhen Xu, et al., EMNLP, 2017][Alex Lamb, et al., NIPS, 2016][Yizhe Zhang, et al., ICML, 2017]

Reinforcement Learning: [Yu, et al., AAAI, 2017][Li, et al., EMNLP, 2017][Tong Che, et al, arXiv, 2017][Jiaxian Guo, et al., AAAI, 2018][Kevin Lin, et al, NIPS, 2017][William Fedus, et al., ICLR, 2018]

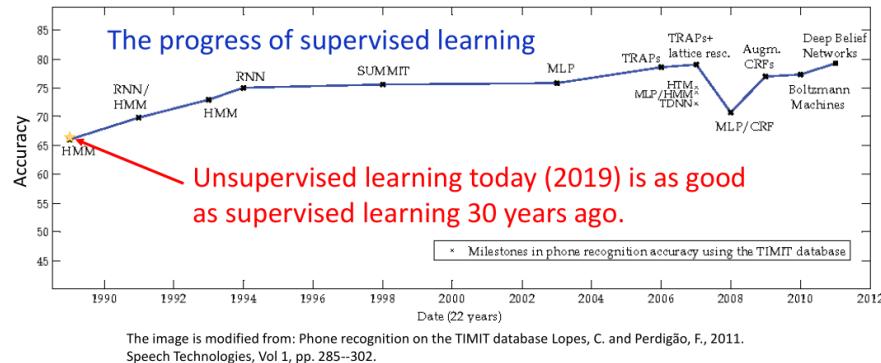
Speech Recognition

Unsupervised Conditional GAN其实除了风格转换以外还可以做其它的事情，比如说：Unsupervised 语音辨识。我们在做语音辨识时通常是 supervised，也就是说我们要给机器一大堆的句子，还要给除句子对应的文字是什么，这样的句子收集上万小时，然后期待机器自动学会语音辨识。但是世界上语言有7000多种，其实不太可能为每一种语言都收集训练资料。

所以能不能想象语音辨识能不能是Unsupervised的，也就是说我们收集一大堆语言，一大堆文字，但是我们没有收集语言跟文字之间的对应关系。机器做的就是听一大堆人讲话，然后上面读一大堆文章，然后期待机器自动学会语音辨识。

现在有很多人很笼统的问一个问题：语音辨识可以做到什么样的错误率、正确率，这个问题我都是没有办法去回答的，这个问题好像是你的数学会考多少分，都没有说是考哪一门数学。所以你今天要问一个语言辨识系统的正确率有多少，你应该问是拿什么样的资料去测试它。

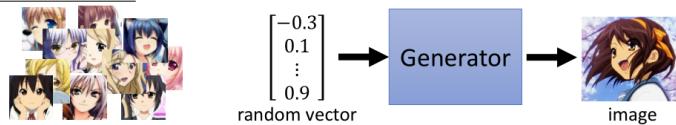
这张图讲的是supervised learning过去变化的情形（纵轴是正确率，横纵是时间），Unsupervised learning（没有提供给机器文字与语音之间的对应关系）可以做到跟三十年前supervised learning的结果一样好。



Concluding Remarks

Three Categories of GAN

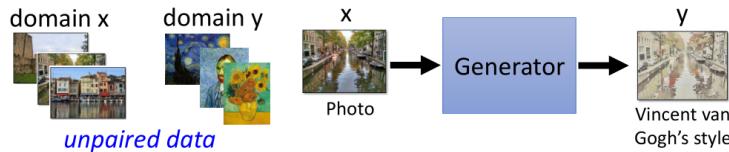
1. Typical GAN



2. Conditional GAN



3. Unsupervised Conditional GAN



Introduction of Generative Adversarial Network

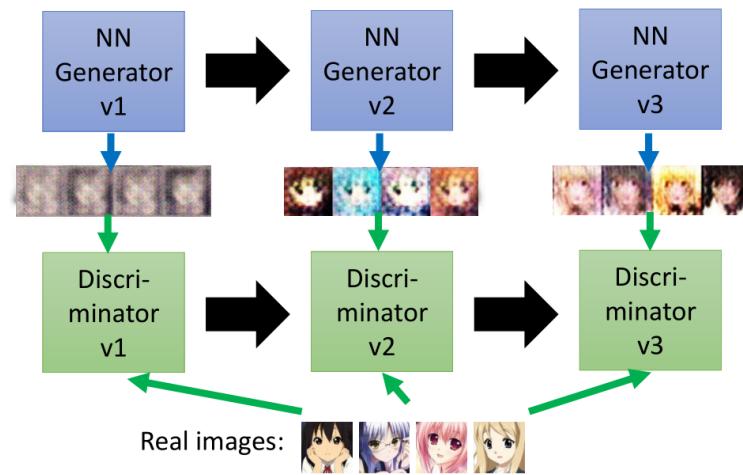
Basic Idea of GAN

Generator v.s. Discriminator

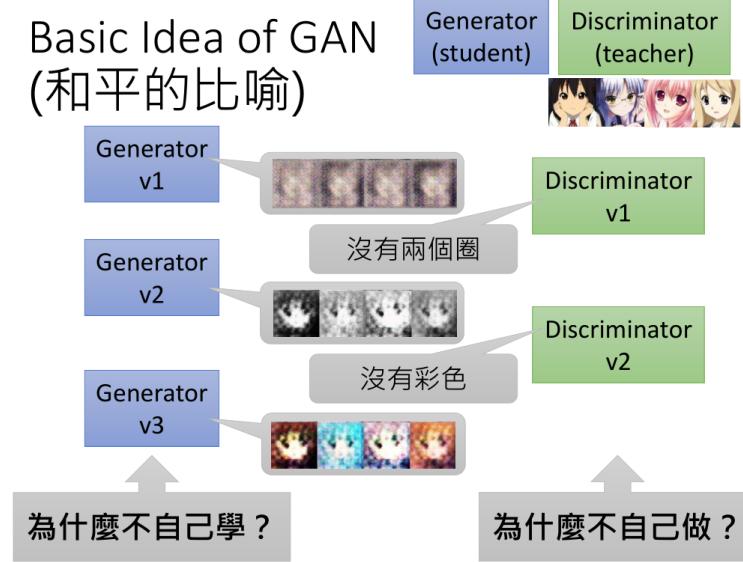
名为敌人，实为朋友

Basic Idea of GAN

This is where the term
"adversarial" comes from.
 You can explain the process
 in different ways.....



Basic Idea of GAN (和平的比喩)

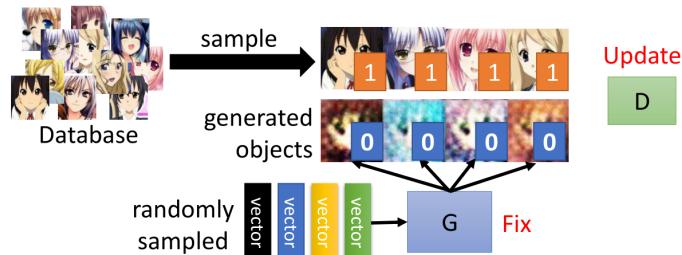


Algorithm

Algorithm

- Initialize generator and discriminator
- In each training iteration:

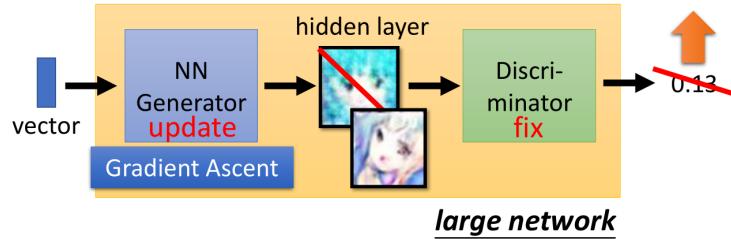
Step 1: Fix generator G, and update discriminator D



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

Step 2: Fix discriminator D, and update generator G

Generator learns to “fool” the discriminator



Algorithm Initialize θ_d for D and θ_g for G

- In each training iteration:

- Sample m examples $\{x^1, x^2, \dots, x^m\}$ from database
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}, \tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_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 m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Update generator parameters θ_g to maximize
 - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log (D(G(z^i)))$
 - $\theta_g \leftarrow \theta_g + \eta \nabla \tilde{V}(\theta_g)$

Learning
D

Learning
G

GAN as structured learning

Structured Learning

Machine learning is to find a function f

$$f : X \rightarrow Y$$

Regression: output a scalar

Classification: output a “class” (one-hot vector)

1	0	0	0	1	0	0	0	1
Class 1			Class 2			Class 3		

Structured Learning/Prediction: output a sequence, a matrix, a graph, a tree

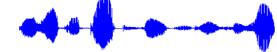
Output is composed of components with dependency

$$\text{Output Sequence} \quad f : X \rightarrow Y$$

Machine Translation

X : “機器學習及其深層與
結構化”
(sentence of language 1) Y : “Machine learning and
having it deep and structured”
(sentence of language 2)

Speech Recognition

X : 
(speech)

Y : 感謝大家來上課
(transcription)

Chat-bot

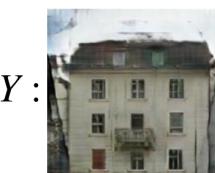
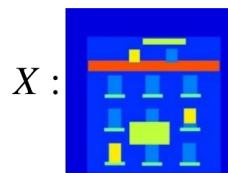
X : “How are you?”
(what a user says)

Y : “I’m fine.”
(response of machine)

Output Matrix

$$f : X \rightarrow Y$$

Image to Image



Colorization:



Ref: <https://arxiv.org/pdf/1611.07004v1.pdf>

Text to Image

X : “this white and yellow flower
have thin white petals and a
round yellow stamen”

Y :



ref: <https://arxiv.org/pdf/1605.05396.pdf>

Why Structured Learning Challenging?

One-shot/Zero-shot Learning

- In classification, each class has some examples.
- In structured learning,
 - If you consider each possible output as a “class”
 - Since the output space is huge, most “classes” do not have any training data.
 - Machine has to create new stuff during testing.
 - Need more intelligence

Machine has to learn to do **planning**

- Machine generates objects component-by-component, but it should have a big picture in its mind.
- Because the output components have dependency, they should be considered globally.
- 在 Structured Learning 里面, 真正重要的是component和component之间的关系

Structured Learning Approach

综上, Structured Learning 有趣而富有挑战性的问题。而GAN 他其实是一个 Structured Learning 的 solution。

Structured Learning有两种方法, Bottom Up和Top Down, 前者是每个component分开产生, 缺点是容易失去大局观, 后者是产生一个完整的object后再从整体来看这个object好不好, 缺点是这个方法很难做generation

Generator 可以视为是一个 Bottom Up 的方法, Discriminator 可以视为是一个 Top Down 的方法, 把这两个方法结合起来就是 Generative Adversarial Network, 就是 GAN。

Generator

Learn to generate the object at the component level



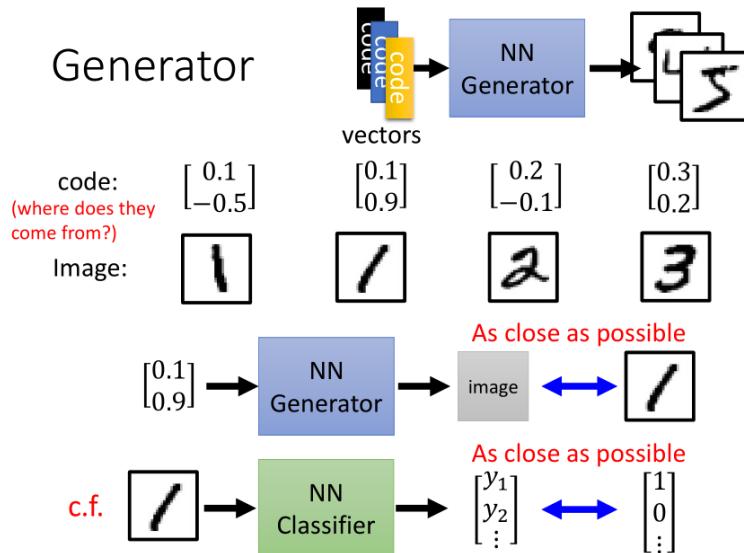
Discriminator

Evaluating the whole object, and find the best one



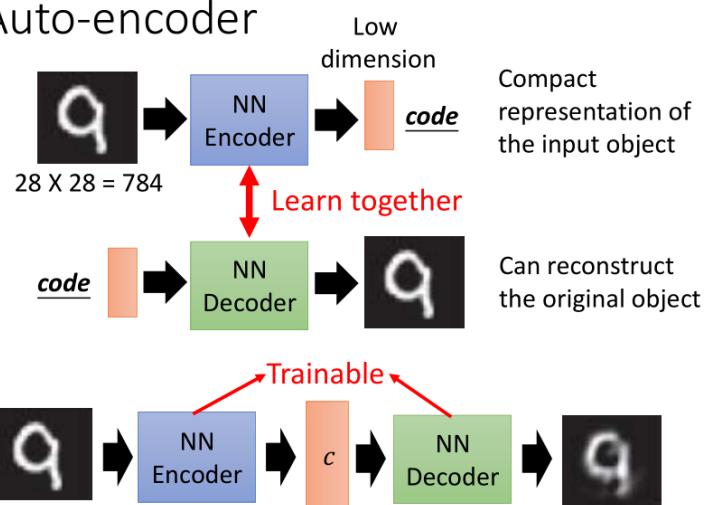
Can Generator learn by itself?

事实上Generator是可以自己学的。在传统的Supervised Learning里面，给network input跟output的pair，然后train下去就可以得到结果。搜集一堆图片，并给每张图片assign一个vector即可。输入是一个vector，输出是图片（一个很长的向量）。因此NN Generator和NN Classifier其实可以用同样的方式来train（两者输入输出都是向量）。

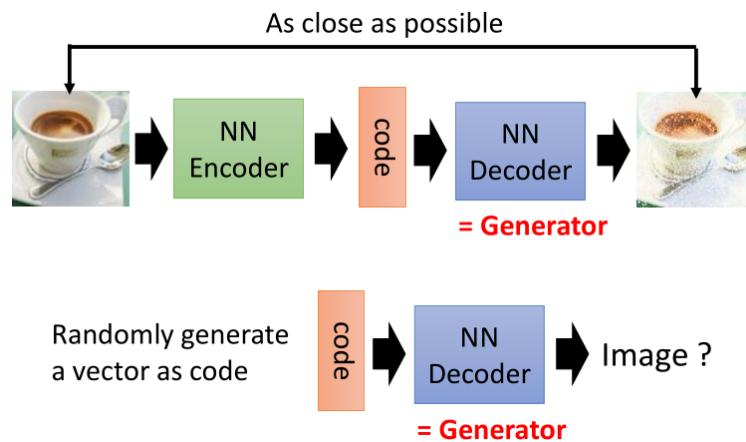


问题在于如何产生input vector, Encoder in auto-encoder provides the code

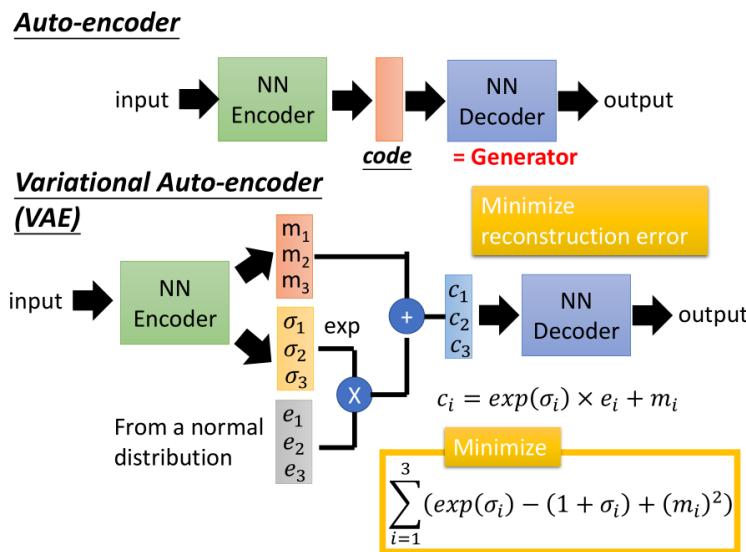
Auto-encoder



Decoder 其实就是我们要的 generator，随便丢一些东西就会 output 你想要的 object



也可以用VAE把decoder train 的更加稳定



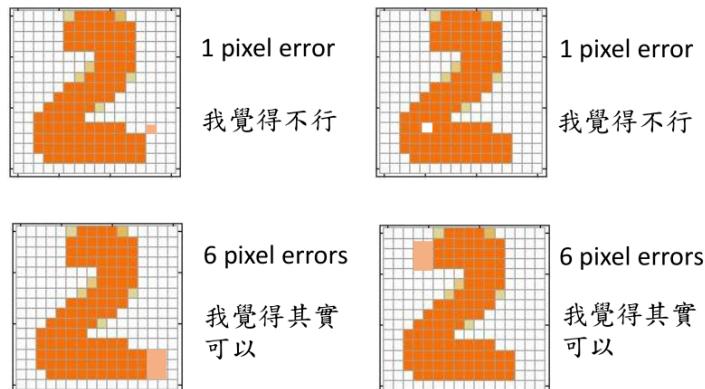
至于code的维度，是需要自己决定的，train 不起来，增加 dimension 会有效，但是增加 dimension 以后未必会得到你要的东西，因为 train 的是一个 Auto-encoder，训练的目标是要让 input 跟 output 越接近越好，要达到这个目标其实非常简单，把中间那个 code 开得跟 input 一样大，就不会有任何 loss，因为 machine 只要学着一直 copy 就好了，但这个并不是我们要的结果。虽然说 input 的 vector 开得越大 loss 可以压得越低，但 loss 压得越低并不代表 performance 会越好。

What do we miss?

It will be fine if the generator can truly copy the target image.

What if the generator makes some mistakes

- Some mistakes are serious, while some are fine.



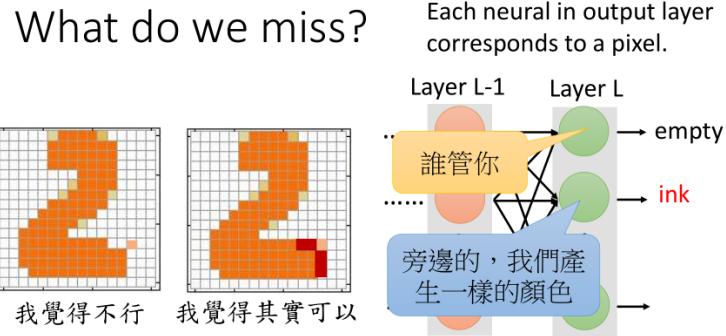
不能够单纯的去让output跟目标越像越好

在 Structured Learning 里面 component 和 component 之间的关系是非常重要的，但一个 network 的架构其实没有那么容易让我们把 component 和 component 之间的关系放进去。

Although highly correlated, they cannot influence each other.

Need deep structure to catch the relation between components.

根据经验，如果有同样的 network，一个用 GAN train，一个用 Auto-encoder train。往往就是 GAN 的那个可以产生图片，Auto-encoder 那个需要更大的 network 才能够产生跟 GAN 接近的结果。因为要把 correlation 考虑进去会需要比较深的 network。



The relation between the components are critical.

Although highly correlated, they cannot influence each other.

Need deep structure to catch the relation between components.

Can Discriminator generate?

可以，但很卡

Discriminator也被称为Evaluation function, Potential Function, Energy Function ...

Discriminator相较于Generator来说，考虑component和component之间的correlation比较容易，检查correlation对不对是比较容易的。因为是产生完一张完整的 image 以后再把这张 image 丢给 discriminator。

如何用Discriminator生成？

Suppose we already have a good discriminator
 $D(x)$...

Inference

- Generate object \tilde{x} that

$$\tilde{x} = \arg \max_{x \in X} D(x)$$

Enumerate all possible x !!!

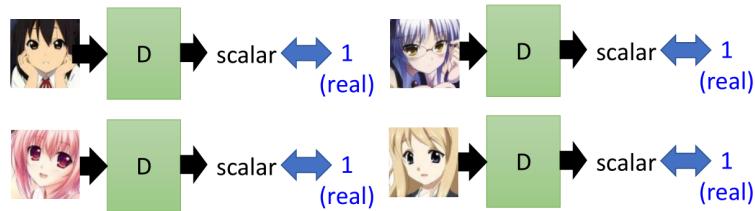
It is feasible ???

How to learn the discriminator?

Training

如何训练Discriminator？

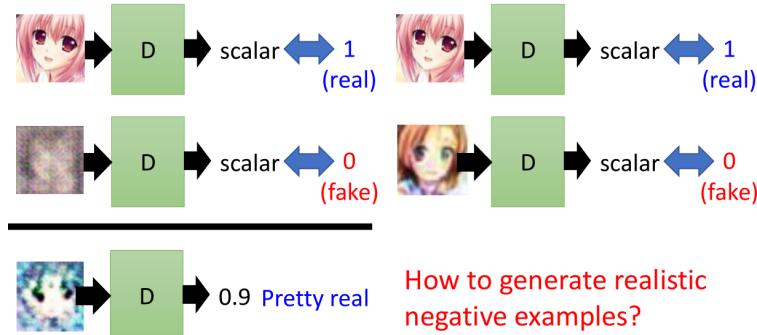
- I have some real images



Discriminator only learns to output "1" (real).

Discriminator training needs some negative examples.

- Negative examples are critical.



从哪里找 negative example 非常关键。如果找出来的 negative example 非常的差，你就跟机器说人画的就是好的，就是要给高分，然后随机产生一大堆的 noise，这些 noise 就是要给它低分。对机器来说当然可以分辨这两种图片的差别，但之后给它左下角这种图片，也许画得很难看，但是它觉得这个还是比 noise 好很多，也会给它高分，这个不是我们要的。

假设可以产生非常真实的 negative example，但还是有些错，比如说两只眼睛的颜色不一样，你可以产生非常好的 negative example，这样 discriminator 才能够真的学会鉴别好的 image 跟坏的 image。

现在问题就是怎么产生这些非常好的 negative example。要产生这些好的 negative example也需要一个很好的 process 去产生这些 negative example，现在就是不知道怎么产生 image 才要 train model，这样就变一个鸡生蛋，蛋生鸡的问题。要有好的 negative example 才能够训练 discriminator，要有好的 discriminator 才能够帮我们找出好的 negative example。

实际上可以迭代训练

General Algorithm



- Given a set of **positive examples**, randomly generate a set of **negative examples**.

- In each iteration**

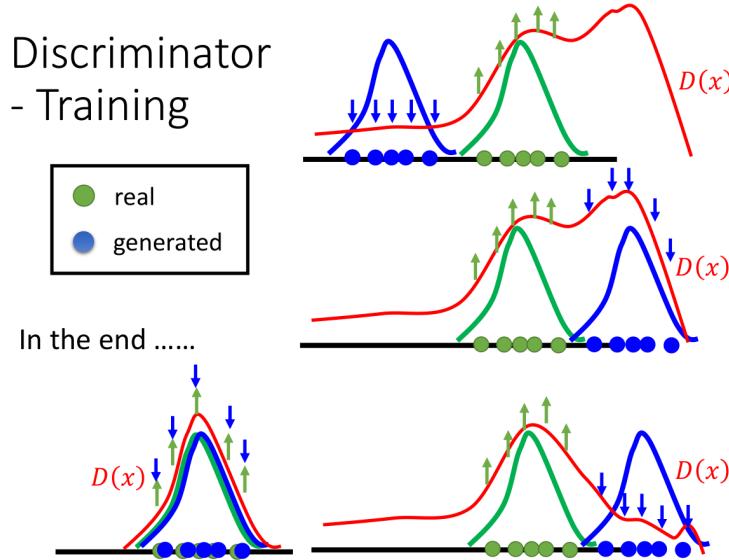


- Learn a discriminator D that can discriminate positive and negative examples.



- Generate negative examples by discriminator D

$$\tilde{x} = \arg \max_{x \in X} D(x)$$



Structured Learning and Graphical Model

有人真的拿 discriminator 做生成吗?有的, 其实有满坑满谷的工作都是拿 discriminator 来做生成。

假设你熟悉整个 Graphical Model 的 work 的话, 仔细想一下, 刚才讲的那个 train discriminator 的 process 其实就是 general 的 Graphical Model 的 training。只是在不同的 method 里面讲法会略有不同而已。

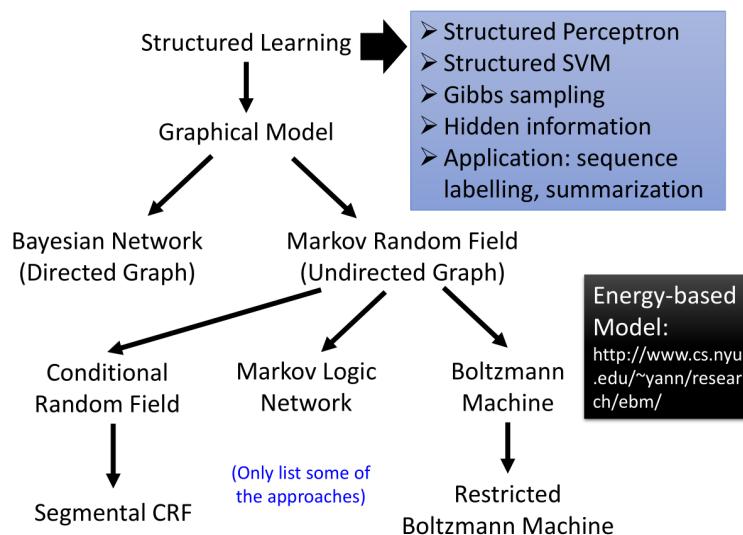
Graphical Model 其实就是 Structured Learning 的一种, Graphical Model 里面又分成很多类, 比如说有 Bayesian Network、Markov Random Field 等等。

Structured Learning 的技术里面, 其中非常具有代表性的东西就是 Graphical Model

在 Markov Random Field、Bayesian Network 里面定一个 graph, 这个 graph 上面有一个 Potential Function, 这个东西就是你的 discriminator。你输入你的 observation, 那个 graph 会告诉你这组 data 产生出来的机率有多少, 那个机率就是 discriminator assign 的分数。

Graphical Model 里面的那个 graph、你的 Potential Function、你的 Markov Random Field、你的 Bayesian Network 其实就是 discriminator。

再回想一下当你去 train Structured SVM、train Graphical Model 的时候, train 种种和 Structured Learning 有关的技术的时候是不是 iterative 的去 train 的。你做的事情是不是用 positive 用 negative example 训练出你的 model, 接下来用 model sample 出 negative example 再去 update model。就跟我刚才讲的 training discriminator 的流程其实是一样的, 只是把同样的事情换个名词来讲, 让你觉得不太一样而已。但你仔细想想, 它们就是同一回事。



Generator v.s. Discriminator

Generator

- Pros
 - Easy to generate even with deep model
- Cons
 - Imitate the appearance
 - Hard to learn the correlation between components

Discriminator

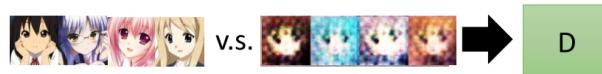
- Pros
 - Considering the big picture
- Cons
 - Generation is not always feasible (不知道如何解argmax)
 - Especially when your model is deep
 - How to do negative sampling?

Generator + Discriminator

General Algorithm



- Given a set of **positive examples**, randomly generate a set of **negative examples**.
- In each iteration
 - Learn a discriminator D that can discriminate positive and negative examples.



- Generate negative examples by discriminator D

$$\boxed{G \rightarrow \tilde{x}} = \boxed{\tilde{x} = \arg \max_{x \in X} D(x)}$$

GAN不一样的就是我们有了 generator, generator 就是取代了这个 arg max 的 problem。

本来要一个 algorithm 来解这个 arg max 的 problem, 往往我们都不知道要怎么解。但现在用 generator 来产生 negative example, generator 它就可以产生出 \tilde{x} , 即可以让 discriminator 给它高分的 image。

所以可以想成 generator 在学怎么解 arg max 这个 problem。学习的过程中就是在学怎么产生 discriminator 会给高分的那些 image。

过去是解这样一个 optimization 的 problem, 现在不一样, 是用一个 intelligent 的方法, 用一个 network 来解这个 arg max 的 problem.

Benefit of GAN

From Discriminator's point of view

- Using generator to generate negative samples

$$G \rightarrow \tilde{x} = \arg \max_{x \in X} D(x)$$

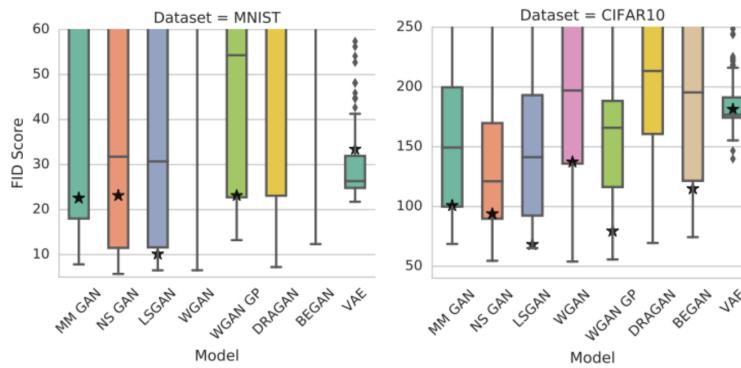
efficient

From Generator's point of view

- Still generate the object component-by-component
- But it is learned from the discriminator with global view.

从 discriminator 的角度来看，过去不知道怎么解 $\arg \max$ 的 problem，现在用 generator 来解 $\arg \max$ 的 problem，显然是比这个方法更加有效，而且更容易一般化。

对 generator 来说，它在产生 object 的时候仍然是 component by component 的产生，但是得到的 feedback 不再是 L1 L2 的 loss，不再是 pixel by pixel 的去算两张图片的相似度。它的 loss 将是来自于一个有大局观的 discriminator。希望通过 discriminator 带领，generator 可以学会产生有大局观的结果。



FID [Martin Heusel, et al., NIPS, 2017]: Smaller is better

这是来自于 Google 的一篇 paper，这篇 paper 主要的内容是想要比较各种不同 GAN 的技术。它得到的结论是所有不同的 GAN 其实 performance 都差不多。

这个纵轴是 FID Score，FID Score 越小代表产生出来的图片越像真实的图片

对于不同的 GAN 它们都试了各种不同的参数，GAN 在 training 的时候是非常的 sensitive 的，往往可能只有特定某一组参数才 train 得起来，它会发现 GAN 用不同的参数它的 performance 有一个非常巨大的 range。

如果比较 VAE 跟这些 GAN 的话可以发现，VAE 倒是明显的跟 GAN 有非常大的差别。首先 VAE 比较稳，给不同的参数，VAE 的分数非常的集中，虽然比较稳，但它比较难做到最好，所以比较每一个 model 可以产生的最好的结果的话，VAE 相较于 GAN 还是输了一截的。

Conditional Generation by GAN

Text-to-Image

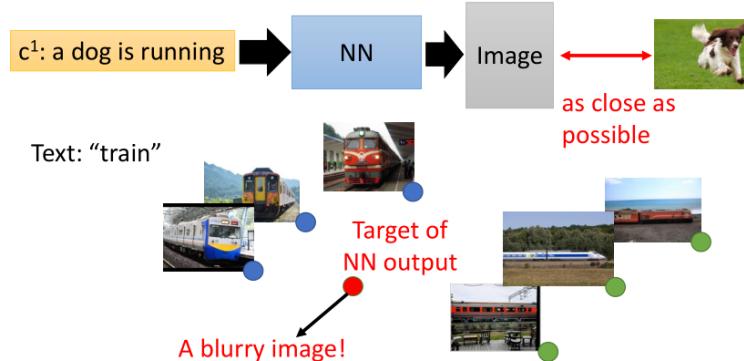
Traditional supervised approach

A blurry image

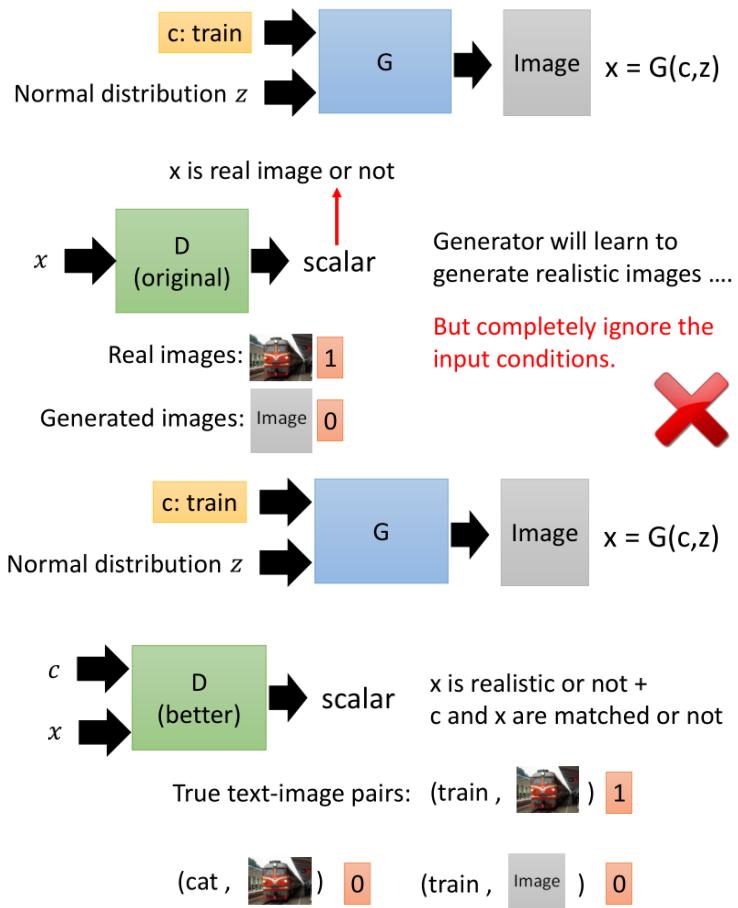
Text-to-Image



- Traditional supervised approach



Conditional GAN



Algorithm

- In each training iteration
- Learning D
 - 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 θ_d to maximize

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

$$\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$$

- Learning G

- 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 θ_g to maximize

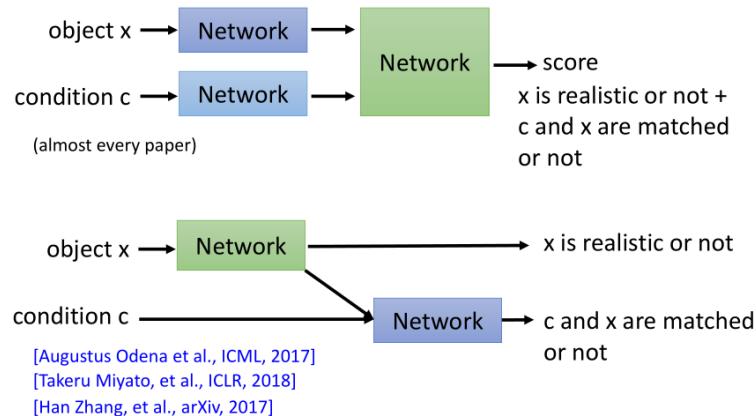
$$\begin{aligned}\tilde{V} = & \frac{1}{m} \sum_{i=1}^m \log (D(G(c^i, z^i))) \\ \theta_g \leftarrow & \theta_g - \eta \nabla \tilde{V}(\theta_g)\end{aligned}$$

Discriminator

有两种常见架构。

下面的一个架构拆开两个evaluation可能是更合理的，因为给一个清晰的图片低分可能会让Network confused，可能就会觉得这个图片不够清晰。因为有两种错误，机器并不知道是哪种情况的错误，它需要自己分辨。

分开两个case，可以让机器有针对性的使某个case的值变化，当x清晰时，只改变match score即可。



Stack GAN

先产生比较小的图，再产生比较大的图

Stack GAN

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks", ICCV, 2017

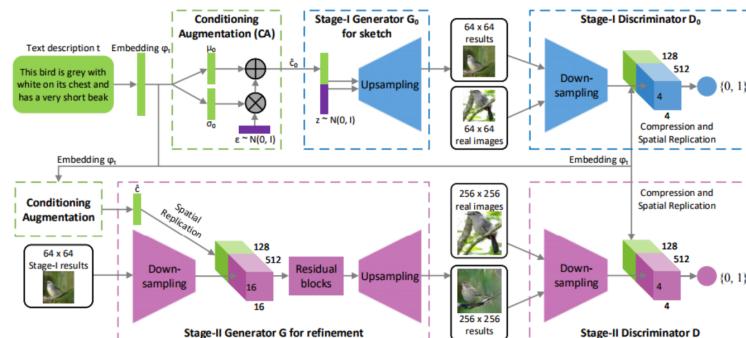
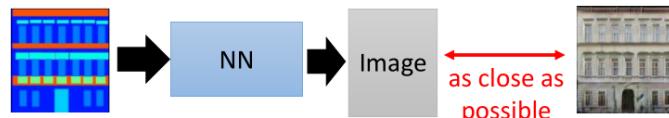


Image-to-image

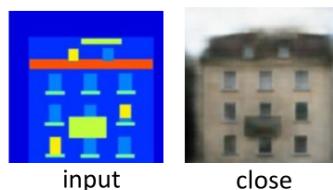
Traditional supervised approach

同一个image可以对应到不同的房子，因此会产生平均的结果，图片会是比较模糊的。

- Traditional supervised approach



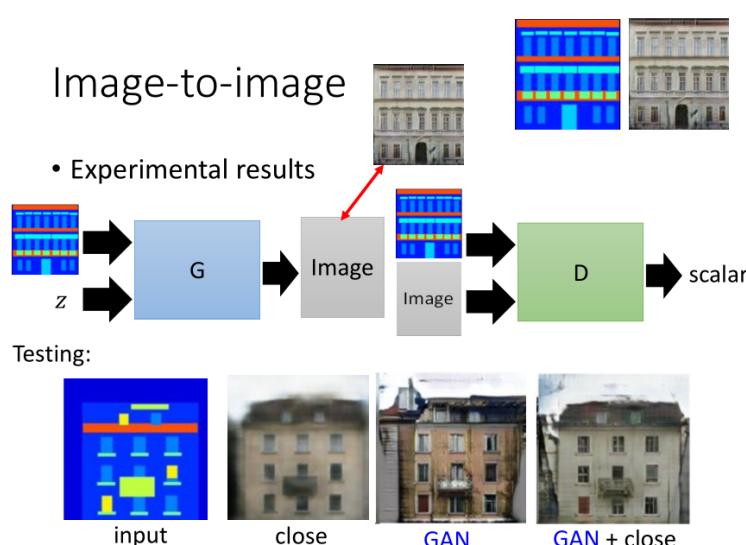
Testing:



It is blurry because it is the average of several images.

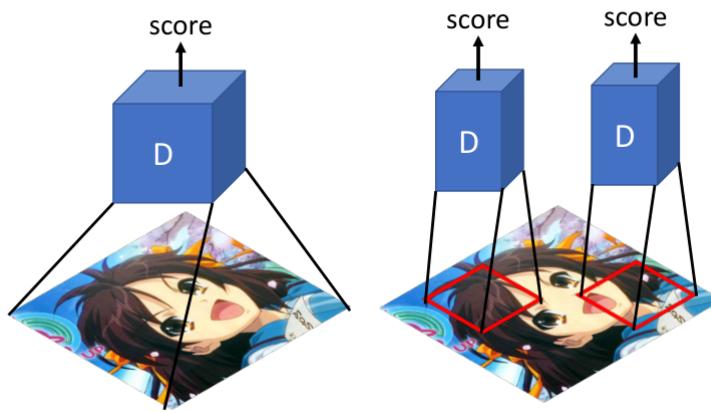
Conditional GAN

GAN有时也会生成奇怪的东西，如左上角。可以加一个constraint，希望Generator Output与原目标越接近越好，考虑这种情况下效果会更好。



Patch GAN

如果image很大，Discriminator参数量太多很容易overfitting或train的时间非常长，因此在image-to-image论文中，每次Discriminator不是检查一整张图片，而是每次检查一小块图片，这样叫Patch GAN。patch的size需要调整。



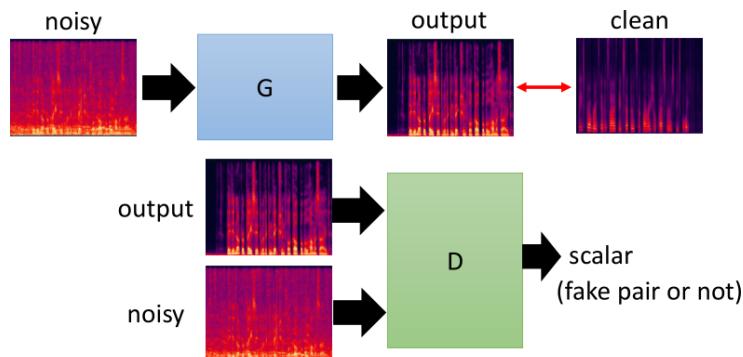
Speech Enhancement

Typical deep learning approach

找很多声音，然后把这些声音加上一些杂讯，接下来，你就 train 一个 generator，input 一段有杂讯的声音，希望 output 就是没有杂讯的声音。

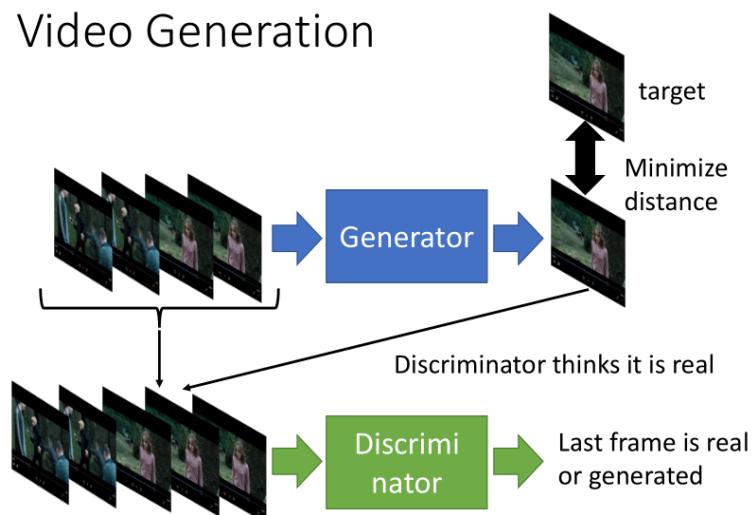
一段声音讯号可以用 spectrum 来表示，它看起来就像是一个 image 一样，所以这个 generator 常常也会直接套用那些 image 上常用的架构。

Conditional GAN



在 conditional GAN 里面，discriminator 要同时看generator 的 input 跟output，然后给它一个分数。这个分数决定现在 output 的这一段声音讯号是不是 clean 的，同时 output 跟input 是不是 match 的。

Video Generation



Unsupervised Conditional Generation by GAN

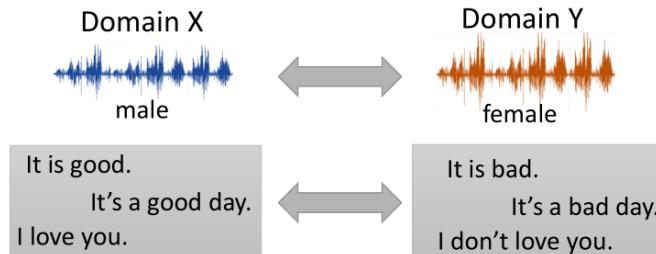
Unsupervised Conditional Generation

如果是 supervised conditional generation, 你需要 label 告诉机器什么样子的 input, 应该有什么样的 output.

所以今天我们要讨论的问题就是, 有没有办法做到 unsupervised conditional generation。只有两堆 data, machine 自己学到怎么从其中一堆转到另外一堆。这样的技术, 有很多的应用, 不是只能够用在影像上。



Transform an object from one domain to another
without paired data (e.g. style transfer)



我 surveyed 一下文献, 我认为大致上可分为两大类的作法

Approach 1: Direct Transformation

第一大类的做法是直接转。直接 learn 一个 generator, input x domain 的东西, 想办法转成 y domain 的东西。在经验上, 如果你今天要用这种 direct transformation 的方法, 你的 input output 没有办法真的差太多。如果是影像的话, 它通常能够改的是颜色、质地。所以如果是画风转换, 这个是比较有可能用第一个方法来实践。

那今天假设你要转的 input 跟 output 差距很大, 它们不是只有在颜色、纹理上面的转换的话, 那你就需要用到第二个方法。

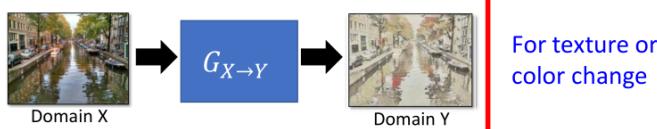
Approach 2: Projection to Common Space

第二个方法是如果今天你的 input 跟 output, 差距很大。比如说你要把真人转成动画人物。真人跟动画人物就是不像, 它不是你改改颜色, 或改改纹理就可以从真人转成动画人物的。

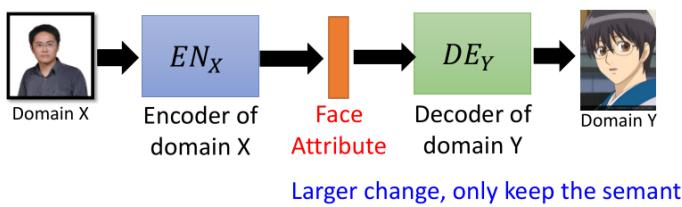
你先 learn 一个 encoder 比如说第一个 encoder 做的事情就是吃一张人脸的图, 然后它把人脸的特征抽出来, 比如说男, 戴眼镜, 接下来你输入到一个 decoder, 这个 decoder 它画出来的就是动画的人物, 它根据你 input 的人脸特征比如说是男的, 有戴眼镜的, 去产生一个对应的角色。如果你 input output 真的差很多的时候, 你就可以做这件事。

Unsupervised Conditional Generation

• Approach 1: Direct Transformation



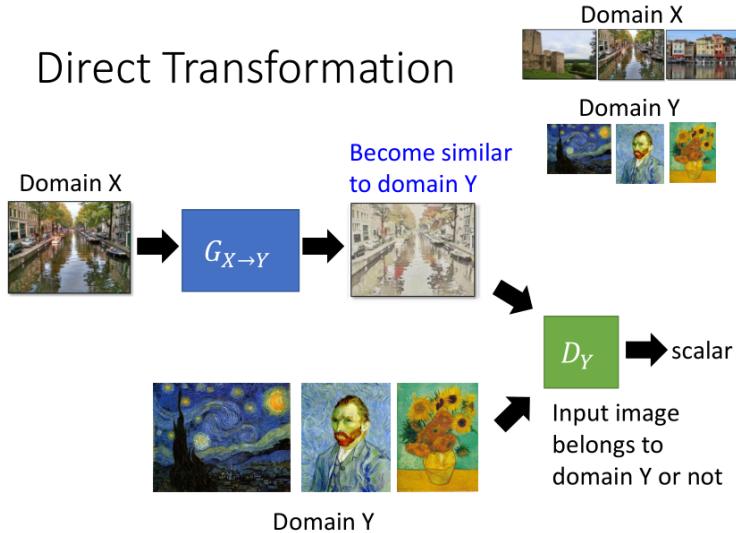
• Approach 2: Projection to Common Space



Direct Transformation

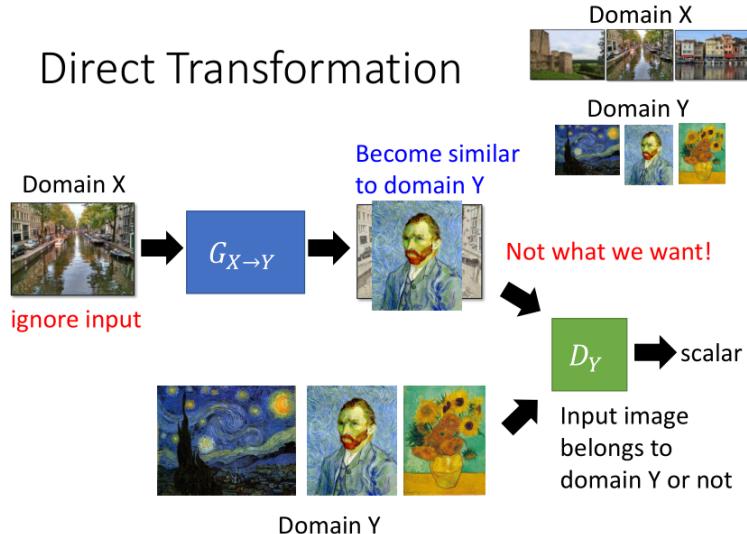
第一个做法是说，我们要 learn 一个 generator。这个 generator input x domain 的东西，要转成 y domain 的东西。那我们现在 x domain 的东西有一堆，y domain 的东西有一堆，但是合起来的 pair 没有，我们没有它们中间的 link。那 generator 怎么知道给一个 x domain 的东西，要 output 什么样 y domain 的东西呢？用 supervised learning 当然没有问题，但现在是 unsupervised generator 怎么知道如何产生 y domain 的东西呢？

这个时候你就需要一个 y domain 的 discriminator。这个 discriminator 做的事情是，它可以鉴别说这张 image 是 x domain 的 image，还是 y domain 的 image。



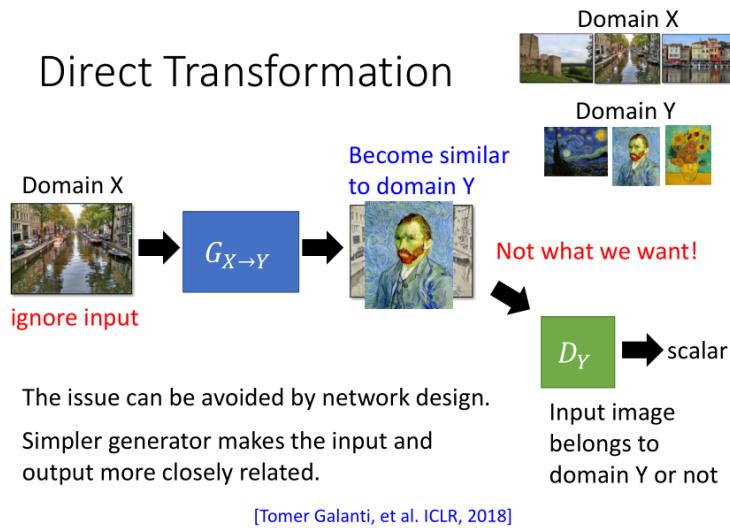
接下来 generator 要做的事情就是想办法去骗过 discriminator。如果 generator 可以产生一张 image 去骗过 discriminator，那 generator 产生的 image，就会像是一张 y domain 的 image。如果 y domain 现在是梵谷的画作，generator 产生的 output 就会像是梵谷的画作，因为 discriminator 知道梵谷的画作，长得是什么样子。

但是现在的问题是，generator 可以产生像是梵谷画作的东西，但完全可以产生一个跟 input 无关的东西。举例来说，它可能就学到画这张自画像就可以骗过 discriminator，因为这张自画像，确实很像是梵谷画的。但是这张自画像跟输入的图片完全没有任何半毛钱的关系，这个就不是我们要的。



所以我们希望 generator 不只要骗过 discriminator，generator 的输入和输出是必须有一定程度的关系的。这件事在文献上有不同的做法，我们等一下会讲 Cycle GAN，这是最知名的方法。

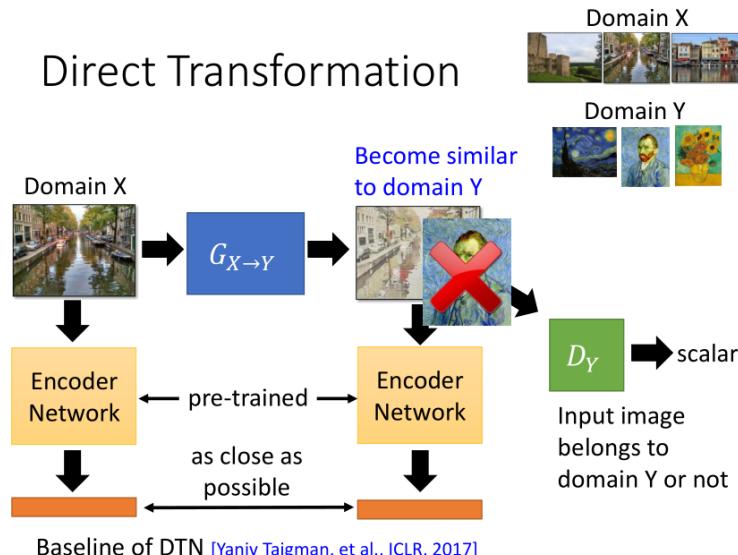
Direct Transformation



第一个方法就是不要管它，这是最简单的做法，无视这个问题直接做下去，事实上有时也做得起来。无视这个问题，直接 learn 一个 generator，一个 discriminator。为什么这样子有机会可以 work 呢？因为 generator 的 input 跟 output 其实不会差太多。假设你的 generator 没有很深，那总不会 input 一张图片，然后 output 一个梵谷的自画像，这未免差太多了。

所以今天其实 generator 如果你没有特别要求它的话，它喜欢 input 就跟 output 差不多。它不太想要改太多，它希望改一点点就骗过 discriminator 就好。所以今天你直接 learn 一个这样的 generator，这样的 discriminator，不加额外的 constrain，其实也是会 work 的，你可以自己试试看。

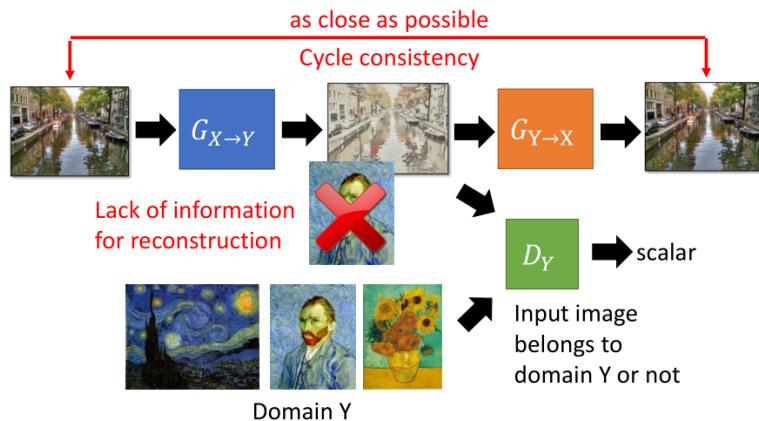
在这个文献里面说如果今天 generator 比较 shallow，所以它 input 跟 output 会特别像，那这个时候，你就不需要做额外的 constrain，就可以把这个 generator learn 起来。那如果你 generator 很 deep，有很多层，那它就真的可以让 input output 非常不一样，这个时候，你就需要做一些额外的处理，免得让 input 跟 output 变成完全不一样的 image。



第二个方法是拿一个 pre-trained 好的 network，比如说 VGG 之类的。把这个 generator 的 input 跟 output 通通都丢给这个 pre trained 好的 network，然后 output 一个 embedding。接下来你在 train 的时候，generator 一方面会想要骗过 discriminator，让它 output 的 image 看起来像是梵谷的画作

但是同时，generator 会希望这个 pre-trained 的 model，它们 embedding 的 output 不要差太多。那这样的好处就是，因为这两个 vector 没有差太多代表说 generator 的 input 跟 output 就不会差太多。

Direct Transformation



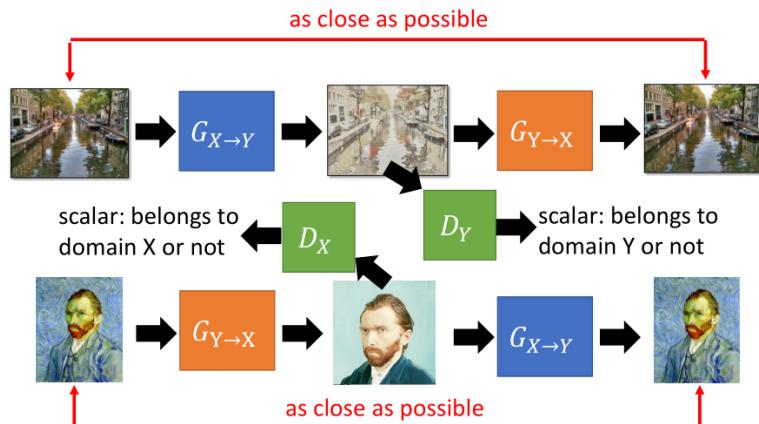
Cycle GAN

第三个做法就是大家所熟知的 Cycle GAN。在 Cycle GAN 里面，你要 train 一个 x domain 和 y domain 的 generator。它的目的是，给它一张 y domain 的图，input 一张风景画，第一个 generator 把它转成 y domain 的图，第二个 generator 要把 y domain 的图，还原回来一模一样的图。

因为现在除了要骗过 discriminator 以外，generator 要让 input 跟 output 越像越好，为了要让 input 跟 output 越像越好，你就不可以在中间产生一个完全无关的图。如果你在这边产生一个梵谷的自画像，第二个 generator 就无法从梵谷的自画像还原成原来的风景画，因为它已经完全不知道原来的输入是什么了。所以这张图片，必须要保留有原来输入的资讯，那这样第二个 generator 才可以根据这张图片转回原来的 image。

这个就是 Cycle GAN，那这样 input 跟 output 越接近越好，input 一张 image 转换以后要能够转得回来，两次转换要转得回来这件事情就叫做 Cycle consistency。

Direct Transformation



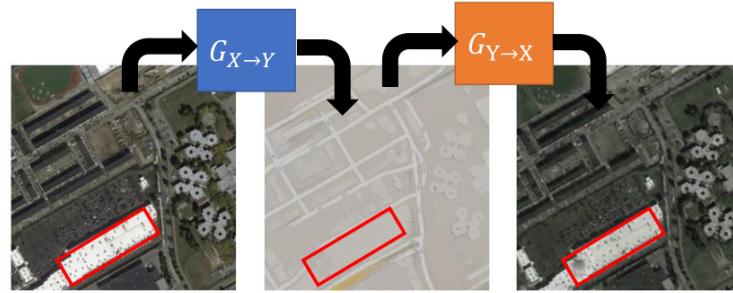
Cycle GAN 可以是双向的，本来有 x domain 转 y domain，y domain 转 x domain，再 train 另外一个 task 把 y domain 的图丢进来，然后把它转成 x domain 的图，同时你要有一个 discriminator 确保这个 generator 它 output 的图像是 x domain 的。接下来再把 x domain 的图转回原来 y domain 的图，一样希望 input 跟 output 越接近越好。这样就可以同时去 train，两个 generator 和两个 discriminator。

Issue of Cycle Consistency

其实 Cycle GAN，现在还是有一些问题是没有解决的。一个问题就是，NIPS 有一篇 paper 叫做 Cycle GAN: a master of stenography。stenography 是隐写术，就是说 Cycle GAN 会把 input 的东西藏起来，然后在 output 的时候，再呈现出来。

• CycleGAN: a Master of Steganography (隱寫術)

[Casey Chu, et al., NIPS workshop, 2017]

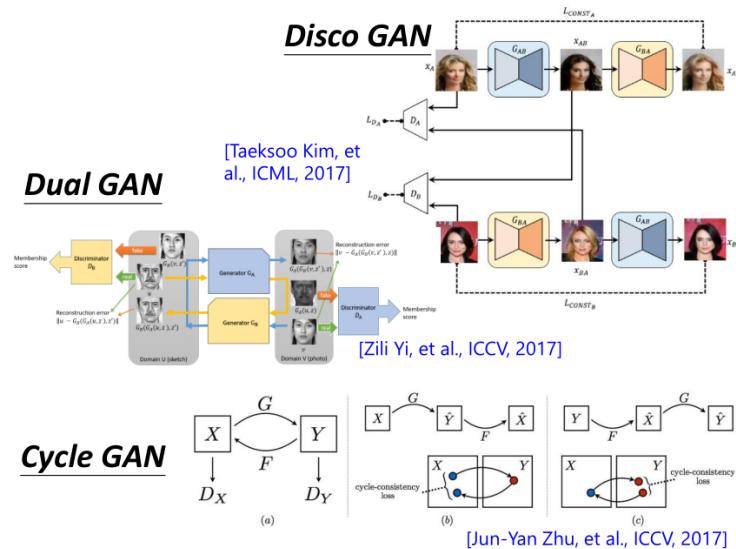


The information is hidden.

对第二个 generator 来说，如果你只看到一张图，屋顶上是没有黑点的，你是怎么知道上面应该要产生黑点的。

所以有一个可能是，Cycle GAN 虽然有 Cycle consistency 的 loss 强迫你 input 跟 output 要越像越好，但是 generator 有很强的能力把资讯藏在人看不出来的地方，也就是说要如何 reconstruct 这张 image 的资讯可能是藏在这张 image 里面，让你看不出来这样，也许这个屋顶上仍然是有黑点的，只是你看不出来而已。那如果是这样子的情况，那就失去 Cycle consistency 的意义了。因为 Cycle consistency 的意义就是，第一个 generator output 跟 input 不要差太多，但如果今天 generator 很擅长藏资讯，然后再自己解回来那这个 output 就有可能跟这个 input 差距很大了。

那这个就是一个尚待研究的问题，也就是 Cycle consistency 不一定有用，machine 可能会自己学到一些方法去避开 Cycle consistency 带给你的 constrain。



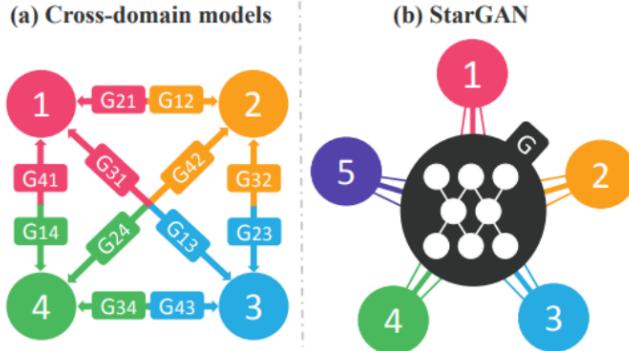
在文献上除了 Cycle GAN 以外，可能还看到其它的 GAN 比如说 Dual GAN、Disco GAN。这 3 个东西没有什么不同，这些方法其实就跟 Cycle GAN 是一样的。

StarGAN

StarGAN

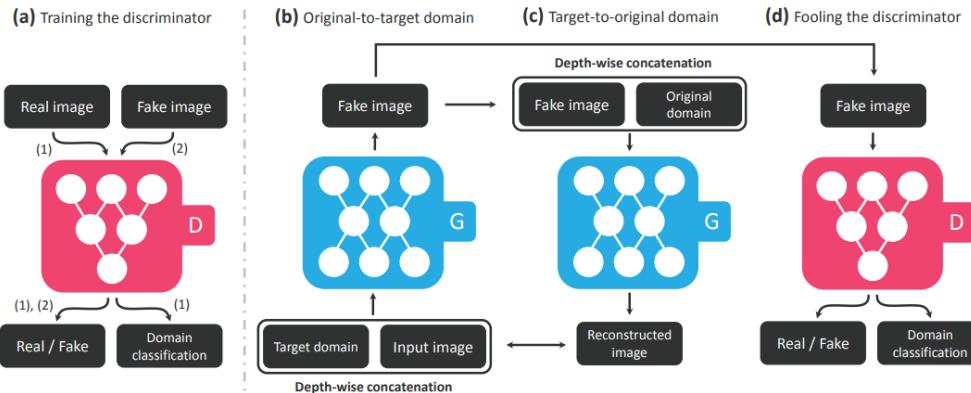
For multiple domains,
considering starGAN

[Yunjey Choi, arXiv, 2017]



有时候你会有一个需求是你要用多个 domain 互转。star GAN 它做的事情是，它只 learn 了一个 generator，但就可以在多个 domain 间互转。

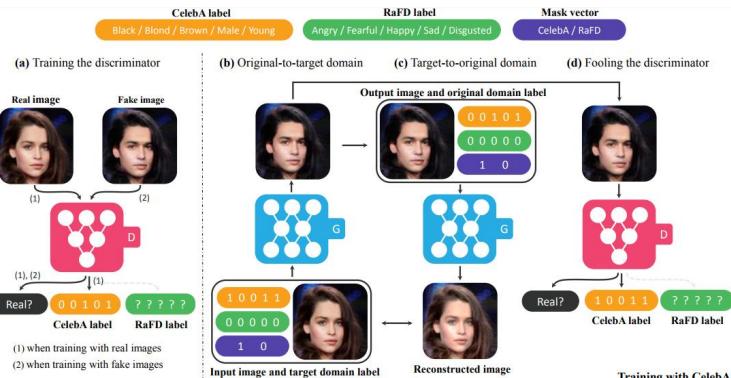
那在 star GAN 里面是怎么样呢？在 star GAN 里面你要 learn 一个 discriminator，这个 discriminator 它会做两件事：首先给它一张 image，它要鉴别说这张 image 是 real 还是 fake 的；再来它要去鉴别这一张 image 来自于哪一个 domain。



在 star GAN 里面，只需要 learn 一个 generator 就好，这个 generator 它的 input 是一张图片跟你目标的 domain。然后它根据这个 image 和目标的 domain，就把新的 image 生成出来。

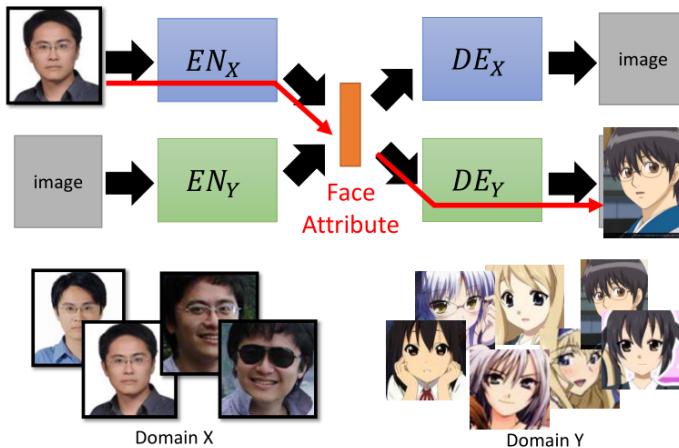
接下来再把这个同样的 image，丢给同一个 generator，把这一张被 generated 出来的 image，丢给同一个 generator，然后再告诉它原来 input 的 image 是哪一个 domain，然后再用这个 generator 合回另外一张图片，希望 input 跟 output 越接近越好。那这个东西就是 Cycle consistency 的 loss。

那这个 discriminator 做的事情就是要确认说这张转出来的 image 到底对不对。那要确认两件事，第一件事是，这张转出来的 image 看起来有没有真实；再来就是，这张转出来的 image 是不是我们要的 target domain。然后 generator 就要去想办法骗过 discriminator。



Projection to Common Space

Target



第二个做法是要把 input 的 object 用 encoder project 到一个 latent 的 space, 再用 decoder 把它转换回来。

那假设有两个 domain 一个是人的 domain, 一个是动画人物的 domain。想要在这两个 domain 间做互相转换的话, 就用 x domain 的 encoder 抽取人的特征, 用 y domain 的 encoder 抽取动画人物的特征, 它们参数可能是不一样的, 因为毕竟人脸和动画人物的脸还是有一些差别, 所以这两个 network 不见得是一样的 network。

x domain 的 encoder 一张image, 它会抽出它的 attribute。

所谓的 attribute 就是一个 latent 的 vector, input 一张 image, encoder 的 output 就是一个 latent 的 vector。

接下来把这个 latent 的 vector, 丢到 decoder 里面, 如果丢到 x domain 的 decoder, 它产生出来的是真实人物的脸; 如果是丢到 y domain 的 decoder, 它产生出来就是二次元人物的脸。

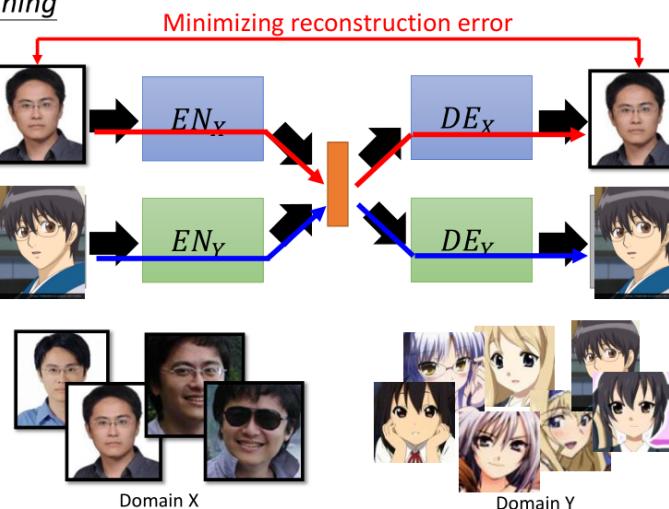
我们希望最后可以达到的结果是你给它一张真人的人脸, 通过 x domain 的 encoder 抽出 latent 的 representation。这个 latent 的 representation 是一个 vector, 我们期待说这个 vector 的每一个 dimension 就代表了 input 的这张图片的某种特征, 有没有戴眼镜, 是什么性别, 等等。

那接下来你用 y domain 的 decoder, 吃这个 vector, 根据这个 vector 里面所表示的人脸的特征合出一张 y domain 的图, 我们希望做到这一件事。

但是实际上如果我们今天有 x domain 跟 y domain 之间的对应关系, 要做到这件事非常容易, 因为就是一个 supervised learning 的问题。

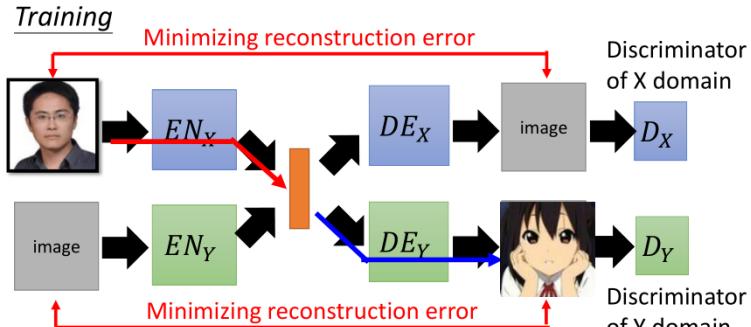
但是现在我们是一个 unsupervised learning 的问题, 只有 x domain 的 image, 跟 y domain 的 image, 它们是分开的, 那怎么 train 这些 encoder 跟这些 decoder 呢?

Training



可以组成两个 auto encoder, x domain 的 encoder 跟 x domain 的 decoder 组成一个 auto encoder, input 一张 x domain 的图让它 reconstruct 回原来 x domain 的图。y domain 的 encoder 跟 y domain 的 decoder 组成一个 auto encoder, input 一个 y domain 的图, reconstruct 回原来 y domain 的图。我们知道这两个 auto encoder 在 train 的时候它们都是要 minimize reconstruction error。

用这样的方法，你确实可以得到2个 encoder, 2个 decoder, 但是这样会造成的问题是这两个 encoder 之间是没有任何关联的。



Because we train two auto-encoders separately ...

The images with the same attribute may not project to the same position in the latent space.

还可以多做一件事情是，可以把 discriminator 加进来。你可以 train 一个 x domain 的 discriminator，强迫 decoder 的 output 看起来像是 x domain 的图。因为我们知道，假设如果只 learn auto encoder，只去 minimize reconstruction error，decoder output 的 image 会很模糊。有一个 x domain 的 discriminator 吃这一张 image，然后鉴别它是不是 x domain 的图，有一个 y domain 的 discriminator，它吃一张 image 鉴别它是不是 y domain 的图。这样你会强迫你的 x domain 的 decoder 跟 y domain 的 decoder 它们 output 的 image 都比较 realistic。

encoder decoder discriminator，它们 3 个合起来其实就是一个 VAE GAN，可以看做是用 GAN 强化 VAE，也可以看做 VAE 来强化 GAN。另外 3 个合起来其实就是另外一个 VAE GAN。

但是因为这两个 VAE GAN 它们的 training 是完全分开，各自独立的。所以实际上 train 完以后，会发现它们的 latent space 可能意思是不一样的。

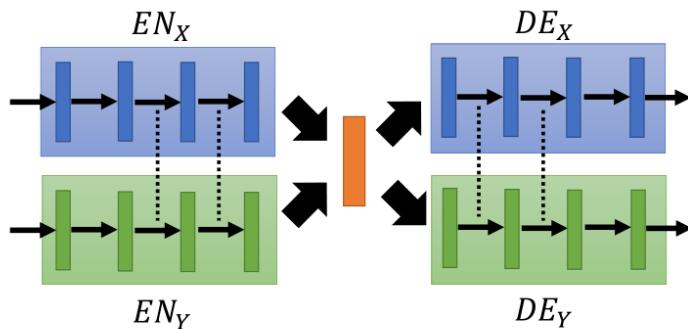
也就是说你今天丢这张人脸进去变成一个 vector，把这个 vector 丢到这张图片里面，搞不好它产生的就是一个截然不同的图片。

因为今天这两组 auto encoder 是分开 train 的，也许上面这组 auto encoder 是用这个 latent vector 的第一维代表性别，第二维代表有没有戴眼镜；下面这个是用第三维代表性别，第四维有没有戴眼镜。

如果是这样子的话，就做不起来，也就是说今天 x 这一组 encoder 跟 decoder，还有 y 这一组 encoder 跟 decoder，它们用的 language 是不一样的，它们说的语言是不一样的。

所以 x domain 的 encoder 吐出一个东西，要叫 y domain 的 decoder 吃下去，它 output 并不会跟 x domain encoder 的 input 有任何的关联性。

怎么解决这个问题？在文献上，有各式各样的解法。



Sharing the parameters of encoders and decoders

Couple GAN [Ming-Yu Liu, et al., NIPS, 2016]

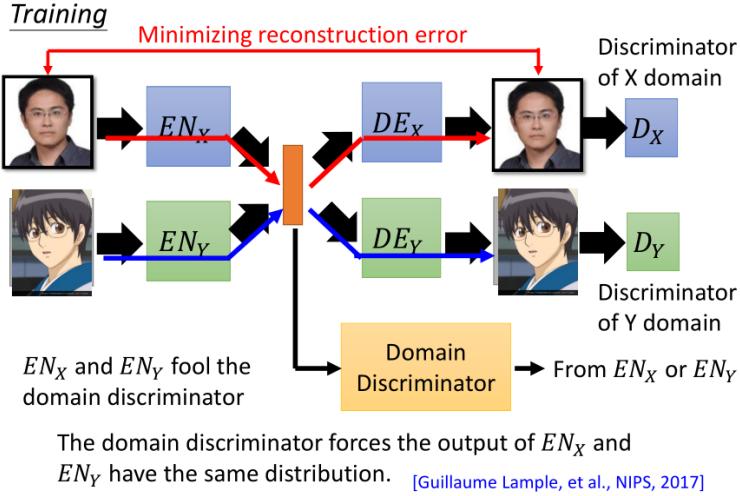
UNIT [Ming-Yu Liu, et al., NIPS, 2017]

一个常见的解法是让不同 domain 的 encoder 跟 decoder，它们的参数被 tie 在一起。

我们知道 encoder 有好几个 hidden layer，x domain encoder 有好几个 hidden layer，y domain encoder 也有好几个 hidden layer。你希望它们最后的几个 hidden layer 参数是共用的，它们共用同一组参数。可能前面几个 layer 是不一样的，但最后的几个 layer，必须是共用的，或者两个不同 domain 的 decoder，它们前面几个 layer 是共用的，后面几个 layer 是不一样。

那这样的好处是什么？因为它们最后几个 hidden layer 是共用的，也许因为透过最后几个 hidden layer 是共用这件事会让这两个 encoder 把 image 压到同样的 latent space 的时候，它们的 latent space 是同一个 latent space，它们的 latent space 会用同样的 dimension 来表示同样的人脸特征。这样的技术，被用在 couple GAN 跟 UNIT 里面。

像这种 share 参数的 task，它最极端的状况就是，这两个 encoder 共用同一组参数，就是同一个 encoder。只是在使用的时候吃一个 flag，代表现在要 encoder 的 image 是来自于 x domain 还是来自于 y domain。

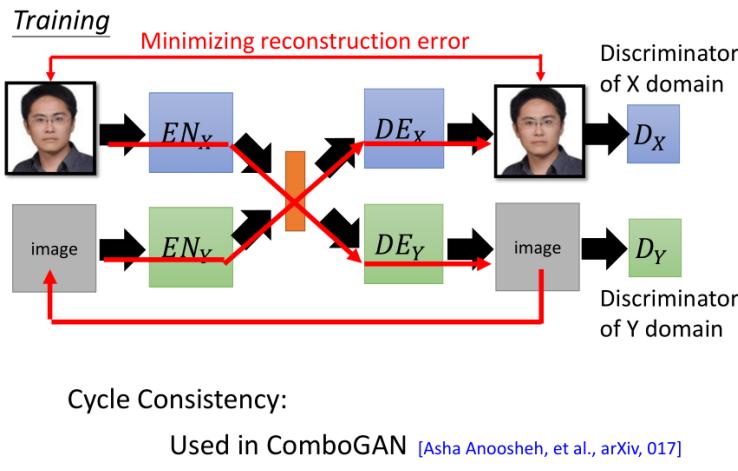


还有满坑满谷的招式。比如说加一个 **domain 的 discriminator**。这个概念跟 domain adversarial training 是一样的，其实是一样的。

它的概念是：原来 x domain 跟 y domain 都是自己搞自己的东西，但我们现在再加一个 domain discriminator，这个 domain discriminator 要做的事情是给它这个 latent 的 vector，它去判断说这个 vector 是来自于 x domain 的 image，还是来自于 y domain 的 image。然后 x domain encoder 跟 y domain encoder，它们的工作就是想要去骗过这个 domain 的 discriminator，让 domain discriminator 没办法凭借这个 vector 就判断它是来自于 x domain 还是来自于 y domain。如果今天 domain 的 discriminator 无法判断这个 vector 是来自于 x domain 和 y domain，意味着，两个 domain 的 image 变成 code 的时候，它们的 distribution 都是一样的。因为它们的 distribution 是一样的，也许我们就可以期待同样的维度就代表了同样的意思。举例来说假设真的照片男女比例是 1:1，动画人物的照片，男女比例也是 1:1。因为男女的比例都是 1:1，最后如果你要让两个 domain 的 feature，它的 distribution 一样，那你就需要用同一个维度来存这个男女比例是 1:1 的 feature，如果是性别都用第一维来存，这样它们的 distribution 才会变得一样。

所以假设你今天的这两个 domain 它们 attribute 的 distribution 是一样的

比如说，男女的比例是一样的，有戴眼镜跟没戴眼镜的比例是一样的，长发短发比例是一样的。那你也许期待说，通过 domain discriminator 强迫这两个 domain 的 embedding latent feature 是一样的时候，那它们就会用同样的 dimension 来表示同样的事情，来表示同样的 characteristic。



还有其它的招数。举例来说，你也可以用 **Cycle consistency**

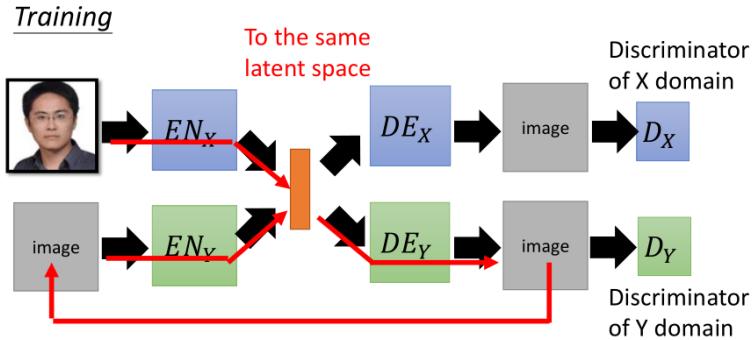
那如果把这个技术来跟 Cycle GAN 来做比较的话，Cycle GAN 就是有两个 transformation 的 network，这个跟 Cycle GAN 的 training 其实就是一模一样的。

x domain 的 encoder 加 y domain 的 decoder，它们合起来就是从 x domain 转到 y domain，然后有一个 discriminator 确定这个 image 看起来像不像 y domain 的 image。

接下来，再把这张 image，通过 y domain 的 encoder 跟 x domain 的 decoder，转回原来的 image，希望 input 的 image 跟 output 的 image越接近越好。

只是原来在 Cycle GAN 里面，我们说从 x domain 到 y domain generator 就是一个 network，我们没有把它特别切成 encoder 跟 decoder，在这边，我们会把它切成把 x domain 到 y domain 的 network 切成一个 x domain 的 encoder 和一个 y domain 的 decoder。从 y domain 到 x domain 的 network，我们说它有一个 y domain 的 encoder，一个 x domain 的 decoder。network 的架构不太一样，然后中间的那个 latent space 是 shared。但是实际上它们是 training 的 criteria，其实是一样的。

这样的技术，就用在 Combo GAN 里面。



Semantic Consistency:

Used in DTN [Yaniv Taigman, et al., ICLR, 2017] and XGAN [Amélie Royer, et al., arXiv, 2017]

还有一个叫做 **semantic consistency**

你把一张图片丢进来，然后把它变成 code，然后接下来，把这个 code

用 y domain 的 decoder 把它转回来。再把 y domain 的 image 丢到 y domain 的 encoder，希望通过 x domain encoder 的 encode 跟 y domain encoder 的 encode，它们的 code 要越接近越好。

那这样的好处是说，我们本来在做 Cycle consistency 的时候，你算的是两个 image 之间的 similarity。那如果是 image 和 image 之间的 similarity，通常算的是 pixel wise 的 similarity，不会考虑 semantic，而是看它们表象上像不像。如果是在这个 latent 的 space 上面考虑的话，你就是在算它们的 semantic 像不像，算它们的 latent 的 code 像不像。

这个技术被用在 XGAN 里面。

Voice Conversion

也可以做 voice conversion，就是把 A 的声音，转成 B 的声音。

过去的 voice conversion，就是要收集两个人的声音，假如你要把 A 的声音，转成 B 的声音，你就要把 A 找来念 50 句话，B 找来也念 50 句话，让它们念一样的句子，接下来 learn 一个 model，比如说 sequence to sequence model 或是其它，吃一个 A 的声音，然后转成 B 的声音就结束了，这就是一个 supervised learning problem。

若用 GAN 的技术，就我们用今天学到的那些技术在两堆声音间互转。只需要收集 speaker A 的声音，再收集 speaker B 的声音，它们两个甚至可以说的就是不同的语言，用我们刚才讲的 Cycle consistency，把 A 的声音转成 B 的声音。

为什么不做 TTS(Text To Speech)呢？voice conversion 可以保留原来说话人的情绪语调。

Theory behind GAN

我们已经讲了 GAN 的直观的想法，今天要来讲 GAN 背后的理论。

要讲的是 2014 年 Ian Goodfellow 在 propose GAN 的时候它的讲法，等一下可以看看跟我们讲的 GAN 的直观的想法里面有没有矛盾的地方。其实是有一些地方还颇矛盾的，至今仍然没有好的 solution、好的手法可以解决。

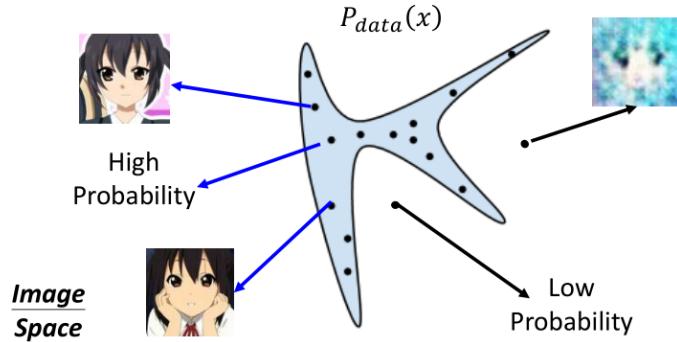
GAN 要做的就是根据很多 example 自己去进行生成。所谓的生成到底是什么样的问题？假设要生成的东西是 image，用 x 来代表一张 image，每一个 image 都是 high dimensional 高维空间中的一个点。假设产生 64×64 的 image，它是 64×64 维空间中的一个点。

这边为了画图方便假设每一个 x 就是二维空间中的一个点，虽然实际上它是高维空间中的一个点。

Generation

x : an image (a high-dimensional vector)

- We want to find data distribution $P_{data}(x)$



现在要产生的东西比如说要产生 image，它其实有一个固定的 distribution 写成 $P_{data}(x)$ 。在这整个 image 的 space 里面，在这整个 image 所构成的高维空间中只有非常少的部分、一小部分 sample 出来的 image 看起来像是人脸，在多数的空间中 sample 出来的 image 都不像人脸。

假设生成的 x 是人脸的话，它有一个固定的 distribution，这个 distribution 在蓝色的这个区域，它的机率是高的，在蓝色的区域以外，它的机率是低的。

我们要机器找出这个 distribution，而这个 distribution 到底长什么样子，实际上是不知道的。可以搜集很多的 data 知道 x 可能在某些地方分布比较高，但是要我们把这个式子找出来我们是不知道要怎么做的。

现在 GAN 做的是一个 generative model 做的事情，就是要找出这个 distribution。

Maximum Likelihood Estimation

在有 GAN 之前怎么做 generative？我们是用 Maximum Likelihood Estimation。

现在有一个 data 的 distribution 它是 $P_{data}(x)$ ，这个 distribution 它的 formulation 长什么样子我们是不知道的。我们可以从这个 distribution sample 它，假设做二次元人物的生成，就是从 database 里面 sample 出 image。

接下来我们要自己去找一个 distribution，这个 distribution 写成 $P_G(x; \theta)$ 。这个 distribution 是由一组参数 θ 所操控的。由 θ 所操控的意思是这个 distribution 假设它是一个 Gaussian Mixture Model，这个 θ 指的就是 Gaussian 的 mean 跟 variance。我们要去调整 Gaussian 的 mean 跟 variance，使得我们得到的这个 distribution $P_G(x; \theta)$ 跟真实的 distribution $P_{data}(x)$ 越接近越好。

虽然我们不知道 $P_{data}(x)$ 长什么样子，我们只能从 $P_{data}(x)$ 里面去 sample，但是我们希望 $P_G(x; \theta)$ 可以找一个 θ 让 $P_G(x; \theta)$ 跟 $P_{data}(x)$ 越接近越好。

怎么做？

首先可以从 $P_{data}(x)$ sample 一些东西出来

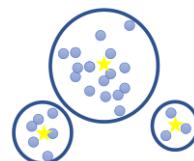
- Given a data distribution $P_{data}(x)$ (We can sample from it.)
- We have a distribution $P_G(x; \theta)$ parameterized by θ
 - We want to find θ such that $P_G(x; \theta)$ close to $P_{data}(x)$
 - E.g. $P_G(x; \theta)$ is a Gaussian Mixture Model, θ are means and variances of the Gaussians

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 θ^* maximizing the likelihood

接下来对每一个 sample 出来的 x^i ，我们都可以计算它的 likelihood

假设给定一组参数 θ , 我们就知道 P_G 这个 probability 的 distribution 长什么样子, 我们就可以计算从这个 distribution 里面sample 出某一个 x^i 的机率, 可以计算这个 likelihood。

接下来要做的就是, 我们要找出一个 θ , 使得 P_G 跟 $P_{data}(x)$ 越接近越好。

我们希望这些从 $P_{data}(x)$ 里面 sample 出来的 example, 如果是用 P_G 这个 distribution 来产生的话, 它的 likelihood 越大越好。把所有的机率乘起来就得到 total 的 likelihood, 我们希望 total likelihood 越大越好。

就是要去找一个 θ^* , 找一组最佳的参数, 它可以去 maximize L 这个值。

Minimize KL Divergence

$$\begin{aligned}
 \theta^* &= \arg \max_{\theta} \prod_{i=1}^m P_G(x^i; \theta) = \arg \max_{\theta} \log \prod_{i=1}^m P_G(x^i; \theta) \\
 &= \arg \max_{\theta} \sum_{i=1}^m \log P_G(x^i; \theta) \quad \{x^1, x^2, \dots, x^m\} \text{ from } P_{data}(x) \\
 &\approx \arg \max_{\theta} E_{x \sim P_{data}} [\log P_G(x; \theta)] \\
 &= \arg \max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) dx - \int_x P_{data}(x) \log P_{data}(x) dx \\
 &= \arg \min_{\theta} KL(P_{data} || P_G) \quad \text{How to define a general } P_G?
 \end{aligned}$$

Maximum Likelihood 的另外一个解释是: Maximum Likelihood Estimation = Minimize KL Divergence。

这个式子可以稍微改变一下, 取一个 log, 把 log 放进去, 变成 summation over 第一笔 example 到第 m 答 example。

Suppose we sample N of these $x \sim P_{data}$. Then, the Law of Large Number $\bar{X}_n = \frac{1}{n}(X_1 + \dots + X_n)$ says that as N goes to infinity:

$$\frac{1}{N} \sum_i^N \log P_G(x^i | \theta) = \mathbb{E}_{x \sim P_{data}} [\log P(x | \theta)]$$

这件事情其实就是在 approximate 从 $P_{data}(x)$ 这个 distribution 里面 sample x 出来, maximize $\log P_G(x)$ 的 expected value, expectation distribution 是你要 sample 的 data

接下来可以把 expectation 这一项展开, 就是一个积分

加一项看起来没有什么用的东西, 在后面加这么一项, 里面只有 $P_{data}(x)$, 跟 P_G 是完全没有任何关系的。所以加这一项根本不会影响你找出来的最大的 x。

那为什么要加这一项? 目的是为了告诉你 Maximum Likelihood 它就是 KL Divergence。把式子做一下整理, 这个式子它就是 $P_{data}(x)$ 跟 P_G 的 KL Divergence。

所以找一个 θ 去 maximize likelihood, 等同于找一个 θ 去 minimize $P_{data}(x)$ 跟 P_G 的 KL Divergence。

在机器学习里面讲的所谓 Maximum Likelihood, 我们要找一个 Generative Model 去 Maximum Likelihood, Maximum Likelihood 这件事情就等同于 minimize 你的 Generative Model 所定义的 distribution P_G 跟现在的 data $P_{data}(x)$ 之间的 KL Divergence。

Generative Adversarial Network

接下来会遇到的问题是: 假设我们的 P_G 只是一个 Gaussian Mixture Model 显然有非常多的限制, 我们希望 P_G 是一个 general 的 distribution。但假设把 P_G 换成比 Gaussian 更复杂的东西, 会遇到的问题就是算不出你的 likelihood, 算不出 $P_G(x; \theta)$ 这一项。

它可能是一个 Neural Network, 你就没有办法计算它的 likelihood, 所以就有了一些新的想法。

让 machine 自动的生成东西比如说做 image generation 从来都不是新的题目, 你可能看最近用 GAN 做了很多 image generation 的 task, 好像 image generation 是这几年才有的东西。其实不是, image generation 在八零年代就有人做过了。

那个时候的作法是用 Gaussian Mixture Model, 搜集很多很多的 image, 每一个 image 就是高维中中间一个 data point, 就可以 learn 一个 Gaussian Mixture Model 去 maximize 产生那些 image likelihood。

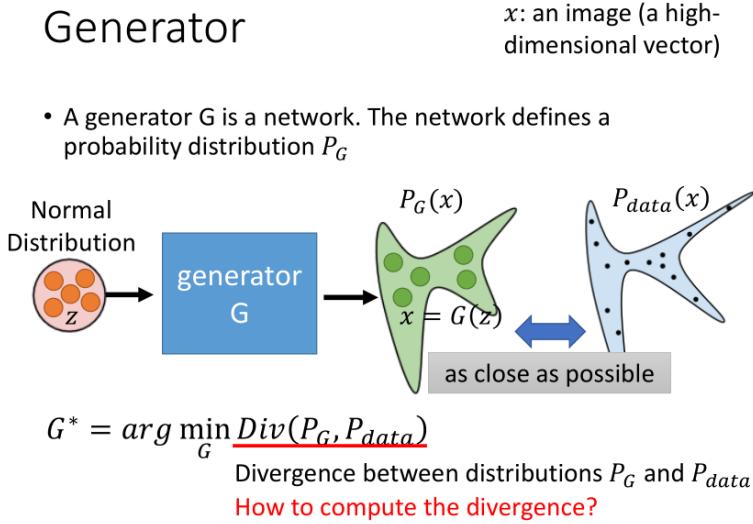
但如果你看古圣先贤留下来的文献的话, 就会发现如果用 Gaussian Mixture Model 产生的 image, 非常非常的糊。

这个可能原因是因为 image 它是高维空间中一个 manifold。image 其实是高维空间中一个低维的 manifold。

所以如果用 Gaussian Mixture Model，它其实就不是一个 manifold。用 Gaussian Mixture Model 不管怎么调 mean 跟 variance，它就不像是你的 target distribution，所以怎么做都是做不好。

所以需要用更 generalize 的方式来 learn generation 这件事情。

Generator



在 Generative Adversarial Network 里面，generator 就是一个 network。

我们都知道 network 就是一个东西然后 output 一个东西。举例来说，input 从某一个 distribution sample 出来的 noise z ，input 一个随机的 vector z ，然后它就会 output 一个 x 。

如果 generator G 看作是一个 function 的话，这个 x 就是 $G(z)$ 。如果是做 image generation 的话，那你的 x 就是一个 image。

我们说这个 z 是从某一个 prior distribution，比如说是从一个 normal distribution sample 出来的，sample 出来的 z 通通通过 G 得到另外一大堆 sample，把这些 sample 通通集合起来得到的就是另外一个 distribution。

虽然 input 是一个 normal distribution 是一个单纯的 Gaussian Distribution，但是通过 generator 以后，因为这个 generator 是一个 network，它可以把这个 z 通过一个非常复杂的转换把它变成 x ，所以把通过 generator 产生出来的 x 通通集合起来，它可以是一个非常复杂的 distribution。而这个 distribution 就是我们所谓的 P_G 。

有人可能会问这个 Prior Distribution 应该要设成什么样子。文献上有人会用 Normal Distribution，有人会用 Uniform Distribution。我觉得这边其实 Prior Distribution 用哪种 distribution 也许影响并没有那么大。

因为 generator 它是一个 network。一个 hidden layer 的 network 它就可以 approximate 任何 function，更何况是有多个 hidden layer 的 network，它可以 approximate 非常复杂的 function。

所以就算是 input distribution 是一个非常简单的 distribution，通过了这个 network 以后，它也可以把这个简单的 distribution 凹成各式各样的形状，所以不用担心这个 input 是一个 normal distribution 对 output 来说有很大的限制。

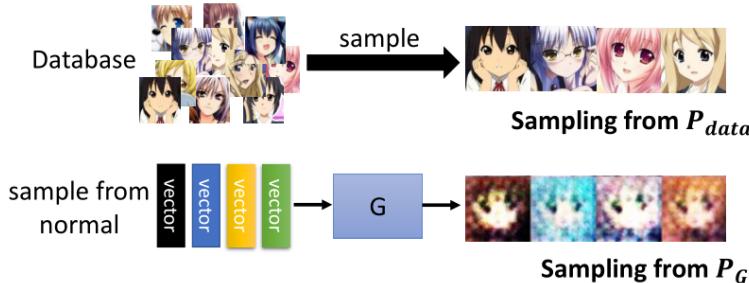
接下来目标是希望根据这个 generator 所定义出来的 distribution P_G 跟我们的 data 的 distribution $P_{data}(x)$ 越接近越好。

Discriminator

Discriminator

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

Although we do not know the distributions of P_G and P_{data} , we can sample from them.



如果要写一个 Optimization Formulation 的话，这个 formulation 看起来是这个样子。

我们要找一个 generator G，这个 generator 可以让它所定义出来的 distribution P_G 跟我们的 data $P_{\text{data}}(x)$ 之间的某种 divergence 越小越好。

举例来说如果是 Maximum Likelihood 的话它就是要 minimize KL Divergence。在 GAN 里面 minimize 的不是 KL Divergence 而是其它的 Divergence。这边写一个 Div 就代表反正它是某一种 Divergence。

假设能够计算这个 Divergence，要找一个 G 去 minimize 这个 Divergence，那就用 Gradient Descent 就可以做了。但问题是该怎么计算出这个 Divergence？

$P_{\text{data}}(x)$ 的 formulation 我们是不知道的。它并不是什么 Gaussian Distribution。 P_G 的 formulation 我们也是不知道的。

假设 P_G 跟 $P_{\text{data}}(x)$ 它的 formulation 我们是知道的，我们代进 Divergence 的 formulation 里面就可以算出它的 Divergence 是多少，就可以用 Gradient Descent 去 minimize 它的 Divergence。

问题就是 P_G 跟 $P_{\text{data}}(x)$ 它的 formulation 我们是不知道的，我们根本就不知道要怎么去计算它的 Divergence。所以根本不知道要怎么找一个 G 去 minimize 它的 Divergence。

这个就是 GAN 神奇的地方。在进入比较多的数学式之前我们先很直观的讲一下，GAN 到底怎么做到 minimize Divergence 这件事情。

这边的前提是我们不知道 P_G 跟 $P_{\text{data}}(x)$ 的 distribution 长什么样子，但是我们可以从这两个 distribution 里面 sample data 出来

从 $P_{\text{data}}(x)$ 去 sample distribution 出来就是把你的 database 拿出来

然后从里面 sample 很多 image 出来。

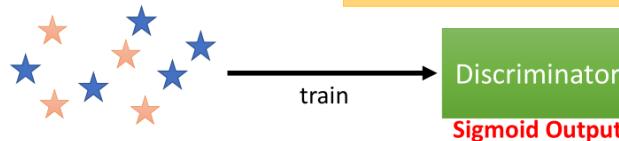
从 P_G 里面做 sample 其实就是 random sample 一个 vector，把这个 vector 丢到 generator 里面产生一张 image，这个就是从 P_G 里面做 sample。

我们可以从 P_G 和 $P_{\text{data}}(x)$ 做 sample，根据这个 sample 我们要怎么知道这两个 distribution 的 Divergence 呢？

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

- ★ : data sampled from P_{data}
- ★ : data sampled from P_G

Using the example objective function is exactly the same as training a binary classifier.



Example Objective Function for D

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

↑
(G is fixed)

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

[Goodfellow, et al., NIPS, 2014]

The maximum objective value is related to JS divergence.

GAN 神奇的地方就是通过 discriminator，我们可以量这两个 distribution 间的 Divergence。假设蓝色的星星是从 $P_{data}(x)$ 里面 sample 出来的东西，红色的星星是从 P_G sample 出来的东西。根据这些 data 我们去训练一个 discriminator，上周我们已经讲过训练 discriminator 意思就是给 $P_{data}(x)$ 的分数越大越好，给 P_G 的分数越小越好。这个训练的结果就会告诉我们 $P_{data}(x)$ 跟 P_G 它们之间的 Divergence 有多大。

我们怎么训练 discriminator 呢，我们会写一个 Objective Function D，这个 Objective Function 它跟两项有关，一个是跟 generator 有关，一个是跟 discriminator 有关。

在 train discriminator 的时候我们会 fix 住 generator，所以 G 这一项是 fix 住的，公式的意思是 x 是从 $P_{data}(x)$ 里面 sample 出来的，我们希望 $\log D(x)$ 越大越好，也就是我们希望 discriminator 的 output，假设 x 是从 $P_{data}(x)$ 里面 sample 出来的，我们就希望 $D(x)$ 越大越好。

反之假设 x 是从 generator sample 出来的，是从 P_G 里面 sample 出来的，那我们要 maximize $\log(1 - D(x))$ ，就是要 maximize $1 - D(x)$ ，也就是要 minimize $D(x)$ 。

在训练的时候就是要找一个 D，它可以 maximize 这个 Objective Function。

如果你之前 Machine Learning 有学通的话，下面这个 optimization 的式子跟 train 一个 Binary Classifier 的式子，其实是完全一模一样的。

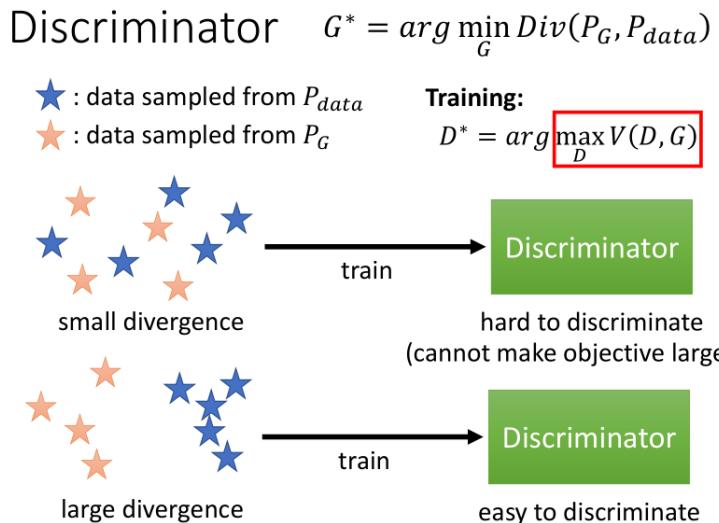
假设今天要 train 一个 Logistic Regression 的 model，Logistic Regression Model 是一个 Binary Classifier。然后就把 $P_{data}(x)$ 当作是 class 1，把 P_G 当作是 class 2，然后 train 一个 Logistic Regression Model。

你会发现你的 Objective Function 其实就是这个式子。所以这个 discriminator 在做的事情跟一个 Binary Classifier 在做的事情其实是一模一样的。

假设蓝色的点是 class 1，红色的点是 class 2。discriminator 就是一个 Binary Classifier。然后这个 Binary Classifier 它是在 minimize Cross Entropy，你其实就是在解这个 optimization 的 problem。这边神奇的地方是当我们解完这个 optimization 的 problem 的时候，你最后会得到一个最小的 loss，或者是得到最大的 objective value。

我们今天这边不是 minimize loss，而是 maximize 一个 Objective Function。这个 V 是我们的 Objective Value，我们要调 D 去 maximize 这个 Objective Value。然后这边神奇的地方是，这个 maximize Objective Value

就是把这个 D train 到最好，给了这些 data，把这个 D train 到最好，找出最大的 D 可以达到的 Objective Value。这个 value 其实会跟 JS Divergence 是有非常密切关系，你可以说这个结果它其实就是 JS Divergence。



直观的解释：你想想看，假设现在 sample 出来的 data 它们靠得很近，这个蓝色的这些星星跟红色的星星如果把它们视成两个类别的话，它们靠得很近。对一个 Binary Classifier 来说，它很难区别红色的星星跟蓝色的星星的不同，因为对一个 Binary Classifier 也就是 discriminator 来说，它很难区别这两个类别的不同，所以直接 train 下去，loss 就没有办法压低。反过来说

在 training data 上的 loss 压不下去，就是我们刚才看到的 Objective Value 没有办法把它拉得很高，没有办法找到一个 D 它让 V 的值变得很大。这个时候意味这两堆 data 它们是非常接近的，它们的 Divergence 是小的。

所以如果对一个 discriminator 来说，很难分别这两种 data 之间的不同，它很难达到很大的 Objective Value，那意味着这两堆 data 的 Divergence 是小的。所以最后你可以达到最好的 Objective Value，跟 Divergence 是会有非朝紧密的关系的。

这是一样的例子，假设蓝色的星星跟红色的星星它们距离很远，它们有很大的 Divergence，对 discriminator 来说它就可以轻易地分辨这两堆 data 的不同，也就是说它可以轻易的让你的 Objective Value，V 的这个 value 变得很大。当 V 的 value 可以变得很大的时候，意味着从 $P_{data}(x)$ 里面 sample 出来的东西和从 P_G generate 出来的东西，它们的 Divergence 是大的，所以 discriminator 就可以轻易地分辨它们的不同，discriminator 就可以轻易的 maximize Objective Value。

Math

接下来就是实际证明为什么 Objective Value 跟 Divergence 是有关系的

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

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

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

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

Assume that $D(x)$ can be any function

- Given x , the optimal D^* maximizing

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

转换成积分的形式，假设 $D(x)$ 它可以是任何的 function (实际上不见得是成立的，因为假设 $D(x)$ 是一个 network 除非它的 neural 无穷多，不然它也没有办法变成任何的 function)。

对 x 做积分中括号里面的式子，代各个不同的 x 再把它通通加起来

这就是积分在做的事情。假设 $D(x)$ 可以是任意的 function 的话，这个时候 maximize 等同于把某一个 x 拿出来，然后要找一个 D 它可以让这个式子越大越好，所有不同的 x 通通都分开来算，因为所有的 x 都是没有任何关系的，因为不管是哪一个 x 你都可以 assign 给它一个不同的 $D(x)$ 。所以积分里面的每一项都分开来算，你就可以分开为它找一个最好的 $D(x)$ 。

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

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

- Given x , the optimal D^* maximizing

$$\begin{matrix} P_{data}(x) \log D(x) & + P_G(x) \log(1 - D(x)) \\ a & D & b & D \end{matrix}$$

- Find D^* maximizing: $f(D) = a \log(D) + b \log(1 - D)$

$$\begin{aligned} \frac{df(D)}{dD} &= a \times \frac{1}{D} + b \times \frac{1}{1 - D} \times (-1) = 0 \\ a \times \frac{1}{D^*} &= b \times \frac{1}{1 - D^*} \quad a \times (1 - D^*) = b \times D^* \\ a - aD^* &= bD^* \quad a = (a + b)D^* \\ D^* = \frac{a}{a + b} &\quad \rightarrow \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} < 1 \end{aligned}$$

$P_{data}(x)$ 是固定的， P_G 也是固定的。唯一要做的事情就是找一个 $D(x)$ 的值让这个式子算起来最大。

求一下微分，找出它的 Critical Point，就是微分是 0 的地方，求一下 $D^*(x)$ 是多少即可。

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

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

$$\begin{aligned} \max_D V(G, D) &= V(G, D^*) & D^*(x) &= \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \\ &= E_{x \sim P_{data}} \left[\log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \right] & &+ E_{x \sim P_G} \left[\log \frac{P_G(x)}{P_{data}(x) + P_G(x)} \right] \\ &= \int_x P_{data}(x) \log \frac{\frac{1}{2} P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx & &+ \int_x P_G(x) \log \frac{\frac{1}{2} P_G(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx \\ &\quad + 2 \log \frac{1}{2} - 2 \log 2 & & \end{aligned}$$

接下来要做的事情就是把D*代到这个式子里面，看看Objective Function长什么样子。

为了要把整理成看起来像是 JS Divergence，就把分子跟分母都同除2，把1/2这一项把它提出来变成-2 log2。

$$\max_D V(G, D) \quad \text{JSD}(P \parallel Q) = \frac{1}{2} D(P \parallel M) + \frac{1}{2} D(Q \parallel M) \\ M = \frac{1}{2}(P + Q)$$

$$\begin{aligned} \max_D V(G, D) &= V(G, D^*) & D^*(x) &= \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \\ &= -2 \log 2 + \int_x P_{data}(x) \log \frac{P_{data}(x)}{(P_{data}(x) + P_G(x))/2} dx & &+ \int_x P_G(x) \log \frac{P_G(x)}{(P_{data}(x) + P_G(x))/2} dx \\ &= -2 \log 2 + \text{KL}\left(P_{data} \parallel \frac{P_{data} + P_G}{2}\right) + \text{KL}\left(P_G \parallel \frac{P_{data} + P_G}{2}\right) \\ &= -2 \log 2 + 2 \text{JSD}(P_{data} \parallel P_G) \quad \text{Jensen-Shannon divergence} \end{aligned}$$

后面这两项合起来就叫做JS Divergence，如果 $P_{data}(x)$ 跟 P_G 它们距离的越远这两项合起来就越大，反之它们合起来就越小。

假设 learn 一个 discriminator，写出了某一个 Objective Function，去 maximize 那个 Objective Function 后得到的结果，maximize 的那个 Objective Function，maximize 的那个 value，其实就是 $P_{data}(x)$ 跟 P_G 的 JS Divergence

当我们 train 一个 discriminator 的时候，我们想做的事情就是去 evaluate

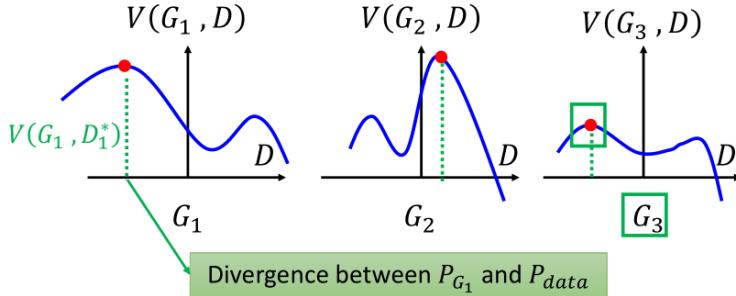
$P_{data}(x)$ 跟 P_G 这两个 distribution 之间的 JS Divergence。如果定的 Objective Function 是跟前面的式子一样的话，你就是在量 JS Divergence

如果把那个 Objective Function 写的不一样，你就可以量其它的各种不同的 Divergence。

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

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

The maximum objective value is related to JS divergence.



现在整个问题变成这个样子

本来要找一个 $G^* = \arg \min_G \text{Div}(P_G, P_{data})$, 但这个式子没有办法算。

于是我们写出一个 Objective Function $V(D, G)$, 找一个 D^* 去 maximize Objective Function, 它就是 P_G 和 $P_{data}(x)$ 之间的 Divergence

所以我们可以把 Divergence 这一项用 max 这一项把它替换掉, 变成上图第一个式子。

所以我们要找一个 generator, generate 出来的东西跟你的 data 越接近越好, 实际上要解这样一个 min max 的 optimization problem, 它实际上做的事情像是这个例子所讲的这样

假设世界上只有三个 generator, 要选一个 generator 去 minimize 这个 Objective Function, 但是可以选的 generator 总共只有三个, 一个是 G_1 一个是 G_2 一个是 G_3 。假设选了 G_1 这个 generator 的话那 $V(G_1, D)$ 就是图中这个样子, 假设这个横坐标在改变的时候, 代表选择了不同的 discriminator。

接下来的问题是我们在给定一个 generator 的时候, 我们要找一个 discriminator 它可以让 $V(G, D)$ 最大。接下来要找一个 G 去 minimize 最大的那个 discriminator 可以找到的 value。找一个 G 它可以 minimize $V(G, D)$, 用最大的 D 可以达到的 value。

现在要解这个 optimization problem, 哪一个 G 才是我们的 solution 呢? 正确答案是 G_3 。现在找出来的 G^* 就是 G_3 。

当我们给定一个 G_1 的时候, 这边这个 D_1^* 的这个高度其实就代表了 G_1 的 generator 它所 generate 出来的 distribution 跟 $P_{data}(x)$ 之间的距离。

所以 G_1 G_2 所定义的 distribution 跟 data 之间的 Divergence 比较大, 今天要 minimize Divergence 所以会选择 G_3 当作是最好的结果。

Algorithm

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

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

The maximum objective value is related to JS divergence.

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

接下来就是要想办法解这个 min max 的 problem。GAN 的这个算法就是在解这个 min max problem。解这个 min max problem 的目的就是要 minimize generator 跟你的 data 之间的 JS Divergence。

为什么这个 algorithm 是在解这一个 optimization problem?

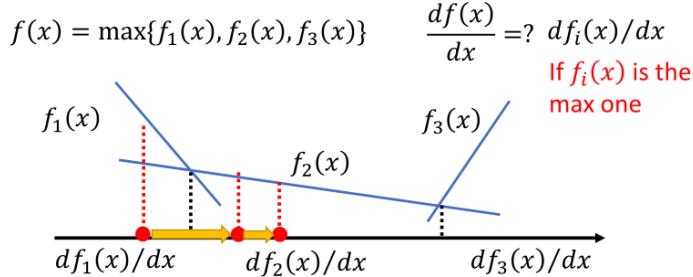
Algorithm

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

$L(G)$

- To find the best G minimizing the loss function $L(G)$,

$$\theta_G \leftarrow \theta_G - \eta \partial L(G) / \partial \theta_G \quad \theta_G \text{ defines } G$$



假设要解这个 optimization problem 的话用 $L(G)$ 来取代 $\maximize V(G, D)$ ，它其实跟 D 是没有关系的，given 一个 G 就会找到最好的 D 让 $V(G, D)$ 的值越大越好，假设最大的值就是 $L(G)$ 。

现在整个问题就变成要找一个最好的 generator G ，它可以 minimize $L(G)$ 。

它就跟 train 一般的 network 是一样的，就是用 Gradient Descent 来解它。

$L(G)$ 式子里面有 max 的，有 max 可以微分吗？

我们之前有学到一个 Maxout Network，Maxout Network 里面也有 max operation，但它显然是有办法用 Gradient Descent 解。

到底实际上是怎么做的呢

如果现在要把 $f(x)$ 对 x 做微分的话，这件事情等同于看看现在的 x 可以让哪一个 function f_1, f_2, f_3 最大，拿最大的那个出来算微分，就是 x 对 $f(x)$ 的微分。

假如你的这个 function 里面有一个 max operation，实际上在算微分的时候，你只是看现在在 f_1, f_2, f_3 里面哪一个最大，就把最大的那个人拿出来算微分。你就可以用 Gradient Descent 去 optimize 这个 $f(x)$ 。

总之就算是 Objective Function 里面有 max operation，你一样是可以对它做微分的。

Algorithm

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

$L(G)$

- Given G_0
- Find D_0^* maximizing $V(G_0, D)$ Using Gradient Ascent

$V(G_0, D_0^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G_0}(x)$

- $\theta_G \leftarrow \theta_G - \eta \partial V(G, D_0^*) / \partial \theta_G \rightarrow$ Obtain G_1 Decrease JS divergence(?)
- Find D_1^* maximizing $V(G_1, D)$

$V(G_1, D_1^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G_1}(x)$

- $\theta_G \leftarrow \theta_G - \eta \partial V(G, D_1^*) / \partial \theta_G \rightarrow$ Obtain G_2 Decrease JS divergence(?)
-

所以就回到现在要解的这个 optimization problem

一开始有一个初始的 G_0 ，接下来要算 G_0 对 $L(G)$ 的 gradient，但是在算 G_0 对 $L(G)$ 的 gradient 之前，因为 $L(G)$ 里面有 max，所以不知道 $L(G)$ 长什么样子，要把 max D 找出来。

所以假设在 given G_0 前提之下， D_0^* 可以让 $V(G_0, D)$ 最大，如果这个 D 代 D_0^* 的话，就可以得到 $L(G)$ 。可以用 Gradient Ascent 就可以找出这个 D 。

找到 D 可以 maximize 这个 Objective Function 以后，就是 $L(G)$ ，把 θ_G 对这一项算 gradient，就可以 update 参数，就得到新的 generator G1。

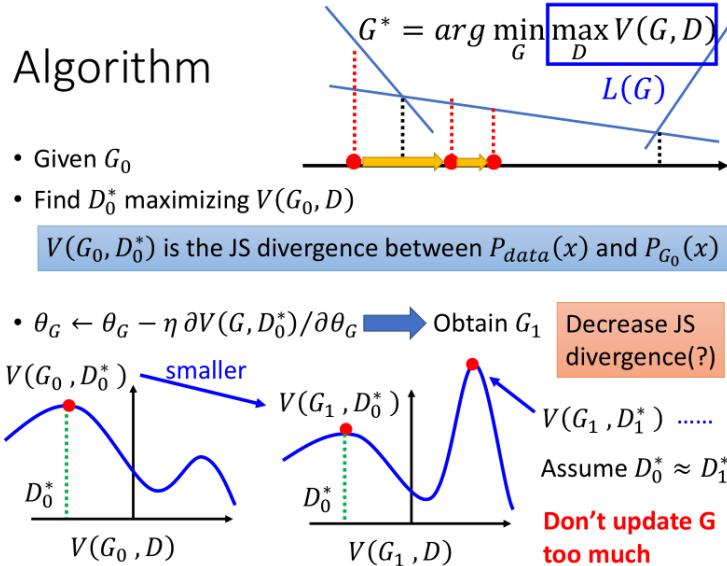
有新的 generator G1 以后，就要重新找一下最好的 D，可以让这个 $V(G_1, D)$ 最大的那个 D 假设是 D_1^* ，接下来就有一个新的 Objective Function，重新计算 gradient 再 update generator，得到 G2

这个 operation 就是有一个 G_0 ，找一个可以让 $V(G_0, D)$ 最大的 D_0^* ，就得到 V 的 function。然后让它对 G 做微分，再重新去找一个新的 D，再重新对 Objective Function 做微分。就会发现这整个 process 其实跟 GAN 是一模一样的。

你可以把它想成现在在找 D_0^* 去 maximize 这个 Objective Function 的 process，其实就是在量 $P_{data}(x)$ 跟 $P_{G_0}(x)$ 的 JS Divergence。

找到一个 D_1^* 它可以让这个 Objective Function 的值变 maximum，其实就是在计算 $P_{data}(x)$ 跟 $P_{G_1}(x)$ 的 JS Divergence。

我们求 gradient 的一项就是你的 JS Divergence，你要 update generator 去 minimize JS Divergence，这个时候你其实就是在减少你的 JS Divergence，就是在达成你的目标。



但是这边打了一个问号，因为这件事情未必等同于真的在 minimize JS Divergence。

为什么这么说，因为假设给你一个 generator G_0 ，那你的 $V(G_0, D)$ 假设它长这个样子，找到一个 D_0^* ，这个 D_0^* 的值，就是 G_0 跟你的 data 之间的 JS Divergence；但是当你 update 你的 G_0 ，变成 G_1 的时候，这个时候 $V(G_1, D)$ 它的 function 可能就会变了。本来 $V(G_0, D)$ 是这个样子， $V(G_0, D_0^*)$ 就是 G_0 跟你的 data 的 JS Divergence，今天你 update 你的 G_0 变成 G_1 ，这个时候整个 function 就变了，这个时候因为 G_0^* 仍然是固定的，所以 $V(G_1, D_0^*)$ 它就不是在 evaluate JS Divergence。我们说 evaluate JS Divergence 的 D 是 $V(G, D)$ 这个值里面最大的那个，所以当你的 G 变了，你的这个 function 就变了，当你的 function 变的时候同样的 D 就不是在 evaluate 你的 JS Divergence。如果在这个例子里面，JS Divergence 会变大。

但是为什么我们又说这一项可以看作是在减少 JS Divergence 呢？这边作的前提假设就是这两个式子可能是非常像的，假设只 update 一点点的 G 从 G_0 变到 G_1 ，G 的参数只动了一点点，那这两个 function 它们的长相可能是比较像的。因为它们的长相仍是比较像的，所以一样用 D_0^* 你仍然是在量 JS Divergence，这边本来值很小，突然变很高的情形可能是不会发生的。因为 G_0 跟 G_1 是很像的所以这两个 function 应该是比较接近。所以你可以只同样用固定的 D_0^* ，就可以 evaluate G_0 跟 G_1 的 JS Divergence。

所以在 train GAN 的时候，它的 tip 就是因为你有这个假设，就是 G_0 跟 G_1 应该是比较像的，所以在 train generator 的时候，你就不能够一次 update 太多。但是在 train discriminator 的时候，理论上应该把它 train 到底，应该把它 update 多次一点，因为你必须要找到 maximum 的值你才是在量 JS Divergence，所以 train discriminator 的时候，你其实会需要比较多的 iteration 把它 train 到底。但是 generator 的话，你应该只要跑比较少的 iteration，免得投影片上讲的假设是不成立的。

In practice ...

接下来讲一下实际上在做 GAN 的时候其实是怎么做的。

我们的 Objective Function 里面要对 x 取 expectation，但是在实际上没有办法真的算 expectation，所以都是用 sample 来代替 expectation。

实际上在做的时候，我们就是在 maximize 图中这个式子，而不是真的去 maximize 它的 expectation。

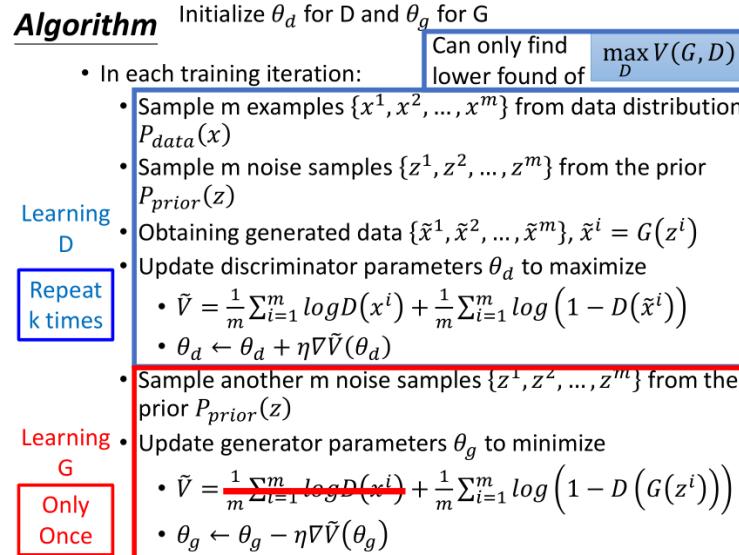
这个式子就等同于是在 train 一个 Binary Classifier

所以在实作 GAN 的时候，你完全不需要用原来不知道的东西，你在 train discriminator 的时候，你就是在 train 一个 Binary Classifier。

实际在做的时候discriminator是一个Binary Classifier，这个Binary Classifier它是一个Logistic Regression，它的output有接一个sigmoid，所以它output的值是介于0到1之间的。

然后从 $P_{data}(x)$ 里面sample m笔data出来，这m笔data就当作是positive example或是class 1的example；然后从 P_G 里面再sample另外m笔data出来，这m笔data就当作是negative example，就当作是class 2的example。接下来就train你的Binary Classifier，train一个criterion来minimize Cross Entropy，minimize Cross Entropy的式子写出来，它会等同于上面maximize这个Objective Function。

Algorithm



最后就再重新复习一次GAN的algorithm

我们之前有讲过我们train discriminator的目的是什么，是为了要evaluate JS Divergence，而当它可以让你的V的值最大的时候，那个discriminator才是在evaluate JS divergence。

所以你一定要train很多次，train到收敛为止，它才能让V的值最大，但在实作上你没有办法真的train很多次，train到收敛为止。但是你会说，我今天train d的时候，我要反复k次，这个参数要update k次，而不是像投影片上面只写update一次而已，你可能会update三次或五次才停止。

这个步骤是在解这个问题，找一个D它可以maximize V(G, D)

但是其实你没有办法真的找到一个最好的D去maximize V(G, D)，你能够找的其实只是一个lower bound而已。因为这边通常在实作的时候你没有办法真的train到收敛，你没有办法真的一直train，train到说可以让V(G, D)变的最大，通常就是train几步然后就停下来。就算我们退一万步说这边可以一直train，train到收敛，你其实也未必真的能够maximize这个Objective Function，因为在train的时候，D的capacity并不是无穷大的，你会卡在一个Local Maximum然后就结束了，你并不真的可以maximize这个式子。再退一万步说假设没有Local Maximum的问题，你可以直接解这个问题，你的D它的capacity也是有限，记得我们说过如果要量JS Divergence，一个假设是D可以是任何function，事实上D是一个network，所以它也不是任何function，所以你没有办法真的maximize V(G, D)，你能够找到的只是一个lower bound而已。但我们就假设你可以maximize这一项就是了。

接下来要train generator，我们说train discriminator是为了量JS Divergence，train generator的时候是为了要minimize JS Divergence。

为了要减少JS Divergence，下面这个式子里面你会发现第一项跟generator是没有关系的，因为第一项只跟discriminator有关，它跟generator没有关系，所以要train generator去minimize这个式子的时候，第一项是可以不用考虑它的，所以把第一项拿掉只去minimize第二项式子。这个第二个步骤就是在train generator，刚才有讲过generator不能够train太多，因为一旦train太多的话，discriminator就没有办法evaluate JS Divergence。所以generator不能train太多，你只能少量的update它的参数而已，所以通常generator update一次就好。

你可以update discriminator很多次，但是generator update一次就好。你update太多，量出来JS Divergence就不对了。所以这边就不能够update太多。

Objective Function for Generator in Real Implementation

Objective Function for Generator in Real Implementation

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

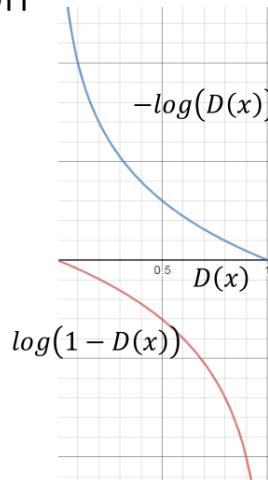
Slow at the beginning

Minimax GAN (MMGAN)

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

Real implementation:
label x from P_G as positive

Non-saturating GAN (NSGAN)



到目前为止讲说 train generator 的时候，你要去 minimize 的式子长上面这个样子。

但在 Ian Goodfellow 原始的 paper 里面，从有 GAN 以来，它就不是在 minimize 这个式子，paper 加了一小段，说这个式子 $\log(1 - D(x))$ 它长的是右边这个样子，而我们一开始在做 training 的时候 $D(x)$ 的值通常是很小的，因为 discriminator 会知道 generator 产生出来的 image 它是 fake 的，所以它会给它很小的值，所以一开始 $D(x)$ 的值会落在微分很小的地方，所以在 training 的时候，会造成你在 training 的一些问题，所以说我们把它改成这个样子。

没有为什么，它们的趋势是一样的，但是它们在同一个位置的斜率就变得不一样。一开始 $D(x)$ 还很小的时候，算出来的微分会比较大，所以 Ian Goodfellow 觉得这样子 training 是比较容易的。

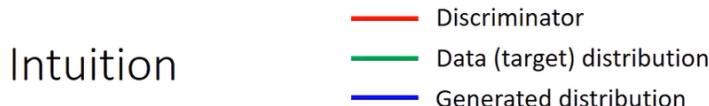
其实你再从另外一个实作的角度来看，如果你是要 minimize 上边这个式子，你会发现你需要改 code 有点麻烦。如果你是 minimize 下边这个式子你可以不用改 code。如果你是要 minimize 下面这个式子的时候，其实只是把 Binary Classifier 的 label 换过来，本来是说从 data sample 出来的是 class 1，从 generator sample 出来的是 class 2，把它 label 换过来，把 generator sample 出来的改标成 label 1，然后用同样的 code 跑下去就可以了。我认为 Ian Goodfellow 只是懒得改 code 而已，所以就胡乱编一个理由应该要用下面这个式子。（大雾 但实际上后来有人试了比较这两种不同的方法，发现都可以 train 得起来，performance 也是差不多的，不知道为什么 Ian Goodfellow 一开始就选了这个。

后来 Ian Goodfellow 还写了另外一篇文章，把上面这个叫做 Minimax GAN 就是 MMGAN，把下面这个叫做 Non-saturating GAN 就是 NSGAN。

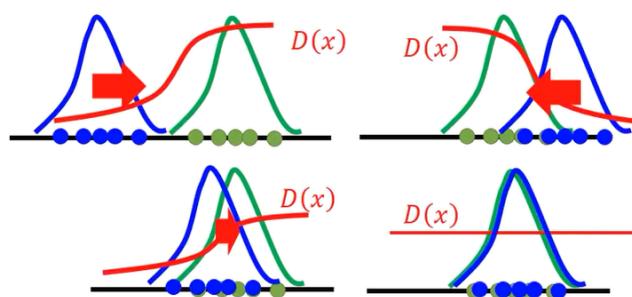
Intuition

现在讲一些比较直观的东西

所以按照 Ian Goodfellow 的讲法今天这个 generator 和 discriminator 它们之间的关系是什么样呢？



- Discriminator leads the generator



绿色的点是你的目标，蓝色的点是 generator 产生出来的东西

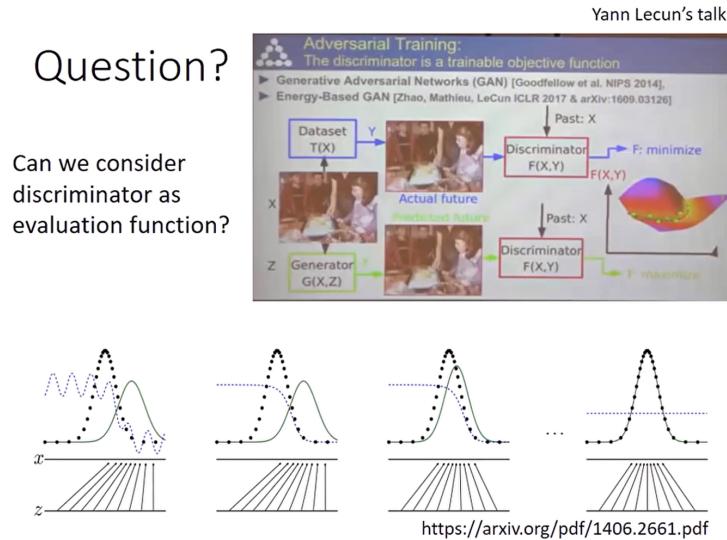
背景的颜色是 discriminator 的值，discriminator 会 assign 给每一个 space 上的 x 一个值，背景的这个颜色是 discriminator 的值。

你就会发现这个 discriminator 就把 P_G 产生出来蓝色的点赶来赶去，直到最后蓝色的点跟绿色的点重合在一起的时候，discriminator 就会坏掉，因为完全没有办法分辨 generator 跟 discriminator 之间的差别。

Question

会不会出现 data imbalance ?

一般在做的时候，在 train 一个 classifier 的时候其实会害怕 data imbalance 的问题，今天在这个 task 里面，data 是自己 sample 出来的，我们不会给自己制造 data imbalance 的问题，所以两种 task 会 sample 一样的数目，假设从 generator 里面 generate 256 笔 data，那你今天从你的 sample 的 database 里面你也会 sample 256 笔 data。



你不觉得今天讲的跟上周讲的是有点矛盾的吗

如果按照 Ian Goodfellow 的讲法，最后 discriminator train 到后来

它就会烂掉变成一个水平线，但我们说 discriminator 其实就是 evaluation function，也就是说 discriminator 的值代表它想要评断这个 object，generate 出来的东西它到底是好还是不好。

如果 discriminator 是一条水平线，它就不是一个 evaluation function，对它来说所有的东西都是一样好，或者是一样坏。

右上角是 Yann LeCun 画的图，这个图就是 discriminator 的图，绿色的点就是 real data 分布，你发现他在画的时候，在他的想象里面 discriminator 并没有烂掉变成一个水平线，而是有 data 分布的地方它会得到比较小的值，而没有 data 分布的地方它会得到比较大的值。跟之前讲的是相反的，不过意思完全是一样的。

跟 Ian Goodfellow 讲的是有一些矛盾的，这个就是神奇的地方，因为这个都是尚待发展中的理论，所以有很多的问题是未知的。

以前在 train Deep Learning 的时候，我们都要用 Restricted Boltzmann Machine，过去我们都相信没有 Restricted Boltzmann Machine 是 train 不起来的，但现在根本就用不上这个技术。

所以这个变化是非常快的，也许明年再来讲同样东西的时候，就会有截然不同的讲法也说不定。

你如果问我到底是哪一种的话，假设你硬要我给你一个答案，告诉你到底应该是 Ian Goodfellow 讲得比较对，还是 Yann LeCun 讲得比较对。我的感觉是首先可以从实验上来看看，如果你真的 train 完你的 GAN，然后去 evaluate 一下 discriminator，它的感觉好像是介于这两个 case 之间，它绝对不是烂掉，绝对不是变成一个完全烂掉的 discriminator。

你自己回去做做看，几时 train 出这样的结果，虽然是这种简单的例子你也 train 不出这个结果的，就算是一维的例子也都做不出这个结果。所以不太像是 Ian Goodfellow 讲的这样。但是 discriminator 也不完全反映了 data distribution，感觉是介于这两个 case 之间。

这些观点到底对我们了解 GAN 有什么帮助？

也许 GAN 的 algorithm 就是一样，那算法就是那个样子，就是 train generator、train discriminator、iterative train，也许它的 algorithm 是不会随着你的观点不同。

但是你用不同的观点来看待 GAN，你其实在设计 algorithm 的时候，中间会有些微妙的差别，也许这些微妙的差别导致最后 training 的结果会是很不一样的。

我觉得也许 Yann LeCun 的这个讲法，之前讲的discriminator 是在 evaluate 一个 object 的好还是不好，它是在反映了 data distribution 这件事也许更接近现实。

为什么会这么说？

首先，你在文献上会看到很多人会把 discriminator 当作 classifier 来用，所以先 train 好一个 GAN，然后把 discriminator 拿来做其它事情。假设 discriminator train 到最后，按照 Ian Goodfellow 猜想会烂掉的话，拿它来当作 pre-training 根本就没有意义，但很多人会拿它当作 pre-training，也显示它是有用的，所以它不太可能真的 train 到后来就坏掉。这个是第一个 evidence。

另外一个 evidence 是你想看你在 train GAN 的时候，你并不是每一次都重新 train discriminator，而是会拿前一个 iteration 的 discriminator，当作下一个 iteration 的 initialize 的参数。如果你的 discriminator 是想要衡量两个 data distribution 的 Divergence 的话，你其实没有必要把前一个 iteration 的东西拿来用，因为 generator 已经变了，保留前一个 iteration 的东西有什么意义呢？这样感觉是不太合理的。也有人可能会说因为 generator update 的参数，update 的量是比较小的，所以也许把前一个 time step 得到的 generator，当作下一个 time step 的 initialization，可以加快 discriminator 训练的速度，也说不定，这个理由感觉也是成立的。

不过在文献上我看到有人在 train GAN 的时候它有一招，每次 train 的时候它不只拿现在的 generator 去 sample data，它也会拿过去的 generator 也 sample data，然后把这些各个不同 generator sample 的 data 通通集合起来，再去 train discriminator，可以得到的 performance 会是比较好的。

如果 discriminator 是在 evaluate 现在的 generator，跟 data distribution 的差异的话，好像做这件事情也没有太大的意义，因为现在量 generator 跟 data 之间的差异，拿过去 generator 产生的东西有什么用？没什么用。但是在实作上发现拿过去 generator 产生的东西，再去训练 discriminator 是可以得到比较好的成果。所以这样看起来，也许这是另外一个 support 支持也许 discriminator 在做的事情，并不见得是在 evaluate 两个 distribution 之间的 Divergence。

不过至少 Ian Goodfellow 一开始是这么说的，所以我们把 GAN 最开始的理论告诉大家。

fGAN: General Framework of GAN

我们定某种 objective function，就是在量 js divergences。那我们能不能够量其他的 divergence 呢？

fGAN 就是要告诉我们怎么量其他的 divergences。

这个东西有点用不上，原因就是，fGAN 可以让你用不同的 f divergences 来量你 generated 的 example 跟 real example 的差距。但是用不同的 x divergences 的结果是差不多的，所以这一招好像没什么特别有用的地方。

但是我们还是跟大家介绍一下，因为这个在数学上，感觉非常的厉害，但是在实作上，好像没什么特别的不同。

fGAN 想要告诉我们的，其实不只是用 js divergence，任何的f-divergence都可以放到 GAN 的架构里面去。

f-divergence

f-divergence P and Q are two distributions. $p(x)$ and $q(x)$ are the probability of sampling x .

$$D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \quad \begin{array}{l} f \text{ is convex} \\ f(1) = 0 \end{array} \quad D_f(P||Q) \text{ evaluates the difference of } P \text{ and } Q$$

If $p(x) = q(x)$ for all x

smallest $D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx = 0$

$$\begin{aligned} &= 1 \\ &= 0 \end{aligned}$$

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

Because f is convex $\geq f\left(\int_x q(x)\frac{p(x)}{q(x)}dx\right)$

$$= f(1) = 0$$

If P and Q are the same distributions,
 $D_f(P||Q)$ has the smallest value, which is 0

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

有两个条件， f is convex and $f(1) = 0$

f-divergence

$$D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \quad \begin{array}{l} f \text{ is convex} \\ f(1) = 0 \end{array}$$

$$f(x) = x \log x$$

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

$$f(x) = -\log x$$

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

$$f(x) = (x - 1)^2$$

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

假设 f 带不同的式子，你就得到各式各样的 f -divergence 的 measure。

Fenchel Conjugate

Fenchel Conjugate

$$D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \quad \begin{array}{l} f \text{ is convex} \\ f(1) = 0 \end{array}$$

- Every convex function f has a conjugate function f^*

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

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

$$x_1 \mathbf{t}_1 - f(x_1)$$

$$x_2 \mathbf{t}_1 - f(x_2)$$

$$x_3 \mathbf{t}_1 - f(x_3)$$

t_1

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

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

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

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

t_2

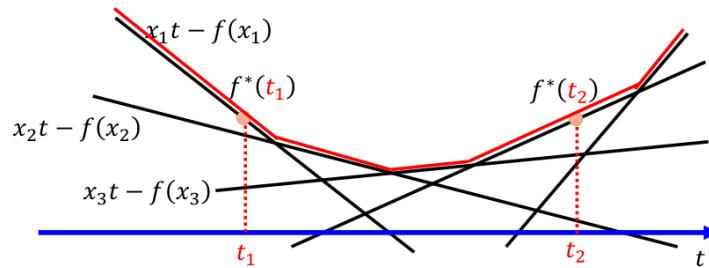
要知道 $f^*(t)$ 长什么样子，就把 t 的每一点，通通这个方法去算。

Fenchel Conjugate

$$D_f(P||Q) = \int_x q(x)f\left(\frac{p(x)}{q(x)}\right)dx \quad \begin{array}{l} f \text{ is convex} \\ f(1) = 0 \end{array}$$

- Every convex function f has a conjugate function f^*

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

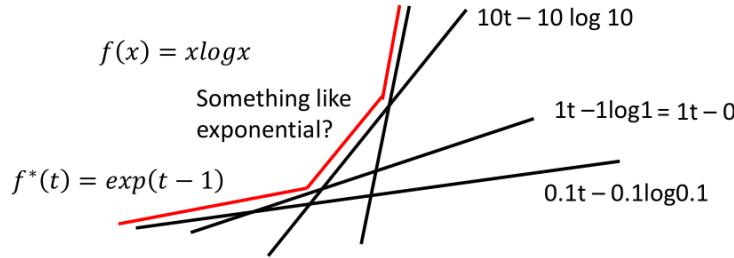


也可以用这个方法把 $f^*(t)$ 画出来。就是把所有不同的 x 所造成的直线，通通画出来，然后再取它们的 upper bound。

所以今天你会发现 $f^*(t)$ 一定是 convex 的，如果很多条直线，随便乱画，不管你画得怎么样，最后你只要找的是 upper bound，得到的 function 都是 convex 的。

- Every convex function f has a conjugate function f^*

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



你从 0.1 带进去，得到一条线，0.1001 带进去，也是一条线，0.1002 带进去，也是一条线，通通带进去，你得到无穷无尽的线。把这些所有的线 upper bound 都找出来，就是红色这一条线，会发现说这条红色的线，它看起来像是 exponential。这一条红色的线，它是 $\exp(t-1)$ 。所以 $f(x) = x \log x$ 的 conjugate，就是 $\exp(t-1)$ 。

Proof

- Every convex function f has a conjugate function f^*

- $(f^*)^* = f$

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

$$f(x) = x \log x \longleftrightarrow f^*(t) = \exp(t-1)$$

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

$$g(x) = xt - x \log x \quad \text{Given } t, \text{ find } x \text{ maximizing } g(x)$$

$$t - \log x - 1 = 0 \quad x = \exp(t-1)$$

$$f^*(t) = \exp(t-1) \times t - \exp(t-1) \times (t-1) = \exp(t-1)$$

Connection with GAN

$$\begin{aligned} f^*(t) &= \max_{x \in \text{dom}(f)} \{xt - f(x)\} \longleftrightarrow f(x) = \max_{t \in \text{dom}(f^*)} \{xt - f^*(t)\} \\ D_f(P||Q) &= \int_x q(x) f\left(\frac{p(x)}{q(x)}\right) dx \quad \boxed{\frac{p(x)}{q(x)}} \quad \boxed{\frac{p(x)}{q(x)}} \\ &= \int_x q(x) \left(\max_{t \in \text{dom}(f^*)} \left\{ \frac{p(x)}{q(x)} t - f^*(t) \right\} \right) dx \\ &\approx \max_D \int_x p(x) D(x) dx - \int_x q(x) f^*(D(x)) dx \end{aligned}$$

$$\begin{aligned} D_f(P||Q) &\geq \int_x q(x) \left(\frac{p(x)}{q(x)} D(x) - f^*(D(x)) \right) dx \\ &= \int_x p(x) D(x) dx - \int_x q(x) f^*(D(x)) dx \end{aligned}$$

D is a function
whose input is x,
and output is t

我们 learn 一个 D ，它就是 input 一个 x ，它 output 的这个 scalar，就是这边这个 t ，所以我们把这个 t 用 $D(x)$ 取代掉。

所以我们希望可以 learn 出一个 function，这个 discriminator 帮我们解这个 max 的 problem，input 一个 x ，它告诉我们说，你现在 input 这个 x 后，到底哪一个 t ，可以让这个值最大。 D 就是要做这件事。

但是因为假设 D 的 capacity 是有限的，那你今天把这个 t 换成 $D(x)$ ，就会变成是 f -divergence 的一个 lower bound。

所以我们找一个 D ，它可以去 maximize 这一项，它就可以去逼近 f -divergence。

$$\begin{aligned}
D_f(P||Q) &\approx \max_{\mathbf{D}} \int_x p(x) D(x) dx - \int_x q(x) f^*(D(x)) dx \\
&= \max_{\mathbf{D}} \{E_{x \sim P}[D(x)] - E_{x \sim Q}[f^*(D(x))]\} \\
&\quad \text{Samples from P} \qquad \text{Samples from Q} \\
D_f(P_{data}||P_G) &= \max_{\mathbf{D}} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[f^*(D(x))]\} \\
G^* &= \arg \min_G D_f(P_{data}||P_G) \quad \text{Original GAN has different } V(G,D) \\
&= \arg \min_G \max_D \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[f^*(D(x))]\} \\
&= \arg \min_G \max_D V(G, D) \quad \text{familiar? 😊}
\end{aligned}$$

变成期望的形式，把 p 改成 p data，把 Q 改成 PG。

所以，p data 跟 PG 之间的 f-divergence，就可以写成这个式子。f-divergence 是什么，就会影响到这个 f^* 是什么。

所以今天假如你的 f-divergence 是 KL divergence，那你就看 KL-divergence f^* 是什么。KL divergence f 是 $x \log x$ ，它的 f^* 是 $\exp(t-1)$ ，所以这个 f^* 就带 $\exp(t-1)$ 。

这个式子跟 GAN 看起来的式子，看起来很像。

想想看我们今天在 train 一个 generator 的时候，我们要做的事情，就是去 minimize 某一个 divergence。而这个 divergence，我们就可以把它写成这个式子。随着你要用什么 divergence，你这 f^* 就换不同的式子，你就是在量不同的 divergence。

而这个东西就是我们说在 train GAN 的时候，你要用 discriminator 去 maximize 你的 generator 去 minimize 的 objective function V of (G, D) 。只是 $V(G, D)$ 的定义不同，就是在量不同的 divergence。

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

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	$\int (p(x) - q(x)) \log \frac{p(x)}{q(x)} dx$	$(u - 1) \log u$
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	$\int p(x) \log \frac{2p(x)\pi+(1-\pi)q(x)}{p(x)+q(x)} + (1-\pi)q(x) \log \frac{2q(x)\pi+(1-\pi)p(x)}{p(x)+q(x)} dx$	$\pi u \log u - (1-\pi + \pi u) \log(1-\pi + \pi u)$
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	$\frac{2}{2-2\sqrt{1-t}}$
Squared Hellinger	$\frac{1}{4}t$
Jeffrey	$W(e^{1-t}) + \frac{1}{W(e^{1-t})} + t - 2$
Jensen-Shannon	$-\log(2 - \exp(t))$
Jensen-Shannon-weighted	$(1-\pi) \log \frac{1-\pi}{1-\pi+u} - \log(1 - \exp(t))$
GAN	

Using the f-divergence you like 😊

<https://arxiv.org/pdf/1606.00709.pdf>

这边就是从 paper 上面的图，它就告诉你说各种不同的 divergence 的 objective function。

那可以 optimize 不同的 divergence，到底有什么厉害的地方呢？

Mode Collapse

也许这一招可以解决一个长期以来困扰大家的问题是

当你 train GAN 的时候你会遇到一个现象叫做，Mode Collapse

Mode Collapse 的意思是说你的 real data 的 distribution 是比较大的，但是你 generate 出来的 example，它的 distribution 非常的小。

Mode Collapse



举例来说，你在做二次元人物生成的时候，如果你 update 的 iteration 太多，你得到的结果可能会某一张特定的人脸开始蔓延，变得到处都是这样，但它这些人脸，其实是略有不同的，有的比较偏黄，有的比较偏红，但是他们都是看起来就像是同一张人脸。也就是说你今天产生的 distribution 它会越来越小，而最后会发现同一张人脸不断的反复出现，这个 case，叫做 Model collapse。

Mode Dropping

那有另外一个 case 比 mode collapse 稍微轻微一点叫做 Mode dropping。

意思是说你的 distribution 其实有很多个 mode，假设你 real distribution 是两群，但是你的 generator 只会产生同一群而已，他没有办法产生两群不同的 data。

举例来说，你可能 train 一个人脸产生的系统

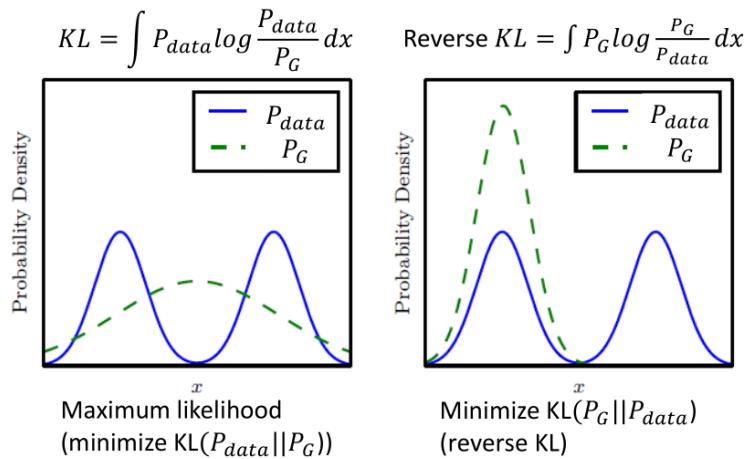
你在 update 一次 generator 参数以后，产生的 image，他没有产生黄皮肤的人，他只有产生肤色比较偏白的人；但是你 update 一次，它就变成产生黄皮肤的人，就没产生白皮肤的人；再 update 一次，它就变成产生黑皮肤的人。他每次都只能产生某一种肤色的人。

那为什么会发生这种现象呢？

一个远古的猜测是也许是因为我们 divergence 选得不好。

Modified from Ian Goodfellow's tutorial

Flaw in Optimization?



如果今天你的 data 的 distribution 是蓝色的分布，你的 generator 的 distribution，它只能有一个 mixture，它是绿色的虚线分布。

如果你选不同的 divergence，你最后 optimize 的结果，最后选出来可以 minimize divergence 的那个 generator distribution，会是不一样的。

假设你用 maximum likelihood 的方法，去 minimize KL divergence，那你的 generator 最后认为最好的那个 distribution 长左边这个样子。

假设你的 generator distribution 长的是这个样子，你从它里面去 sample data，你 sample 在 mixture-mixture 之间，结果反而会是差的。

所以这个可以解释为什么，过去没有 GAN 的时候，我们是在 minimize KL divergence，我们是在 maximize likelihood，我们产生的图片会那么模糊。

也许就是因为我们的 distribution 是这个样子的，我们在 sample 的时候其实并不是真的在 data density 很高的地方 sample，而是会 sample 到 data density 很低的地方，所以这地方就对应到模糊的图片。

那有人就说，如果你觉得是 KL divergence 所造成的，那如果换别的 divergence，比如说你换 reverse KL divergence。

那你就会发现说，对 generator 来说最好的 distribution 是完全跟某个 mode 一模一样，就因为如果你看这个 reverse KL divergence 的式子，

你就会发现说，对它来说，如果他产生出来的 data 是蓝色 distribution 没有涵盖它的 penalty 比较大，所以如果你今天选择的是 reverse KL divergence，那你的那个 generator，它就会选择集中在某一个 mode 就好，而不是分散在不同的 mode。

而我们传统的 GAN 的那个 JS divergence，它比较接近 reverse KL divergence，这也许解释了为什么你 train GAN 的时候，会有 mode collapse 或者是 mode dropping 的情形。

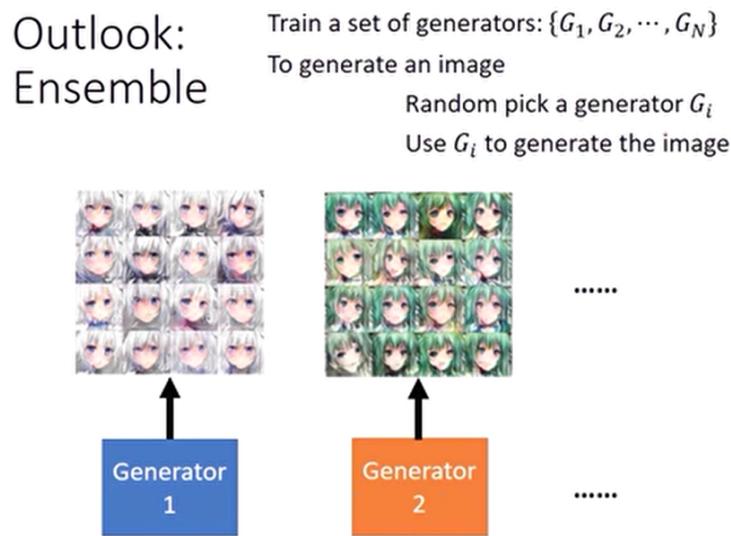
因为对你的 generator 来说，产生这种 mode collapse 或 mode dropping 的情形其实反而是比较 optimal 的。

所以今天 fGAN 厉害的地方就是，如果你觉得是 JS divergence 的问题，你可以换 KL divergence。但结果就是，换不同的 divergence，mode dropping 的 case 状况还是一样，所以看起来不是 mode dropping 或 mode collapse 的问题，并不完全是选择不同的 divergence 所造成的。

那你可能会问说，那我要怎么解决 mode collapse 的问题呢？

你很可能会遇到 mode collapse 的问题，你的 generator 可能会产生出来的图通通都是一样的。

那要怎么避免这个情形呢？就是做 Ensemble



什么意思呢？今天要你产生 25 张图片，你就 train 25 个 generator

然后你的每一个 generator 也许它都 mode collapse，但是对使用者来说，使用者并不知道你有很多个 generator，那所以你产生的结果，看起来就会 diverse。这是一个我觉得最有效可以避免 mode collapse 的方法。

Tips for Improving GAN

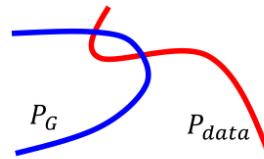
JS divergence is not suitable

In most cases, P_G and P_{data} are not overlapped.

1. The nature of data

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

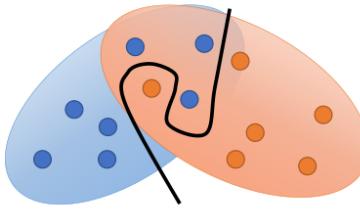
The overlap can be ignored.



2. Sampling

Even though P_{data} and P_G have overlap.

If you do not have enough sampling



最原始的 GAN，他量的是 generated data 跟 real data 之间的 JS divergence。但是用 JS divergence 来衡量的时候，其实有一个非常严重的问题。

你的 generator 产生出来的 data distribution，跟你的 real data 的 distribution，往往是没有重叠的。

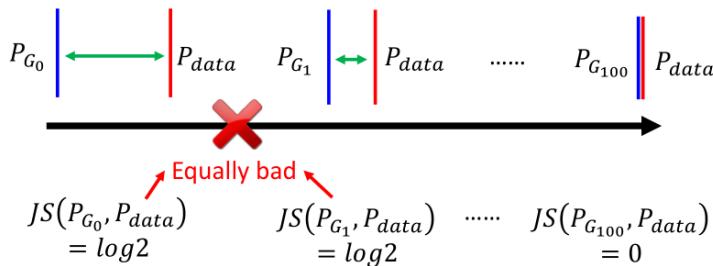
为什么 generate 出来的 data，跟 real 的 data，往往是没有重叠的呢？

一个理由是，data 本质上问题。因为我们通常相信 image 实际上在高维空间中的分布，其实是低维的一个 manifold。在一个高维空间中的两个低维的 manifold，它们的 overlap 的地方几乎是忽略的，你有两条曲线，在一个二维的平面上，他们中间重叠的地方几乎是忽略的。

从另外一个角度，我们实际上在衡量 PG 跟 Pdata 的 divergence 的时候，我们是先做 sample，我们从两个 data distribution 里面做一些 sample 得到两堆 data，再用 discriminator 去量他们之间的 divergence。

那所以我们现在就算你的 PG 跟 Pdata 这两个 distribution 是有 overlap 的，但是你是先从这两个 distribution 里面做一些 sample，而且 sample 的时候，你也不会 sample 太多。也就是从红色 distribution sample 一些点，从蓝色 distribution 再 sample 一些点。这两堆点，它们的 overlap 几乎是不会出现的，除非你 sample 真的很多，不然这两堆点其实完全就可以视为是两个没有任何交集的 distribution。所以就算本质上 Pdata 跟 PG 有 overlap，但你在量 divergence 的时候，你是 sample 少量的 data 出来才量 divergence，那在你 sample 出来的少量 data 里面，PG 跟 Pdata，看起来就是没有重合的。

What is the problem of JS divergence?



JS divergence is $\log 2$ if two distributions do not overlap.

Intuition: If two distributions do not overlap, binary classifier achieves 100% accuracy

→ Same objective value is obtained. → Same divergence

当 PG 跟 Pdata 没有重合的时候，你用 JS divergence 来衡量 PG 跟 Pdata 之间的距离，会对你 training 的时候，造成很大的障碍。

因为 JS divergence 它的特性是：如果两个 distribution 没有任何的重合，算出来就是 $\log 2$ ，不管这两个 distribution 实际上是不是有接近，只要没有重合，没有 overlap，算出来就是 $\log 2$ 。

所以假设你的 Pdata 是红色这一条线，虽然实际上 G1 其实是比 G0 好的，因为 G1 产生的 data，其实相较于 G0 更接近 real data distribution。但从 JS divergence 看起来，G1 和 G0 是一样差的，除非说现在你的 G100 跟 Pdata 完全重合，这时候 JS divergence，算出来才会是 0。

只要没有重合，他们就算是非常的靠近，你算出来也是 $\log 2$ 。所以这样子会对你的 training 造成问题。

因为我们知道说我们实际上 training 的时候，generator 要做的事情就是想要去 minimize 你的 divergence。

你用 discriminator 量出 divergence，量出 JS divergence，或其他 divergence 以后，generator 要做的事情是 minimize 你的 divergence。那对 generator 来说，PG0 跟 PG1 他们其实是一样差的。所以对 generator 来说，他根本就不会把 PG0 update 成 PG1。所以你没有办法把 PG0 update 到 PG1，你最后也没有办法 update 到 PG100，因为在 PG0 的地方就卡住了，他没有办法 update 到 PG1。

所以你今天是用 JS divergence 来衡量两个 distribution，而恰好这两个 distribution 又没有太多重叠，他们重叠几乎可以无视的时候，你会发现，你 train 起来是有问题的。

从另外一个直觉的方向来说，为什么今天只要两个 distribution 没有重合，他们算出来的 loss，他们量出来的 divergence 就会一样。

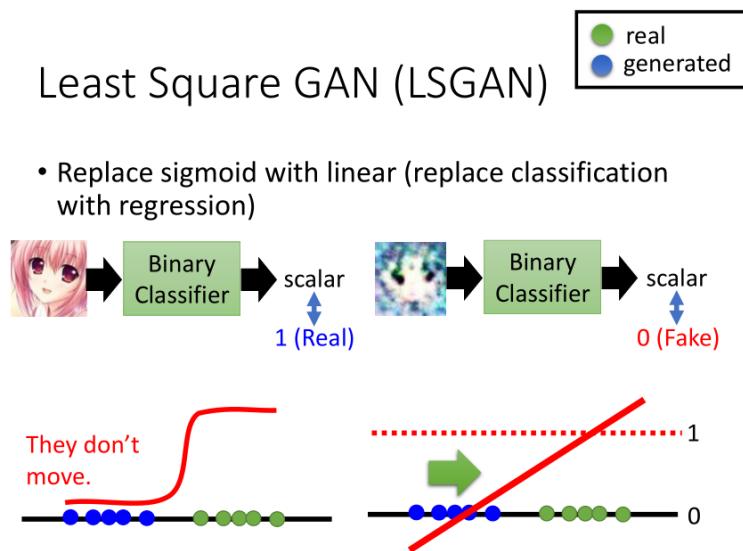
因为你想看，我们今天实际上在量 JS divergence 的时候，我们做的事情是什么？

我们有两群 data，把它视为是两个 class，learn 一个 discriminator，你用 minimize cross entropy 当成你的 loss function，去分别出这两组 data 之间的差异。但假设你 learn 的是一个 binary 的 classifier，其实只要这两堆 data，没有重合，它的 loss 就是一样的。

因为假设这两堆 data 没有重合，binary 的 classifier，假如它 capacity 是无穷大，它就可以分辨这两堆 data。在这两堆 data 都可以分辨的前提下，你算出来的 loss，其实会是一样大或者是一样小的。在 train binary classifier 的时候，你 train 到最后得到的那个 loss，或是 objective value 其实就是你的 JS divergence。

今天如果你的 binary 的 classifier，在 G1 这个 case 和 G2 这个 case，它都可以完全把两堆 data 分开。它算出来的 objective 都是一样大，它算出来的 loss 都是一样小的。那意味着，你量出来的 divergence，就是这样。

Least Square GAN (LSGAN)



在原始的 GAN 里面，当你 train 的是一个 binary classifier 的时候，你会发现，你是比较难 train 的。

用另外一个直观的方法来说明

这个 binary classifier 会给蓝色的点 0 分，绿色的点 1 分。我们知道我们的 binary classifier 它的 output 是 sigmoid function，所以它在接近 1 这边特别平，它在接近 0 这边特别平。

那你 train 好这个 classifier 以后，本来我们期待，train 一个 generator，这个 generator 会带领这些蓝色的点，顺着这个红色的线的 gradient，就 generator 会顺着 discriminator 给我们的 gradient，去改变它的 generated distribution。

所以我们本来是期待 generator，会顺着这个红色线的 gradient，把蓝色的点往右移。但实际上你会发现，这些蓝色的点是不动的，因为在这些蓝色的点附近的 gradient 都是 0。

如果你今天是 train 一个 binary 的 classifier，它的 output 有一个 sigmoid function 的话，他在蓝色的点附近，它是非常平的，你会发现说他的微分几乎都是 0，你根本就 train 不动它。

所以你真的直接 train GAN，然后 train 一个 binary classifier 的话，你很容易遇到这样子的状况。

过去的一个解法是说，不要把那个 binary classifier train 的太好。

因为如果你 train 的太好的话，它把这些蓝色的点，都给他 0，这边就会变得很平，绿色点都给它 1，就会变得很平。不要让它 train 的太好，不要 update 太多次，让它在这边仍然保有一些斜率。

那这样的问题就是，什么叫做不要 train 的太好，你就会很痛苦，你搞不清楚什么叫做不要 train 的太好，你不能够在 train discriminator 的时候太小力，太小力没办法分别 real 跟 fake data；太大力也不行，太大力的话你就会陷入这个状况，你会陷入这个微分是 0，没有办法 train 的状况。

但是什么叫做不要太大力，不要太小力，你就会很难控制。

那在早年还没有我们刚才讲的种种 tip 的时候，GAN 其实不太容易 train 起来，所以你 train 的时候通常就是，你一边 update discriminator，然后你就一边吃饭，然后你就看他 output 的结果，每 10 个 iteration 就 output 一次结果，我要看它好不好，如果发现结果不好的话，就重做这样子。

所以来就有一个方法，叫做 Least Square GAN (LSGAN)，那 LSGAN 做的事情，就是把 sigmoid 换成 linear。

这样子你就不会有这种在某些地方特别平坦的情形，因为你现在的 output 是 linear 的。

那我们本来是一个 classification problem，现在把 output 换成了 linear 以后呢，它就变成一个 regression 的 problem。

这 regression problem 是说如果是 positive 的 example，我们就让它的值越接近 1 越好，如果是 negative example，我们就让它的值越接近 0 越好。

但其实跟原来 train binary classifier 是非常像的，只是我们把 sigmoid 拆掉，把它变成 linear。

Wasserstein GAN (WGAN)

那今天很多人都会用的一个技术，叫做 WGAN。

WGAN 是什么呢？在 WGAN 里面我们做的事情是我们换了另外一种 evaluation 的 measure 来衡量 Pdata 跟 PG。

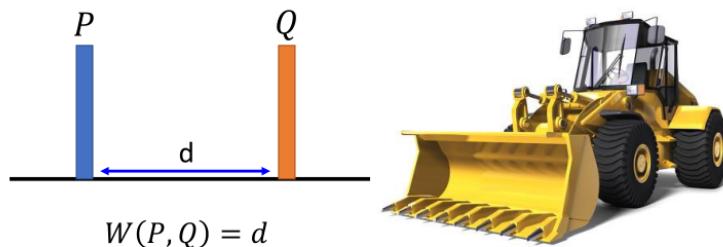
我们之前说在原来的 GAN 里面要衡量 Pdata 跟 PG 的差异，用的是 JS divergence。

在我们讲 fGAN 的时候我们说，你不一定要用 JS divergence，你其实可以用任何其他的 f divergence，在 WGAN 里面用的是 Earth Mover's Distance 或叫 Wasserstein Distance 来衡量两个 distribution 的差异。它其实不是 f divergence 的一种，所以在 fGAN 那个 table 里面，其实是没有 WGAN 的。

所以这边是另外不一样的方法。同样的地方是，就是你换了一个 divergence 来衡量你的 generated data 和 real data 之间的差异。

Earth Mover's Distance

Considering one distribution P as a pile of earth,
and another distribution Q as the target
The average distance the earth mover has to move
the earth.



那我们先来介绍一下，什么是 Earth Mover's Distance

Earth Mover's Distance 的意思是这样，假设你有两堆 data，这两个 distribution 叫做 P and Q，Earth Mover's Distance 的意思是说，你就想象成你是在开一台推土机，那你的土从 P 的地方铲到 Q 的地方。

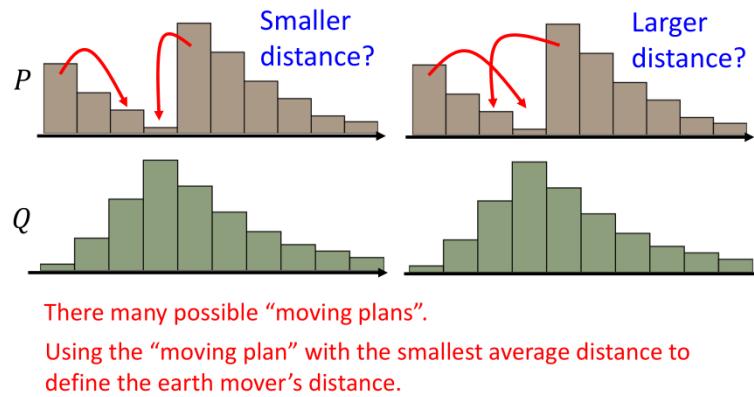
P 的地方是一堆土，Q 的地方是你准备要把土移过去的位置。然后你看推土机把 P 的土铲到 Q 那边，所走的平均的距离，就叫做 Earth Mover's Distance，就叫做 Wasserstein Distance。

那这个 Wasserstein Distance 怎么定义呢？

如果是在这个非常简单的 case，我们假设 P 的 distribution 就集中在一维空间中的某一个点，Q 的 distribution，也集中在一维空间中的某一个点。

如果你要开一台推土机把 P 的土挪到 Q 的地方去，那假设 P 跟 Q 它们之间的距离是 d，那你的 Wasserstein Distance，P 这个 distribution 跟 Q distribution 的 Wasserstein Distance 就等于 d。

但是实际上你可能会遇到一个更复杂的状况



假设你 P distribution 是长这个样子

假设你 Q distribution 是长这个样子

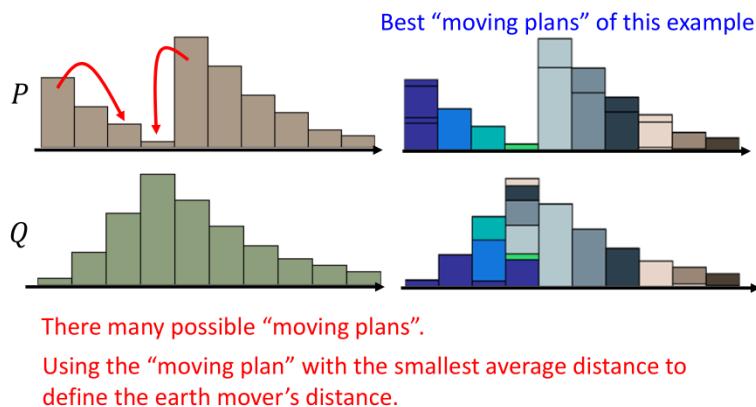
那如果你今天要衡量这两个 distribution 之间的

Earth Mover's Distance, 假设你要衡量他们之间的 Wasserstein Distance, 怎么办呢?

你会发现当你要把 P 的土铲到 Q 的位置的时候, 其实有很多组不同的铲法。推土机走的平均距离是不一样的, 这样就会变成说同样的两个 distribution 推土机走的距离不一样, 你不知道哪个才是 Wasserstein Distance。

我们说你把某一堆土, 铲到你目标的位置去, 平均所走的距离就是, Wasserstein Distance。

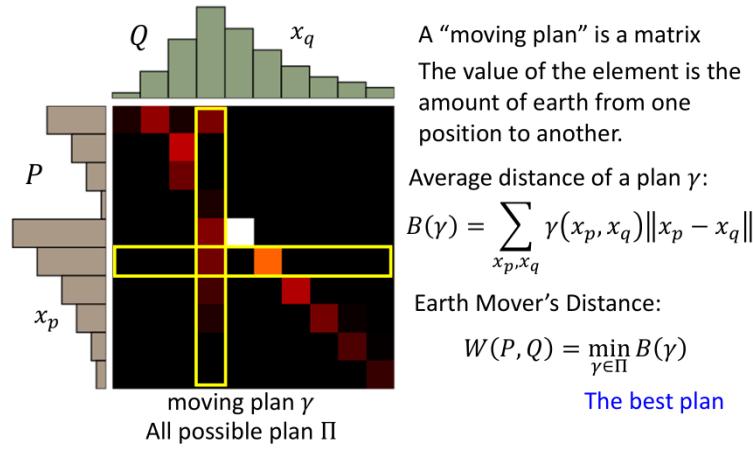
但现在的问题就是, 铲土的方法有很多种, 到底哪一个才是 Wasserstein Distance 呢?



所以今天 Wasserstein Distance 实际上的定义是, 穷举所有可能铲土的方法。每种铲土的方法, 我们就叫它一个 moving plan, 叫它一个铲土的计划。

穷举出所有铲土的计划, 有的可能是比较有效的, 有的可能是舍近求远的。每一个铲土的计划, 推土机平均要走的距离通通都算出来, 看哪一个距离最小, 就是 Wasserstein Distance。

那今天在这个例子里面, 其实最好的铲土的方法, 是像这个图上所示这个样子。这样你用这一个 moving plan 来挪土的时候, 你的推土机平均走的距离是最短的, 这个平均走的距离就是 Wasserstein Distance。



这是一个更正式的定义，假设你要把这个 P 的图挪到 Q 这边，那首先你要定一个 moving plan。那什么是一个 moving plan 呢？

moving plan 其实你要表现它的话，你可以把它化做是一个 matrix。

今天这个矩阵，就是某一个 moving plan，我们把它叫做 γ

那在这个矩阵上的每一个 element，就代表说，我们要从纵坐标的这个位置挪多少土到横坐标的这个位置。这边的值越亮，就代表说，我们挪的土越多。

实际上你会发现你把 column\row 这些值合起来就会变成 bar 的高度

接下来的事情是，假设给你一个 moving plan 叫做 γ ，你会不会算用这个 moving plan，挪土的时候要走多少距离呢？

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

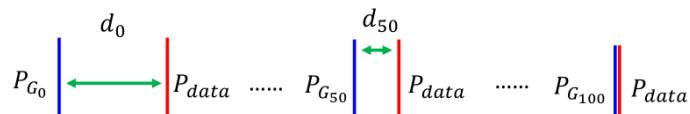
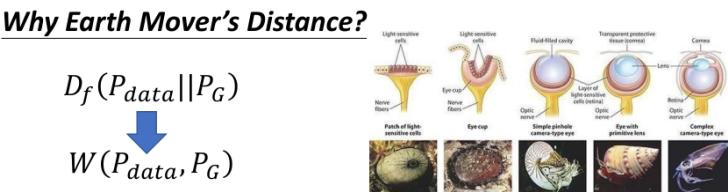
Wassertein Distance 或 Earth mover's distance 就是穷举所有可能的 γ ，

看哪一个 γ 算出来的距离最小，这个最小的距离就是 Wassertein Distance。

Wassertein Distance，它是一个很神奇的 distance。今天一般的 distance 就是直接套一个公式运算出来你就得到结果，但 Wassertein Distance，你要算它的话你要解一个 optimization problem，很麻烦。

所以今天给你两个 distribution，要算 Wassertein Distance 是很麻烦的，因为你要解一个 optimization problem，才算得出 Wassertein Distance。

Why Earth Mover's Distance?



$$JS(P_{G_0}, P_{data}) = \log 2$$

$$JS(P_{G_{50}}, P_{data}) = \log 2$$

$$JS(P_{G_{100}}, P_{data}) = 0$$

$$W(P_{G_0}, P_{data}) = d_0$$

$$W(P_{G_{50}}, P_{data}) = d_{50}$$

$$W(P_{G_{100}}, P_{data}) = 0$$

用 Wassertein Distance 来衡量两个 distribution 的距离有什么样的好处？

假设你今天是用 JS divergence，这一个 G_0 跟 $data$ 的距离， G_{50} 跟 $data$ 之间的距离对 JS divergence 来说，根本就是一样的。

除非你今天可以把 G0 一步跳到 G100，然后让 G100 正好跟 Pdata 重叠，不然 machine 在 update 你的 generator 参数的时候，它根本没有办法从 G0 update 到 G50。因为在这个 case，JS divergence 其实是一样大。

那这个其实就让我想到一个演化上的例子，我们知道人眼是非常复杂的器官，有人就会想说，凭借着天择的力量，不断的突变，到底怎么可能让生物突然产生人眼呢？那也许天择的假说并不是正确的，但是实际上今天生物是怎么从完全没有眼睛，变到有眼睛呢？并不是一步就产生眼睛，而是通过不断微小的突变的累积，才产生眼睛这么复杂的器官。比如说一开始，生物只是在皮肤上面，产生一些感光的细胞，那通过突变，某些细胞具有感光的能力，也许是做得到的，接下来呢，感光细胞所在的那个皮肤，就凹陷下去，凹陷的好处是，光线从不同方向进来，就不同的感光细胞会受到刺激，那生物就可以判断光线进来的方向。接下来因为有凹洞的关系所就会容易堆灰尘，就在里面放了一些液体，然后免得灰尘跑进去，然后再用一个盖子把它盖起来，最后就变成眼睛这个器官。但是你要直接从皮肤就突然突变，变异产生出眼睛是不可能的，所以就像人，没有办法一下子就长出翅膀变成一个鸟人一样。天择只能做小小的变异，而每一个变异都必须是有好处的，那才能够把这些变异累积起来，最后才能够产生巨大的变异。所以从产生感光细胞，到皮肤凹陷下去，到产生体液把盖子盖起来等等，每一个小小步骤对生物的生存来说都是有利的。所以演化才会由左往右走，生物才会产生眼睛。那如果要产生翅膀可能就比较困难，因为假设你一开始产生很小的翅膀，没有办法飞的话，那就没有占到什么优势。

那对这个 generator 来说也是一样的，它如果说 G50 并没有比 G0 好，你就没有办法从 G0，变到 G50，然后慢慢累积变化变到 G100。

但是如果你用 Wasserstein Distance 就不一样了，因为对 Wasserstein Distance 来说，d50 是比 d0 还要小的，所以对 generator 来说，它就可以 update 参数，把 distribution 从这个地方挪到这个地方，直到最后你 generator 的 output 可以和 data 真正的重合。

WGAN

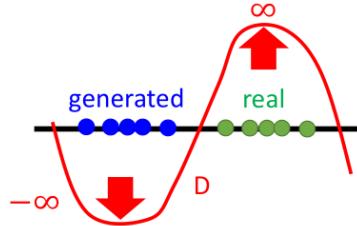
Evaluate wasserstein distance between P_{data} and P_G

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

D has to be smooth enough.

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

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



我们现在要量 P_G 和 P_{data} 之间的 Wasserstein Distance，我们要怎么去改 discriminator，让他可以衡量 P_G 和 P_{data} 的 Wasserstein Distance 呢？

这边就是直接告诉大家结果，这个推论的过程其实是非常复杂的，这个证明过程其实很复杂，所以我们就直接告诉大家结果，怎么样设计一个 discriminator，它 train 完以后 objective function 的值，就是 Wasserstein Distance。

x 是从 P_{data} 里面 sample 出来的，让它的 discriminator 的 output 越大越好，如果 x 是从 P_G 里面 sample 出来的，让它的 discriminator 的 output 越小越好。

你还要有一个 constrain，discriminator 必须要是一个 1-Lipschitz function

所谓的 1-Lipschitz function 意思是说这个 discriminator，他是很 smooth 的。

为什么这个 1-Lipschitz function 是必要的呢？

你可以说根据证明就是要这么做，算出来才是 Wasserstein Distance，但是你也可以非常的直观地了解这件事。

如果我们不考虑这个 constrain，我们只说要让这些绿色 data 带到 discriminator 里面分数越大越好，这些蓝色 data 带到 discriminator 里面分数越小越好。

那你 train 的时候 discriminator 就会知道说，这边的分数要让他一直拉高一直拉高，这边的分数要让他一直压低一直压低。如果你的这两堆 data 是没有 overlap 的，我们讲过 real data 跟 generated data 很有可能是没有 overlap 的。如果这两堆 data 是没有 overlap 的，今天如果只是 discriminator 一味的要让这些 data 值越来越高，这边 data 值越来越小，它就崩溃了，因为这个 training 永远不会收敛，这个值可以越来越大直到无限大，这个值可以越来越小直到无限小，你的 training 永远不会停止。

所以你必须要有一个额外的限制，你今天的 discriminator，必须要是够平滑的，这样就可以强迫你在 learn 这个 discriminator 的时候，不会 learn 到说这边一直上升，这边一直下降永远不会停下来，那最终还是会停下来的。

Weight Clipping [Martin Arjovsky, et al., arXiv, 2017]
 WGAN
 Force the parameters w between c and -c
 After parameter update, if $w > c$, $w = c$;
 if $w < -c$, $w = -c$

Evaluate wasserstein distance between P_{data} and P_G

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

D has to be smooth enough. How to fulfill this constraint?

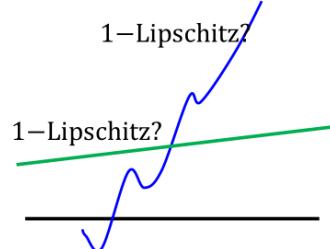
Lipschitz Function

$$\|f(x_1) - f(x_2)\| \leq K \|x_1 - x_2\|$$

Output change Input change

K=1 for "1 - Lipschitz"

Do not change fast



所以这个 Lipschitz function 它的意思到底是什么？

他的意思是说，当你 input 有一个变化的时候，output 的变化不能太大，能够让 input 的差距乘上 K 倍，大于等于 output 的差距。也就是说你 output 的差距不能够太大，不能够比 input 的差距大很多。

当你把 K 设为 1 的时候是 1-Lipschitz function，意味着说，你 output 的变化总是比 input 的变化要小的。

那像蓝色的 function 它变化这么剧烈，它变化这么剧烈，所以那就不是 1-Lipschitz function。那像绿色这个 function，他很平滑，它的变化很小，它在每一个地方，output 的变化都小于 input 的变化，那它就是一个 1-Lipschitz function。

怎么解这个 optimization problem? 如果我们把这个给 discriminator 的 constrain 拿掉，你就用 gradient ascent 去 maximize 它就好了。用 gradient ascent 你就可以 maximize 大括号里面的这个式子。

但现在问题是你的 discriminator 是有 constrain 的，我们一般在做 gradient decent 的时候，我们并不会给我们的参数 constrain，你会发现说如果你要给参数 constrain 的话，在 learning 的时候，还蛮困难的，你会不太清楚应该要怎么做。

所以你今天要给 discriminator constrain 是蛮困难，但实际上到底是怎么做的呢？

在最原始的 WGAN 里面，他的作法就是 weight clipping。

我们用 gradient ascent 去 train 你的 model，去 train 你的 discriminator，但是 train 完之后，如果你发现你的 weight，大过某一个你事先设好的常数 c，就把它设为 c，如果小于 -c 就把它设为 -c，结束。

那他希望说通过这个 weight clipping 的技术，可以让你 learn 出来的 discriminator，它是比较平滑的，因为你限制着它 weight 的大小，所以可以让这个 discriminator 它在 output 的时候，没有办法产生很剧烈的变化，这个 discriminator 可以是比较平滑的。

加了这个限制就可以让他变成 1-Lipschitz function 吗？答案就是不行，因为一开始也不知道要怎么解这个问题，所以就胡乱想一招，能动再说，那我觉得有时候做研究就是这样子嘛，不需要一次解决所有的问题。

在 WGAN 的第一篇原始 paper 里面，他就 propose 说如果 D 是 1-Lipschitz function，那我们就可以量 Wasserstein Distance，但他不知道要怎么真的 optimize 这个 problem，没关系先胡乱提一个挡着先，先 propose，先把 paper publish 出去，再慢慢想这样。

这个是 WGAN 最原始的版本，用的是 weight clipping。那当然它的 performance 不见得是最好的，因为你用这个方法他并没有真的让 D 限制在 1-Lipschitz function，它就只是希望通过这个限制，可以让你的 D 是比较 smooth 的。

Improved WGAN (WGAN-GP)

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

A differentiable function is 1-Lipschitz if and only if it has gradients with norm less than or equal to 1 everywhere.

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

$$V(G, D) \approx \max_D \{E_{x \sim P_{\text{data}}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

$$- \lambda \int_x \max(0, \|\nabla_x D(x)\| - 1) dx$$

Prefer $\|\nabla_x D(x)\| \leq 1$ for all x

\downarrow

$$- \lambda E_{x \sim P_{\text{penalty}}} [\max(0, \|\nabla_x D(x)\| - 1)]$$

Prefer $\|\nabla_x D(x)\| \leq 1$ for x sampling from $x \sim P_{\text{penalty}}$

后来就有一个新的招数，不是用 weight clipping，它是用 gradient 的 penalty，那这个技术叫做 improved WGAN 或者是又叫做 WGAN GP。

那 WGAN GP 这边想要讲的是什么呢？一个 discriminator 它是 1-Lipschitz function 等价于，如果你对所有可能的 input x ，都拿去对 discriminator 求他的 gradient 的话，这 gradient 的 norm 总是会小于等于 1 的，这两件事情是等价的。

你不知道怎么限制你的 discriminator，是 1-Lipschitz function，你能不能限制你的 discriminator 对所有的 input x ，去算他的 gradient 的时候，它的 norm，都要小于等于 1 呢？这件事显然是有办法 approximate 的。

要怎么 approximate 呢？这个 approximate 方法就是说在原来的这项后面，再加一个 penalize 的项，这一项的作用有点像是 regularization，这一项的作用是说，它对所有的 x 做积分，然后取一个 max，也就是说如果这个 gradient norm 小于 1 的话，那就没有 penalty，如果 gradient norm > 1，这一项就会有值，就会有 penalty。

所以今天在 train 这个 discriminator 的时候，今天在 training 的时候会尽量希望这个 discriminator 它的 gradient norm，小于等于 1。

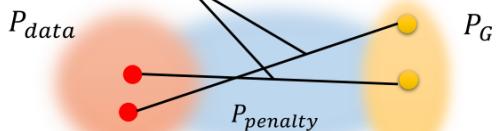
但实际上这么做会有一个问题，因为你不可能对所有的 x 都做积分。我们说一个 function 是 Lipschitz function，它的 if and only if 的条件是对所有的 x 这件事情都要满足。但是你无法真的去 check 说，不管你是在 train 还是在 check 的时候，你都无法做到说 sample 所有的 x ，让他们通通满足这个条件。

x 代表是所有可能的 image，那个 space 这么大，你根本无法 sample 所有的 x ，保证这件事情成立。所以怎么办？

这边做的另外一个 approximation 是说，假设事先定好的 distribution 叫做 P_{penalty} 。这个 x 是从 P_{penalty} 那个 distribution sample 出来的，我们只保证说在 P_{penalty} 那个 distribution 里面的 x ，它的 gradient norm 小于等于 1。

$$V(G, D) \approx \max_D \{E_{x \sim P_{\text{data}}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

$$- \lambda E_{x \sim P_{\text{penalty}}} [\max(0, \|\nabla_x D(x)\| - 1)]$$



"Given that enforcing the Lipschitz constraint everywhere is intractable, enforcing it **only along these straight lines** seems sufficient and experimentally results in good performance."

Only give gradient constraint to the region between P_{data} and P_G because they influence how P_G moves to P_{data}

这个 P_{penalty} 长什么样子呢？

在 WGAN GP 里面，从 P_{data} 里面 sample 一个点出来，从 P_G 里面 sample 一个点出来，把这两个点相连，然后在这两个点所连成的直线间，做一个 random 的 sample，sample 出来的 x 就当作是从 P_{penalty} sample 出来的。

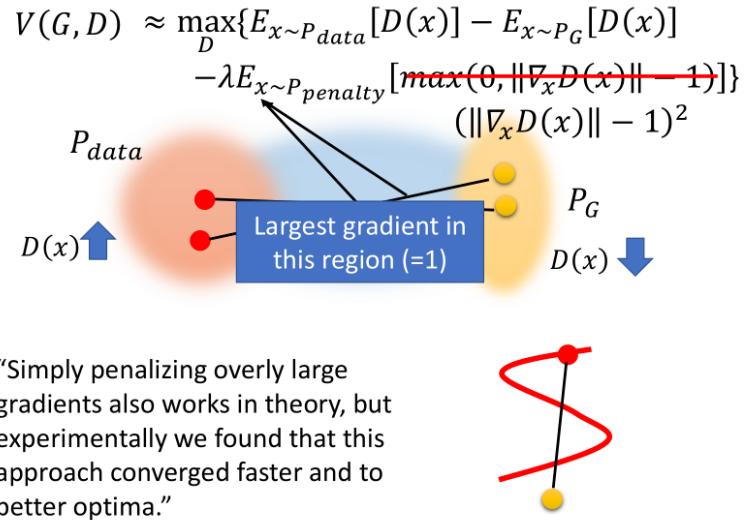
这个红色的点可以是 P_{data} 里面 sample 出来的任何点，这个黄色的点可以是 P_G 里面 sample 出来的任何点，从这两个点连起来，从这个连线中间去 sample，就是 P_{penalty} 。

所以 P_{penalty} 的分布大概就是在 P_G 和 P_{data} 中间，就是蓝色的这一个范围。

为什么会是这样子呢？为什么我们本来应该对，整个 space 整个 image 的 space 所有的 x 通通去给它 penalty，但为什么只在蓝色的部分给 penalty 是可以的呢？

在原始的 improved WGAN paper 它是这样写的，给每个地方都给它 gradient penalty 是不可能的，就是说实验做起来，这样就是好的这样子。实验做起来，这样看起来是 ok 的。

但是你从直觉上也可以了解说这么做是 make sense 的，因为我们今天在 train GAN 的时候，我们不是要 update 那个 generator，然后让 generator 顺着 discriminator 给我们的 gradient 的方向，挪到 P data 的位置去吗。也就是说，我们要让 generator 的这些点慢慢往左移，往左移，在这个例子里面 generator 的点，要慢慢往左移，挪到 P data 的位置去。那所以 generator 在挪动它的位置的时候，在 update 参数的时候，它看的就是 discriminator 的 gradient，所以应该只有在 generator output 的 distribution，跟 real data 的 distribution，中间的连线这个区域，才会真的影响你最后的结果。因为今天这个 PG 是看着这个地方的 gradient，这个地方的斜率，去 update 它的参数的，所以只有 PG 和 P data 之间的区域你需要去考虑你的 discriminator 的 shape 长什么样子，其他这些地方，反正你的 generator 也走不到，那你就不需要去考虑 discriminator 的 shape 长什么样子。所以我觉得在 PG 和 Pdata 中间做 sample 也是有道理的，也算是 make sense 的。



接下来要再做另外一个 approximation。

本来我们是希望这个 gradient norm 如果大过 1 给它 penalty，小于 1 不用 penalty。但实际上在 WGAN 的 implementation 里面，我们实际上 training 的时候，我们是希望 gradient 越接近 1 越好，本来理论上我们只需要 gradient < 1，大过 1 给他惩罚，小于 1 没有关系，但实作的时候说，gradient norm 必须离 1 越接近越好。gradient norm > 1 有惩罚，< 1 也有惩罚。为什么会这样呢？在 paper 里面说，实验上这么做的 performance 是比较好的。

当然这个 improved WGAN 也不会是最终的 solution，实际上你很直觉的会觉得，它是有一些问题的。举例来说我这边举一个例子，假设红色的曲线是你的 data，你在 data 上 sample 一个点是红色的，你在黄色的是你的 distribution，这边 sample 一个点，你说把他们两个连起来，然后给这边的这些线 constrain，你不觉得其实是不 make sense 的嘛。

因为如果我们今天照理说，我们只考虑黄色的点，要如何挪到红色的点，所以照理说，我们应该在红色的这个地方，sample 一个点跟黄色是最近的，然后只 penalize 这个地方跟黄色的点之间的 gradient，这个才 make sense 嘛，因为到时候黄色的点，其实它要挪动的话，它也是走到最近的地方，它不会跨过这些已经有红色点的地方跑到这里来。这个是有点奇怪的，我认为他会走这个方向（最近的点），而不是走这样的方向（连线）。所以你 gradient penalty penalize 在（连线）这个地方，是有点奇怪的。

那其实 improved WGAN 后面还有很多其他的变形，大家可以自己找一下

其实像今年的 ICLR 2018，就有一个 improved WGAN 的变形，叫做 improved 的 improved WGAN 这样子，那 improved 的 improved WGAN 他一个很重要的不同是说，它的 gradient penalty 不是只放在 Pdata 跟 PG 之间，他觉得要放在这个红色的区块。

Spectrum Norm

Spectrum Norm

Spectral Normalization → Keep gradient norm smaller than 1 everywhere [Miyato, et al., ICLR, 2018]

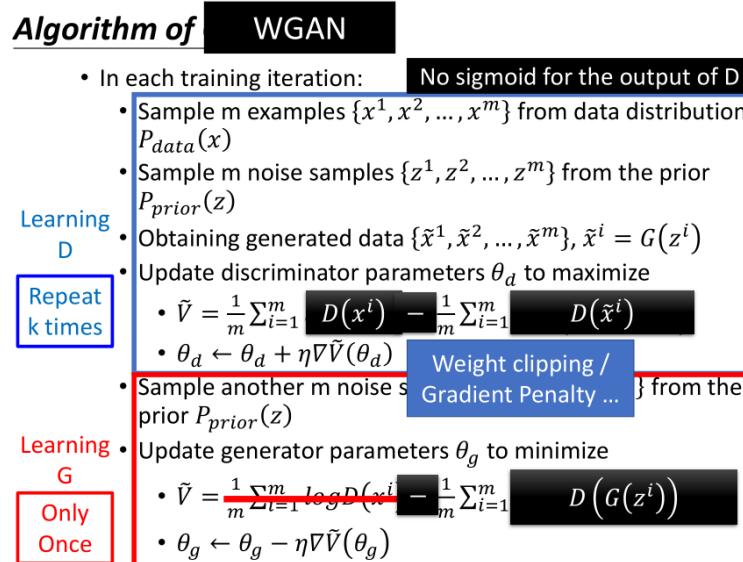


刚才 WGAN 什么都是一堆 approximation 嘛，spectrum norm 是这样，他 propose 了一个方法，这个方法真的可以限制你的 discriminator 在每一个位置的 gradient norm 都是小于 1 的，本来 WGAN GP 它只是 penalize 某一个区域的 gradient norm < 1，但是 spectrum norm 这个方法可以让你的 discriminator learn 完以后，它在每一个位置的 gradient norm 都是小于 1。

这个也是 ICLR 2018 的 paper，那细节我们就不提。

Algorithm of WGAN

我们看一下怎么从 GAN 改成 WGAN



那这边要注意的地方是，在原来的 GAN 里面你的 discriminator 有 sigmoid，有那个 sigmoid 你算出来才会是 JS divergence。

但是在 WGAN 里面，你要把 sigmoid 拆掉，让它的 output 是 linear 的，算出来才会是 Wasserstein Distance。

接下来你在 update 你的 discriminator，在 train 你的 discriminator 的时候呢，要注意一下就是你要加上 weight clipping，或者是加上 gradient penalty，不然这个 training 可能是不会收敛的。

所以你总共只要改 4 个地方，改 objective function、把 sigmoid 拆掉、把 weight clipping 加进去、改一下 generator update 的 objective function，就结束了。

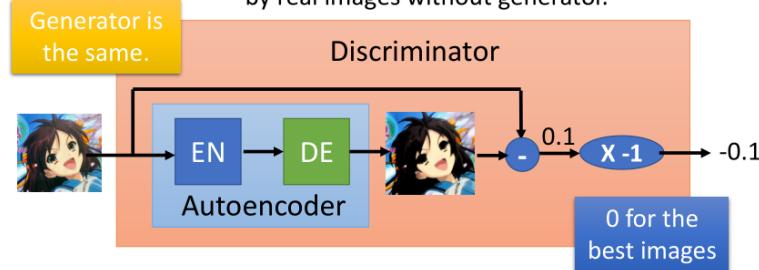
Energy-based GAN (EBGAN)

EBGAN 还有另外一个变形叫做BEGAN，另外一个变形我们不讲。

EBGAN 是什么，EBGAN 他唯一跟一般的 GAN 不同的地方是，它改了 discriminator 的 network 架构。

- Using an autoencoder as discriminator D

- Using the negative reconstruction error of auto-encoder to determine the goodness
- Benefit:** The auto-encoder can be pre-trained by real images without generator.



本来 discriminator 是一个 binary 的 classifier，它现在把它改成 auto encoder。

所以 Energy based GAN 的意思就是说，你的 discriminator 是这样，input 一张 image，有一个 encoder，把它变成 code，然后有一个 decoder 把它解回来，接下来你算那个 auto encoder 的 reconstruction error，把 reconstruction error 乘一个负号，就变成你的 discriminator 的 output。

也就是说这个 energy based GAN 它的假设就是，假设某一张 image 它可以被 reconstruction 的越好，它的 reconstruction error 越低，代表它是一个 high quality 的 image，如果它很难被 reconstruct，它的 reconstruction error 很大，代表它是一个 low quality 的 image。

那这种 EBGAN 他到底有什么样的好处呢？

我觉得他最大的好处就是，你可以 pre-train 你的 discriminator。

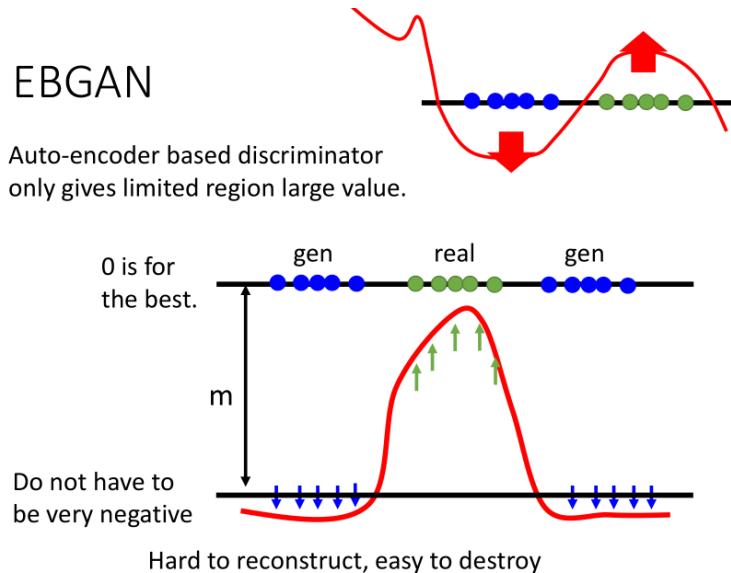
auto encoder 在 train 的时候，不需要 negative example，你在 train 你的 discriminator 的时候，它是一个 binary classifier，你需要 negative example，这个东西无法 pre trained。你没有办法只拿 positive example 去 train 一个 binary classifier。

所以这会造成的问题是一开始你的 generator 很弱，所以它 sample 出来的 negative example 也很弱，用很弱的 negative example 你 learn 出来就是一个很弱的 discriminator，那 discriminator 必须要等 generator 慢慢变强以后，你要 train 很久，才会让 discriminator 变得比较厉害。

但是 energy base GAN 就不一样，discriminator 是一个 auto encoder，auto encoder 是可以 pre trained，auto encoder 不需要 negative example，你只要给它 positive example，让它去 minimize reconstruction error 就好了。

所以你真的要用 energy based GAN 的时候，你要先 pre-train 好你的 discriminator，先拿你手上的那些 real 的 image，去把你的 auto encoder 先 train 好，所以你一开始的 discriminator，会很强，所以因为你的 discriminator 一开始就很强，所以你的 generator 一开始就可以 generate 很好的 image。

所以如果你今天是用 energy base GAN，你会发现说你前面几个 epoch，你就还可以还蛮清楚的 image。那这个就是 energy base GAN 一个厉害的地方。



那 energy based GAN 实际上在 train 的时候，还有一个细节你是要注意的，就是今天在 train energy based GAN 的时候，你要让 real example 它的 reconstruction error 越小越好。

但是要注意，你并不是要让 generated example 的 reconstruction error 越大越好，为什么？

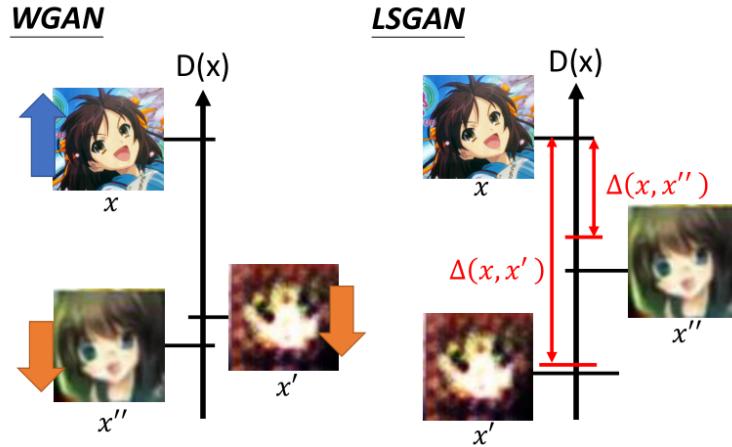
因为建设是比较难的，破坏是比较容易的。reconstruction error 要让它变小很难，因为，你必须要 input 一张 image 把它变成 code，再 output 同样一张 image，这件事很难，但是如果你要让 input 跟 output 非常不像，这件事太简单了，input 一张 image，你要让它 reconstruction error 很大，不就 output 一个 noise 就很大了吗？

所以如果你今天太专注于说要 maximize 这些 generated image 的 reconstruction error，那你的 discriminator，到时候就学到说看到什么 image 都 output 那个 noise，都 output noise，故意把它压低，这个时候你的 discriminator 的 loss 可以把它变得很小，但这个不是我们要的。

所以实际上在做的时候，你会设一个 margin 说，今天 generator 的 reconstruction loss 只要小于某一个 threshold 就好，当然 threshold 这个 margin 是你要手调的。

这个 margin 意思是说 generator loss 只要小于 margin 就好，不用再小，小于 margin 就好，不用让它再更小。

Loss-sensitive GAN (LSGAN)



其实还有另外一个东西也是有用到 margin 的概念，叫做Loss-Sensitive GAN。它也是LSGAN 这样，我们有一个 least square GAN，这边还有一个 Loss-Sensitive GAN。

那 Loss-Sensitive GAN 它也有用到 margin 的概念。我们之前在做 WGAN 的时候是说，如果是 positive example，就让他的值越大越好，negative example，就让他的值越小越好。

但是假设你有些 image 其实已经很 realistic，你让它的值越小越好，其实也不 make sense 对不对，所以今天在 LSGAN 里面它的概念就是，他加了一个叫做 margin 的东西。

就是你需要先有一个方法，去 evaluate 说你现在产生出来的 image 有多好，可能是把你产生出来的 image 呢，如果今天这个 x double prime 跟 x 已经很像了，那它们的 margin 就小一点，如果 x prime 跟 x 很不像，它们 margin 就大一点，所以你会希望 x prime 的分数被压得很低， x double prime 的分数只要压低过 margin 就好，不需要压得太低。

Feature Extraction by GAN

讲一下用 GAN 做 Feature Extraction 有关的事情，我想先跟大家讲的是 InfoGAN。

我们知道 GAN 会 random input 一个 vector，然后 output 一个你要的 object。我们通常期待 input 的那个 vector 它的每一个 dimension 代表了某种 specific 的 characteristic，你改了 input 的某个 dimension，output 就会有一个对应的变化，然后你可以知道每一个 dimension 它做的事情是什么。

但是实际上未必有那么容易，如果真的 train 了一个 GAN 你会发现，input 的 dimension 跟 output 的关系，观察不到什么关系。

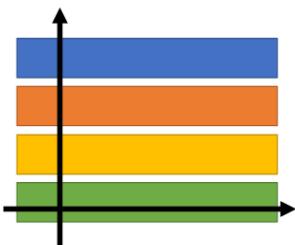
InfoGAN

Regular GAN
(The colors represents the characteristics.)

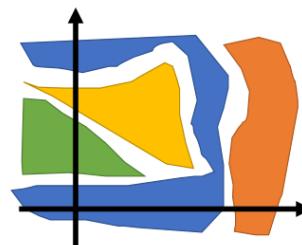
7	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	7	7	7	7	7
9	9	9	9	9	9	9	9	9	9
8	8	8	8	8	8	8	8	8	8

Modifying a specific dimension,
no clear meaning

What we expect



Actually ...



这边这是一个文献上的例子，假设 train 了一个 GAN，这个 GAN 做的事情，是手写数字的生成，你会发现你改了 input 的某一个维度，对 output 来说，横轴代表改变了 input 的某一个维度，output 的变化是看不太出规律的。比如说这边的 7，突然中间写了一横也不知道是什么意思，搞不清楚说，改变了某一维度到底对 output 的结果，起了什么样的作用。

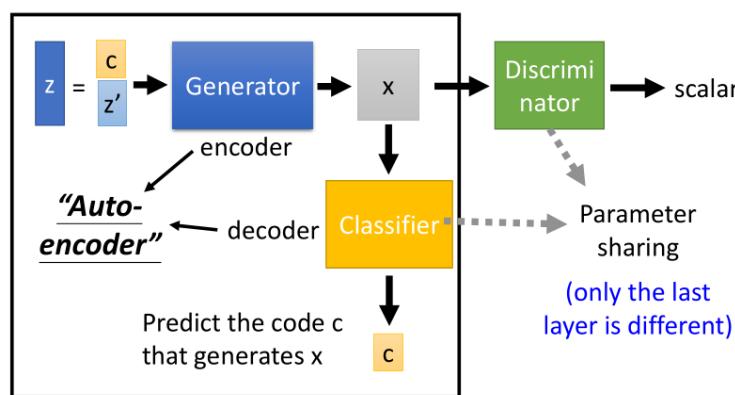
为什么会这样呢，现在这个投影片上这个二维平面，代表 generator input 的 random vector 的 space，假设 input 的 vector 只有两维，我们通常期待在这个 latent 的 space 上面，不同的 characteristic 的 object 它的分布是有某种规律性的，我们这边用不同的颜色来代表，假设你在这个区块，你使用这个区块的 vector 当作 generator 的 input，它 output 会有蓝色的特征，这个区块会有橙色的特征，这个区块会有黄色的特征，这个区块会有绿色的特征。本来的假设是这些不同的特征，他们在 latent space 上的分布是有某种规律性的，但是实际上也许它的分布是非常不规则的。

我们本来期待如果改变了 input vector 的某一个维度，它就会从绿色变到黄色再变到橙色再变到蓝色，它有一个固定的变化，但是实际上也许它的分布长的这个样子，也许 latent space 跟你要生成的那个 object 之间的关系，是非常复杂的。所以当你改变某一个维度的时候，你从蓝色变到绿色再变到黄色又再变回蓝色，你就觉得说不知道在干嘛。

InfoGAN

所以 InfoGAN 就是想要解决这个问题。

在 InfoGAN 里面你会把 input 的 vector 分成两个部分，比如说假设 input vector 是二十维，就说前十维把它叫作 c ，后十维我们把它叫作 z' 。



在 InfoGAN 里面你会 train 一个 classifier，这个 classifier 工作是、看 generator 的 output，然后决定根据这个 generator 这个 output 去预测现在 generator input 的 c 是什么。

所以这个 generator 吃这个 vector，产生了 x ，classifier 要能够从 x 里面反推原来 generator 输入的 c 是什么样的东西。

在这个 InfoGAN 里面，你可以把 classifier 视为一个 decoder，这个 generator 视为一个 encoder。这个 generator 跟 classifier 合起来，可以把它看作是一个 Autoencoder。它跟传统的 Autoencoder 做的事情是正好相反的，所以这边加一个双引号，因为我们知道传统的 Autoencoder 做的事情是给一张图片，他把它变成一个 code，再把 code 解回原来的图片，但是在 InfoGAN 里面这个 generator 和 classifier 所组成的 Autoencoder 做的事情，跟我们所熟悉的 Autoencoder 做的事情，是正好相反的。

在 InfoGAN 里面，generator 是一个 code 产生一张 image，然后 classifier 要根据这个 image 决定那个原来的 code 是什么样的东西。

当然如果只有 train generator 跟 classifier 是不够的，这个 discriminator 一定要存在，为什么 discriminator 一定要存在，假设没有 discriminator 的话，对 generator 来说，因为 generator 想要帮助 classifier，让 classifier 能够成功的预测， x 是从什么样的 c 弄出来的，如果没有 discriminator 的话，对 generator 来说，最容易让 classifier 猜出 c 的方式就是直接把 c 贴在这个图片上，然后 classifier 只要知道他去读这个图片中的数值，就知道 c 是什么，那这样就完全没有意义，所以这边一定要有一个 discriminator。discriminator 会检查这张 image 看起来像不像一个 real image。如果 generator 为了要让 classifier 猜出 c 是什么，而刻意地把 c 原本的数值，我们期待是 generator 根据 c 所代表的信息，去产生对应的 x ，但 generator 它可能就直接把 c 原封不动贴到这个图片上，但是如果只是把 c 原封不动贴到这个图片上，discriminator 就会发现这件事情不对，发现这看起来不像是真的图片，所以 generator 并不能够直接把 c 放在图片里面，透露给 classifier。

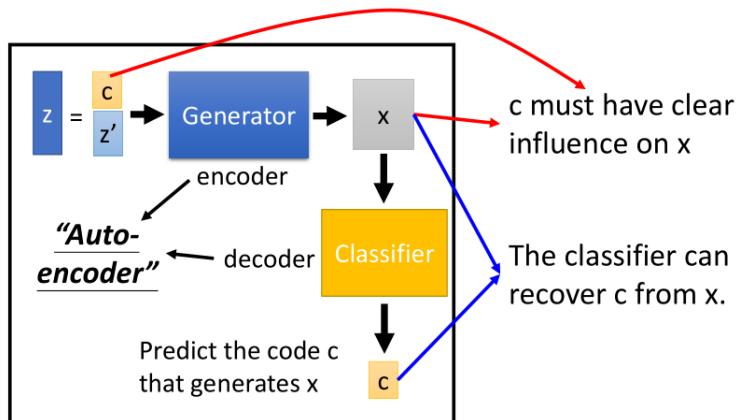
InfoGAN 在实作上 discriminator 跟 classifier 往往会 share 参数，因为他们都是吃同样的 image 当作 input，不过他们 output 的地方不太一样，一个是 output scalar，一个是 output 一个 code、vector，不过通常你可以让他们的一些参数是 share 的。

加上这个 classifier 会有什么好处，我们说我们刚才想要解的问题就是，input feature 它对 output 的影响不明确这件事，InfoGAN 怎么解决 input feature 对 output 影响不明确这件事呢？

InfoGAN 的想法是这个样子：为了要让 classifier 可以成功地从 image x 里面知道原来的 input c 是什么，generator 要做的事情就是，他必须要让 c 的每一个维度，对 output 的 x 都有一个明确的影响，如果 generator 可以学到 c 的每一个维度对 output 的 x 都有一个非常明确的影响，那 classifier 就可以轻易地根据 output 的 image 反推出原来的 c 是什么。如果 generator 没有学到让 c 对 output 有明确影响，就像刚看到那个例子，改了某一个 dimension 对 output 影响是很奇怪的，classifier 就会无法从 x 反推原来的 c 是什么。

在原来的 InfoGAN 里面他把 input z 分成两块，一块是 c 一块是 z' ，这个 c 他代表了某些特征，也就是 c 的每一个维度代表图片某些特征，他对图片是会有非常明确影响，如果你是做手写数字生成，那 c 的某一个维度可能就代表了那个数字笔画有多粗，那另外一个维度可能代表写的数字的角度是什么。

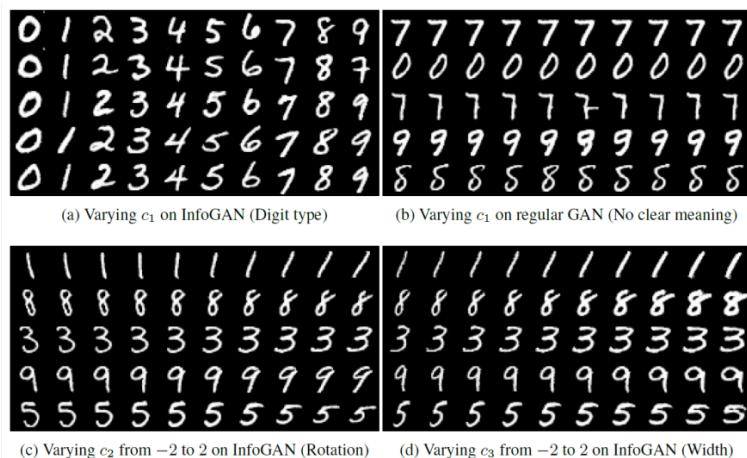
其实在 generator input 里面还有一个 z' ，在原始的 InfoGAN 里面他还加一个 z' ， z' 代表的是纯粹随机的东西，代表的是那些无法解释的东西。



那有人可能会问这个 c 跟 z' 到底是怎么分的，我们怎么知道前十维这个 feature 是应该对 output 有影响的，后十维这个 feature 他是属于 z' ，对 output 的影响是随机的呢？

你不知道，但是这边的道理是这个 c 并不是因为它代表了某些特征，而被归类为 c ，而是因为他被归类为 c 所以他会代表某些特征。

并不是因为他代表某些特征所以我们把他设为 c ，而是因为他被设为 c 以后根据 InfoGAN 的 training，使得他必须具备某种特征，希望大家听得懂我的意思。



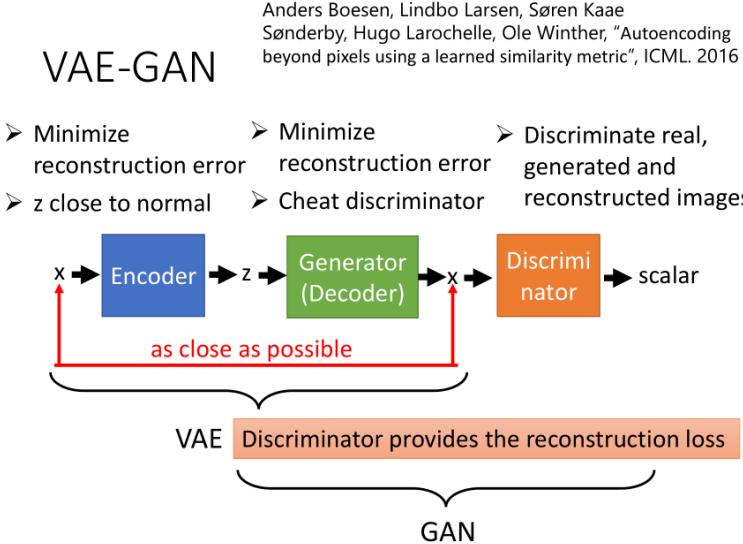
这是文献上的结果，第一张图是 learn 了 InfoGAN 以后，他改了 c 的第一维，然后发现什么事，发现 c 的第一维代表了 digit，这个很神奇，改了 c 的第一维以后，更动他的数值就从 0 跑到 9。这个 b 是原来的结果，他有做普通的 GAN，output 结果是很奇怪的。改第二维的话你产生的数字的角度就变了，改第三维的话你产生的数字就从笔划很细变到笔划很粗，这个就是 InfoGAN。

VAE-GAN

另外一个跟大家介绍的叫作 VAE-GAN，VAE-GAN 是什么，VAE-GAN 可以看作是用 GAN 来强化 VAE，也可以看作是用 VAE 来强化 GAN。

VAE 在 ML 有讲过的就是 Autoencoder 的变形，这个 Variational Autoencoder，Autoencoder 大家都很熟，就是一个 encoder，有一个 decoder，encoder input x output 是一个 z ，decoder 吃那个 z output 原来的 x ，你要让 input 跟 output 越近越好。

这个是 Variational Autoencoder，如果是 Variational Autoencoder 你还会给 z 一个 constraint，希望 z 的分布像是一个 Normal Distribution，只是在这边图上没有把它画出来。



那 VAE-GAN 的意思是在原来的 encoder decoder 之外 再加一个 discriminator。这个 discriminator 工作就是 check 这个 decoder 的 output x 看起来像不像是真的。

如果看前面的 encoder 跟 decoder 他们合起来是一个 Autoencoder，如果看后面的这个 decoder 跟 discriminator，在这边 decoder 他扮演的角色其实是 generator，我们看这个 generator 跟 discriminator 他们合起来是一个 GAN。

在 train VAE-GAN 的时候，一方面 encoder decoder 要让这个 Reconstruction Error 越小越好，但是同时 decoder 也就是这个 generator 要做到另外一件事，他会希望他 output 的 image 越 realistic 越好。如果从 VAE 的角度来看，原来我们在 train VAE 的时候，是希望 input 跟 output 越接近越好，但是对 image 来说，如果单纯只是让 input 跟 output 越接近越好，VAE 的 output 不见得会变得 realistic，他通常产生的东西就是很模糊的，如果你实际做过 VAE 生成的话，因为根本不知道怎么算 input 跟 output 的 loss，如果 loss 是用 L1 L2 norm，那 machine 学到的东西就会很模糊，那怎么办，就加一个 discriminator，你就会迫使 Autoencoder 在生成 image 的时候，不只是只是 minimize Reconstruction Error，同时还要产生比较 realistic image，让 discriminator 觉得是 realistic，所以从 VAE 的角度来看，加上 discriminator 可以让他的 output 更加 realistic。

如果从 GAN 的角度来看，前面这边 generator 加 discriminator 合起来，是一个 GAN。然后在前面放一个 encoder，从 GAN 的角度来看，原来在 train GAN 的时候，你是随机 input 一个 vector，你希望那个 vector 最后可以变成一个 image，对 generator 来说他从来没有看过真正的 image 长什么样子，他要花很多力气，你需要花很多的时间去调参数，才能够让 generator 真的学会产生真正的 image，知道 image 长什么样子。但是如果加上 Autoencoder 的架构，在学的时候 generator 不是只要骗过 discriminator，他同时要 minimize Reconstruction Error，generator 在学的时候他不是只要骗过 discriminator，他还有一个目标，他知道真正的 image 长什么样子，他想要产生一张看起来像是 encoder input 的 image，他在学习的时候有一个目标不是只看 discriminator 的 feedback，不是只看 discriminator 传来那边的 gradient，所以 VAE-GAN 学起来会比较稳一点。

在 VAE-GAN 里面，encoder 要做的事情就是要 minimize Reconstruction Error，同时希望 z 它的分布接近 Normal Distribution，对 generator 来说他也是要 minimize Reconstruction Error，同时他想要骗过 discriminator，对 discriminator 来说他就要分辨一张 image 是真正的 image 还是生成出来的 image，跟一般的 discriminator 是一样的。

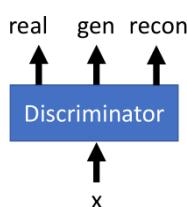
Algorithm

假如你对 VAE-GAN 有兴趣的话这边也是列一下 algorithm，这 algorithm 是这样，有三个东西，一个 encoder 一个 decoder 一个 discriminator，他们都是 network，所以先 initialize 他们的参数。

Algorithm

- Initialize En, De, Dis
- In each iteration:
 - Sample M images x^1, x^2, \dots, x^M from database
 - Generate M codes $\tilde{z}^1, \tilde{z}^2, \dots, \tilde{z}^M$ from encoder
 - $\tilde{z}^i = En(x^i)$
 - Generate M images $\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^M$ from decoder
 - $\tilde{x}^i = De(\tilde{z}^i)$
 - Sample M codes z^1, z^2, \dots, z^M from prior $P(z)$
 - Generate M images $\hat{x}^1, \hat{x}^2, \dots, \hat{x}^M$ from decoder
 - $\hat{x}^i = De(z^i)$
 - Update En to decrease $\|\tilde{x}^i - x^i\|$, decrease $KL(P(\tilde{z}^i | x^i) || P(z))$
 - Update De to decrease $\|\tilde{x}^i - x^i\|$, increase $Dis(\tilde{x}^i)$ and $Dis(\hat{x}^i)$
 - Update Dis to increase $Dis(x^i)$, decrease $Dis(\tilde{x}^i)$ and $Dis(\hat{x}^i)$

Another kind of discriminator:



这 algorithm 是这样说的，我们要先 sample M 个 real image，接下来再产生这 M 个 image 的 code，把这个 code 写作 z tilde，把 x 丢到 encoder 里面产生 z tilde，他们是真正的 image 的 code，接下来再用 decoder 去产生 image，你把真正的 image 的 code z tilde，把 z tilde 丢到 decoder 里面，decoder 就会产生 reconstructed image，就边写作 x tilde，x tilde 是 reconstructed image。

接下来 sample M 个 z，这个现在 z 不是从某一张 image 生成的，这边这个 z 是从一个 Normal Distribution sample 出来的。

用这些从 Normal Distribution sample 出来的 z，再丢到 encoder 里面再产生 image 这边叫做 x hat。

现在总共有三种 image，一种是真的从 database 里面 sample 出来的 image，一个是从 database sample 出来的 image 做 encode，变成 z tilde 以后再用 decoder 再还原出来，叫做 x tilde，还有一个是 generator 自己生成的 image，他不是看 database 里面任何一张 image 生成的，他是自己根据一个 Normal Distribution sample 所生成出来的 image，这边写成 x hat。

再来在 training 的时候，你先 train encoder，encoder 目标是什么，他要 minimize Autoencoder Reconstruction Error，所以要让真正的 image xi 跟 reconstruct 出来的 image x tilde 越接近越好，encoder 目的是什么，他希望原来 input 的 image 跟 reconstructed image，x 跟 x tilde 越接近越好，这第一件他要做的事，第二件他要做的事情是他希望这个 x 产生出来的 z tilde 跟 Normal Distribution 越接近越好，这是本来 VAE 要做的事情。

接下来 decoder 要做的事情是他同时要 minimize Reconstruction Error，他有另外一个工作是他希望他产生出来的东西，可以骗过 discriminator，他希望他产生出来的东西，discriminator 会给他高的分数。

现在 decoder 其实会产生两种东西，一种是 x tilde，是 reconstructed image，通常 reconstructed image 就是会看起来整个结构比较好，但是比较模糊，这个 x tilde 产生一个 reconstructed image，这个 reconstructed image 就到 discriminator 里面，分数要越大越好。那把你 x hat，就是 machine 自己生出来的 image，丢到 discriminator 里面，希望值越大越好。

最后轮到 discriminator，discriminator 要做的事情是如果是一个 real image 给他高分，如果是 faked image，faked image 有两种，一种是 reconstruct 出来的，一种是自己生成出来的，都要给他低分。这是 VAE-GAN 的作法。

我们之前看到 discriminator 都是一个 Binary Classifier，他就是要鉴别一张 image 是 real 还是 fake，其实还有另外一个做法是，discriminator 其实是一个三个 class 的 classifier，给他一张 image 他要鉴别他是 real 还是 generated 还是 reconstructed。因为 generated image 跟 reconstructed image 他们本质上看起来很不像的，在右边的 algorithm 里面，是把 generated 跟 reconstructed 视为是同一个 class，就是 fake 的 class，都当作 fake 的 image。

但是这个做法是把 generate 出来的 image 跟 reconstruct 出来的 image，视为两种不同的 image，discriminator 必须去学着鉴别这两种的差异。generator 在学的时候，有可能产生 generated image，他也有可能产生 reconstructed image，他都要试着让这两种 image discriminator 都误判，认为他是 real 的 image，这个是 VAE-GAN，VAE-GAN 是去修改了 Autoencoder。

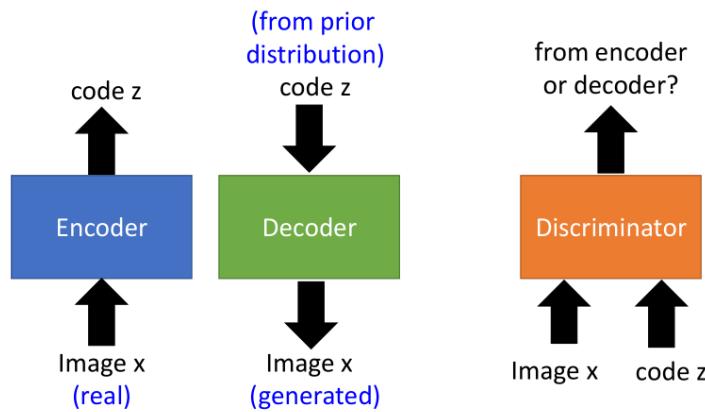
BiGAN

其实 BiGAN 还有另外一个技术，跟他非常地相近，其实不只是相近根本是一模一样，叫做 ALI，BiGAN 跟 ALI 如果没记错的话是同时发表在 ICLR 2017 上面，有什么差别？就是没有任何差别，不同的两群人居然想出了一模一样的方法，而且我发现 BiGAN 的 citation 比较高，我想原因就是因为他有 GAN，然后 ALI 他没有用到 GAN 这个字眼，citation 就少一点。

还有另外一个技术叫做 BiGAN，BiGAN 他也是修改了 Autoencoder。

我们知道在 Autoencoder 里面，有一个 encoder，有一个 decoder，在 Autoencoder 里面是把 encoder 的 output 丢给 decoder 做 reconstruction。

但是在 BiGAN 里面不是，在 BiGAN 里面就有一个 encoder 有一个 decoder，但是他们的 input output 不是接在一起的。



encoder 吃一张 image 他就变成一个 code, decoder 是从一个 Normal Distribution 里面 sample 一个 z 出来丢进去, 他就产生一张 image。

但是我们并不会把 encoder 的输出丢给 decoder, 并不会把 decoder 的输出丢给 encoder, 这两个是分开的。

有一个 encoder 有一个 decoder, 这两个是分开的那他们怎么学呢, 在 Autoencoder 里面可以学是因为收集了一大堆 image 要让 Autoencoder 的 input 等于 Autoencoder output, 现在 encoder 跟 decoder 各自都只有一边, encoder 只有 input 他不知道 output target 是什么, decoder 他只有 input, 他不知道 output 的 image 应该长什么样子, 怎么学这个 encoder 跟 decoder。

这边的做法是再加一个 discriminator, 这个 discriminator 他是吃 encoder 的 input 加 output, 他吃 decoder 的 input 加 output, 他同时吃一个 code z 跟一个 image x , 一起吃进去, 然后它要做的事情是鉴别 x 跟 z 的 pair 他们是从 encoder 来的还是从 decoder 来的, 所以它要鉴别一个 pair 他是从 encoder 来的还是从 decoder 来的。

我们先讲一下 BiGAN 的 algorithm 然后再告诉你为什么, BiGAN 这样做到底是有什么样的道理。

Algorithm

Initialize encoder En, decoder De, discriminator Dis

In each iteration:

- Sample M images x^1, x^2, \dots, x^M from database
- Generate M codes $\tilde{z}^1, \tilde{z}^2, \dots, \tilde{z}^M$ from encoder
 - $\tilde{z}^i = En(x^i)$
- Sample M codes z^1, z^2, \dots, z^M from prior $P(z)$
- Generate M codes $\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^M$ from decoder
 - $\tilde{x}^i = De(z^i)$
- Update Dis to increase $Dis(x^i, \tilde{z}^i)$, decrease $Dis(\tilde{x}^i, z^i)$
- Update En and De to decrease $Dis(x^i, \tilde{z}^i)$, increase $Dis(\tilde{x}^i, z^i)$

现在有一个 encoder 有一个 decoder 有一个 discriminator, 这个跟刚才讲 VAE-GAN 虽然一致, 不过这边 BiGAN 的运作方式跟 VAE-GAN 是非常不一样。

每一个 iteration 里面, 你会先从 database 里面 sample 出 M 张真的 image, 然后把这些真的 image 丢到 encoder 里面, encoder 会 output code 就得到了 M 组 code, 得到了 M 个 z tilde, 这个是用 encoder 生出来的东西。

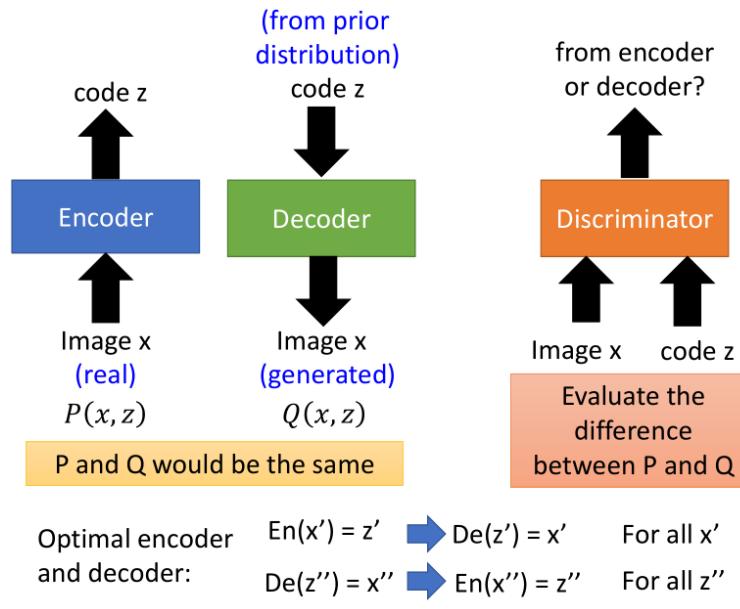
接下来用 decoder 生东西, sample M 个 code, 这个从一个 Normal Distribution sample 出来, 把这些 code 丢到 decoder 里面, decoder 就产生他自己生成的 image x tilde, 所以这边没有 tilde 的东西都是真的, 有 tilde 的东西都是生成的。

这边有 M 个 real image 生成出 M 个 code, 这边有 M 个 code 生成出 M 个 image。

接下来要 learn 一个 discriminator, discriminator 工作是给他 encoder 的 input 跟 output 给他高分, 给它 decoder 的 input 跟 output 给它低分, 如果这个 pair 是 encoder 的 input 跟 output, 给他高分, 如果这个 pair 是 decoder 的 input 跟 output, 就给他低分。

有人会问为什么是 encoder 会给高分, decoder 会给低分, 其实反过来讲你也会问同样的问题, 不管是你要让 encoder 高分 decoder 低分, 还是 encoder 低分 decoder 高分, 是一样的, 意思是完全一模一样的, learn 出来结果也会是一样的, 它并没有什么差别, 只是选其中一个方法来做就是了。

encoder 跟 decoder 要做的事情就是去骗过 discriminator。如果 discriminator 要让 encoder 的 input output 高分，decoder 的 input output 低分，encoder decoder 他们就要连手起来，让 encoder 的 input output 让 discriminator 给它低分，让 decoder 的 input output，discriminator 给他高分。所以 discriminator 要做什么事，encoder 跟 decoder 就要连手起来，去骗过 discriminator 就对了，到底要让 encoder 高分还是 decoder 高分，是无关紧要的。这个是 BiGAN 的 algorithm。



BiGAN 这么做到底是什么道理，我们知道 GAN 做的事情，这个 discriminator 做的事情就是在 evaluate 两组 sample 出来的 data，到底他们接不接近。

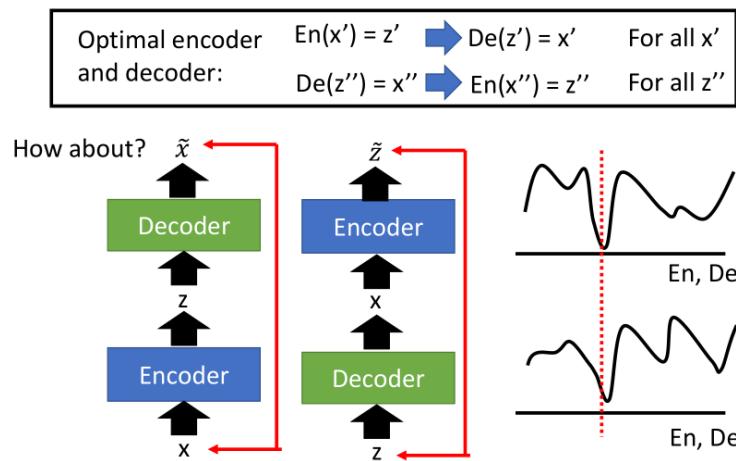
我们讲过从 real database 里面 sample 一堆 image 出来，用 generator sample 一堆 image 出来，一个 discriminator 做的事情其实就是在量这两堆 image 的某种 divergence 到底接不接近。

这个道理是一样的，可以把 encoder 的 input output 合起来，当作是一个 Joint Distribution，encoder input 跟 output 合起来有一个 Joint Distribution 写成 $P(x, z)$ ，decoder input 跟 output 合起来也是另外一个 Joint Distribution $Q(x, z)$ 。discriminator 要做的事情就是去衡量这两个 distribution 之间的差异，然后希望透过 discriminator 的引导让这两个 distribution 之间越近越好。

在原来的 GAN 里面，我们希望 generator 生成出来的 data distribution，跟 P data 越接近越好，这边的道理是完全一模一样的，discriminator 希望 encoder input output 所组成的 Joint Probability，跟 decoder input output 所组成的 Joint Probability，这两个 Data Distribution 越接近越好，所以 eventually 在理想的状况下，应该会学到 P 这个 distribution，也就是 encoder 的 input 跟 output 所组成的 distribution，跟 Q 这个 distribution，这两个 distribution，他们是一模一样。

如果最后他们 learn 到一模一样的时候，会发生什么事情？你可以轻易的知道如果 P 跟 Q 的 distribution，是一模一样的，你把一个 image x' 丢到 encoder 里面让它给你一个 $code z'$ ，再把 z' 丢到 decoder 里面让它给你一个 image x' 。 x' 会等于原来的 input x' ，你把 x' 丢进去它会产生 z' ，你把 z' 丢到 decoder 里面，它会产生原来的 x' 。你把 z'' 丢到 decoder 里面让它产生 x'' ，你就把 x'' 丢到 encoder 里面，那它就会产生 z'' 。

所以 encoder 的 input 产生一个 output，再把 output 丢到 decoder 里面会产生原来 encoder 的 input，decoder 给它一个 input 它产生一个 output，再把它的 output 丢到 encoder 里面，它会产生一模一样的 input，虽然说实际上在 training 的时候，encoder 跟 decoder 并没有接在一起，但是透过 discriminator 会让 encoder decoder 最终在理想上达成这个特性。



所以有人会问这样 encoder 跟 decoder 做的事情是不是就好像是 learn 了一个 Autoencoder，这个 Autoencoder input 一张 image 它变成一个 code，再把 code 用 decoder 解回原来一样的 image。再 learn 一个反向的 Autoencoder，所谓的反向的 Autoencoder 的意思是，decoder 吃一个 code 它产生一张 image，再从这个 image 还原回原来的 code。

假设在理想状况下，BiGAN 它可以 learn 到 optimal 的结果，确实会跟同时 learn 这样子一个 encoder 跟 Autoencoder 得到的结果是一样的。

那有人就会问为什么不 learn 这样子一个 encoder 跟一个 inverse Autoencoder 就好了呢，为什么还要引入 GAN，这样听起来感觉上是画蛇添足。

我觉得如果用 BiGAN learn 的话，得到的结果还是会不太一样，这边想要表达的意思是，learn 一个 BiGAN，跟 learn 一个下面这个 Autoencoder，他们的 optimal solution 是一样的，但它们的 Error Surface 是不一样的，如果这两个 model 都 train 到 optimal 的 case，得到的结果会是一样的，但是实际上不可能 train 到 optimal 的 case。BiGAN 无法真的 learn 到 P 跟 Q 的 distribution 一模一样，Autoencoder 无法 learn 到 input 跟 output 真的一模一样，这件事情是不可能发生的，所以不会真的收敛到 optimal solution。

但不是收敛到 optimal solution 的状况下，这两种方法 learn 出来的结果就会不一样，到底有什么不一样，这边没有把文献上的图片列出来，如果你看一下文献上的图片的话，一般的 Autoencoder learn 完以后，input 一张 image 它就是 reconstruct 另外一张 image，跟原来的 input 很像，然后比较模糊，这个大家应该都知道 Autoencoder 就是这么回事。

但是如果用 BiGAN 的话，其实也是 learn 出来了一个 Autoencoder，learn 了一个 encoder 一个 decoder，他们合起来就是一个 Autoencoder。

但是当你把一张 image 丢到这个 encoder，再从 decoder 输出出来的时候，其实你可能会得到的 output 跟 input 是非常不像的。它会比较清晰，但是非常不像，比如说你把一只鸟丢进去，它 output 还是会是一只鸟，但是是另外一只鸟。这个就是 BiGAN 的特性，你可以去看一下它的 paper，如果跟 Autoencoder 比起来，他们的最佳的 solution 是一样的，但是实际上 learn 出来的结果会发现这两种 Autoencoder，就是用这种 minimize Reconstruction Error 方法 learn 了一个 Autoencoder，还是用 BiGAN learn 的 Autoencoder，他们的特性其实是非常不一样。

BiGAN 的 Autoencoder 它比较能够抓到语意上的信息，就像刚才说的你 input 一只鸟，它知道是一只鸟，它 reconstruct 出来的结果，decoder output 也是一只鸟，但是不是同一只鸟，这就是一个还满神奇的结果。

Triple GAN

Triple GAN 里面有三个东西，一个 discriminator 一个 generator，一个 classifier。如果先不要管 classifier 的话，Triple GAN 本身就是三个 Conditional GAN。

Conditional GAN 就是 input 一个东西，output 一个东西，比如说 input 一个文字，然后就 output 一张图片，generator 就是吃一个 condition，这边 condition 写成 Y，然后产生一个 x，它把 x 跟 y 的 pair 丢到 discriminator 里面，discriminator 要分辨出 generator 产生的东西是 fake 的，real database sample 产生的东西就是 true，所以 generator 跟 discriminator 合起来就是一个 Conditional GAN。

这边再加一个 classifier 是什么意思，这边再加一个 classifier 意思是 Triple GAN 是一个 Semi-supervised Learning 的做法。

假设有少量的 labeled data，但是大量的 unlabeled data，也就是说你有少量的 X 跟 Y 的 pair，有大量的 X 跟 Y 他们是没有被 pair 在一起。

所以 Triple GAN 它主要的目标，是想要去学好一个 classifier，这 classifier 可以 input X，然后就 output Y，你可以用 labeled data 去训练 classifier，你可以从有 label 的 data 的 set 里面，去 sample X Y 的 pair，去 train classifier，但是同时也可根据 generator，generator 会吃一个 Y 产生一个 X，把 generator 产生的 X Y 的 pair，也丢给这个 classifier 去学。它的用意就是增加 training data，本来有 labeled 的 X Y 的 pair 很少，但是有一大堆的 X 跟 Y 是没有 pair 的，所以用 generator 去给他吃一些 Y 让它产生 X，得到更多 X Y 的 pair 去 train classifier。

这个 classifier 它吃 X，然后去产生 Y。

discriminator 会去鉴别这 classifier input 跟 output 之间的关系，看起来跟真正 X Y 的 pair 有没有像。

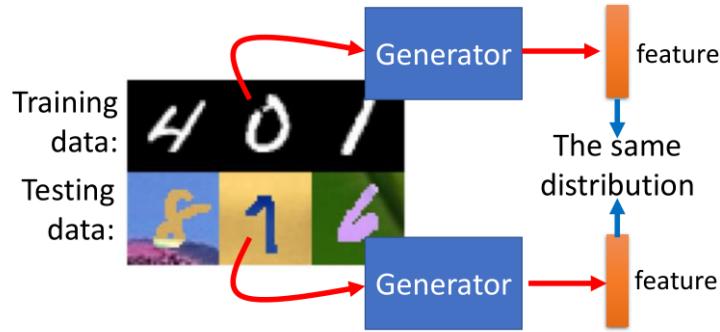
所以 Triple GAN 是一个 Semi-supervised Learning 的做法。

这边就不特别再仔细地说它，只是告诉大家有 Triple GAN 这个东西，有 BiGAN 就要有 Triple GAN。

Domain-adversarial training

在讲 Unsupervised Conditional Generation 的时候，我们用上了这个技术。这个技术在 ML 有讲过，所以这边就只是再复习一下。

- Training and testing data are in different domains



Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand,
Domain-Adversarial Training of Neural Networks, JMLR, 2016

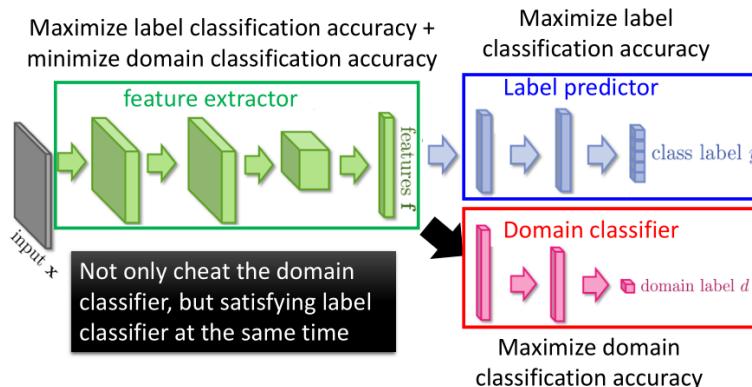
这个 Domain-adversarial training 就是要 learn 一个 generator，这个 generator 工作就是抽 feature。

假设要做影像的分类，这个 generator 工作就是吃一张图片 output 一个 feature。

在做 Machine Learning 的时候，很害怕遇到一个问题，是 training data 跟 testing data 不 match，假设 training data 是黑白的 MNIST，testing data 是彩色的图片，是彩色的 MNIST，你可能会以为你在这个 training data 上 train 起来，apply 到这个 testing data 上，搞不好也 work。因为 machine 搞不好可以学到反正 digit 就是跟颜色无关，考虑形状就好了，所以他在黑白图片上 learn 的东西也可以 apply 到彩色图片。

但是事实上事与愿违，machine 就是很笨，实际上 train 下去，train 在黑白图片上，apply 彩色图片上，虽然你觉得 machine 只要学到把彩色图片自己在某个 layer 转成黑白的，应该就可以得到正确结果，但是实际上不是，它很笨，它就是会答错，怎么办？

我们希望有一个好的 generator，这个 generator 做的事情是 training set 跟 testing set 的 data 不 match 没有关系，透过 generator 帮你抽出 feature。在 training set 跟 testing set 虽然他们不 match，他们的 domain 不一样，但是透过 generator 抽出来的 feature，他们有同样的 distribution，他们是 match 的，这个就是 Domain-adversarial training。



This is a big network, but different parts have different goals.

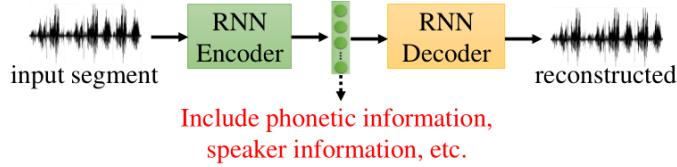
怎么做呢，这个图在 Machine Learning 有看过了，就 learn 一个 generator，其实就是 feature extractor，它吃一张 image 它会 output 一个 feature。有一个 Domain Classifier 其实就是 discriminator，这个 discriminator 是要判断现在这个 feature 来自于哪个 domain，假设有两个 domain，domain x 跟 domain y，你要 train 在 domain x 上面，apply 在 domain y 上面。然后这个时候 Domain Classifier 要做的事情是分辨这个 feature 来自于 domain x 还是 domain y，在这边同时你又有另外一个 classifier，这个 classifier 工作是根据这个 feature 判断，假设现在是数字的分类，要根据这个 feature 判断它属于哪个 class，它属于哪个数字，这三个东西是一起 learn 的，但是实际上在真正 implement 的时候不一定一起 learn。

在原始 Domain-adversarial training 的 paper 里面，它就是一起 learn 的，这三个 network 就是一起 learn，只是这个 Domain Classifier 它的 gradient 在 back propagation 的时候在进入 Feature Extractor 之前，会乘一个负号，但是实际上真的在 implement 的时候你不一定同时一起 train，你可以 iterative train，就像 GAN 一样。

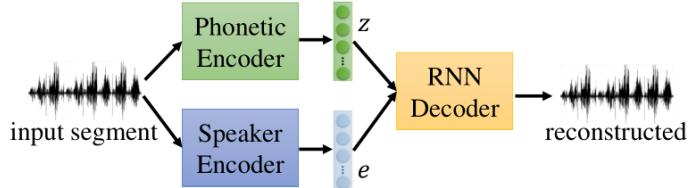
在 GAN 里面也不是同时 train generator 跟 discriminator，你是 iterative 的去 train，有人可能会问能不能够同时 train generator 跟 discriminator，其实是可以的。如果你去看 f-GAN 那篇 paper 的话，它其实就 propose 一个方法，它的 generator 跟 discriminator 是 simultaneously train 的，就跟原始的 Domain-adversarial training 的方法是一样的，有同学试过类似的做法，但发现同时 train 比较不稳定，如果是 iterative train 其实比较稳，如果先 train Domain Classifier，再 train Feature Extractor，先 train discriminator 再 train generator，iterative 的去 train，它的结果会是比较稳的，这个是 Domain-adversarial training。

Feature Disentangle

Original Seq2seq Auto-encoder



Feature Disentangle



用类似这样的技术可以做一件事情，这件事情叫做 Feature Disentangle，Feature Disentangle 是什么意思，用语音来做一下举例，在别的 domain 上比如说 image processing，或者是 video processing，这样的技术也是用得非常多。

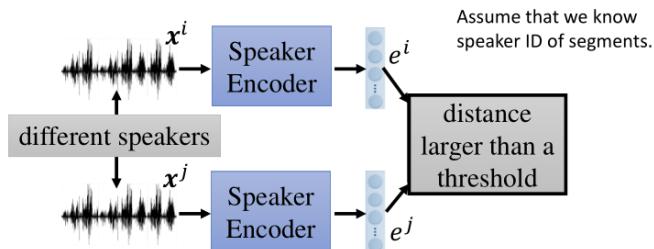
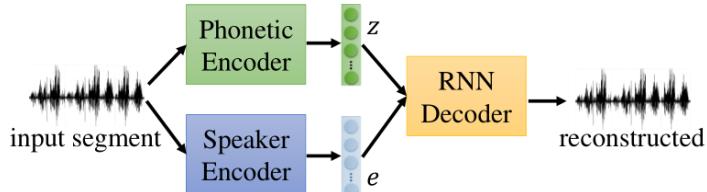
用语音来做例子，假设 learn 一个语音的 Autoencoder，learn 一个 sequence to sequence 的 Autoencoder，learn 一个 Autoencoder 它 input 是一段声音讯号，把这段声音讯号压成 code，再把这段 code 透过 decoder 解回原来的声音讯号，你希望 input 跟 output 越接近越好，就要 learn 这样一个 sequence to sequence Autoencoder，它中间你的 encoder 会抽出一个 latent representation，现在你的期待是 latent representation 可以代表发音的信息，但是你发现你实际 train 这样 sequence to sequence Autoencoder 的时候，你抽出来未必能让中间的 latent representation 代表发音的信息。

为什么？因为中间的 latent representation 它可能包含了很多各式各样不同的信息，因为 input 一段声音讯号，这段声音讯号里面不是只有发音的信息，它还有语者的信息，还有环境的信息，对 decoder 来说，这个 feature 里面一定必须要同时包含各种的信息，包含发音的信息，包含语者的信息，包含环境的信息，这个 decoder 根据所有的信息合起来，才可以还原出原来的声音。

我们希望做的事情是，知道在这个 vector 里面，到底哪些维度代表了发音的信息，那些维度代表了语者的信息或者是其他的信息，这边就需要用到一个叫做 Feature Disentangle 的技术，这种技术就有很多的用处。

因为你可以想象，假设你可以 learn 一个 encoder，它的 output 你知道那些维是跟发音有关的，那些维是跟语者有关的，你可以只把发音有关的部分，丢到语音识别系统里面去做语音识别，把有关语者的信息，丢到声纹比对的系统里面去，然后它就会知道现在是不是某个人说的。所以像这种 Feature Disentangle 技术有很多的应用。

Feature Disentangle



怎么做到 Feature Disentangle 这件事。

现在假设要 learn 两个 encoder，一个 encoder 它的 output 就是发音的信息，另外一个 encoder 它的 output 就是语者的信息，然后 decoder 吃发音的信息加语者的信息合起来，还原出原来的声音讯号。

接下来就可以把抽发音信息的 encoder 拔出来，把它的 output 去接语音识别系统，因为在做语音识别的时候，常会遇到的问题是两个不同的人说同一句话，它听起来不太一样，在声音讯号上不太一样，如果这个 encoder 可以把语者的 variation、语者所造成的差异 remove 掉。对语音识别系统来说辨识就会比较容易，对声纹比对也是一样，同一个人说不同的句子，他的声音讯号也是不一样，如果可以把这种发音的信息、content 的信息、跟文字有关的信息，把它滤掉，只抽出语者的特征的话，对后面声纹比对的系统也是非常有用。

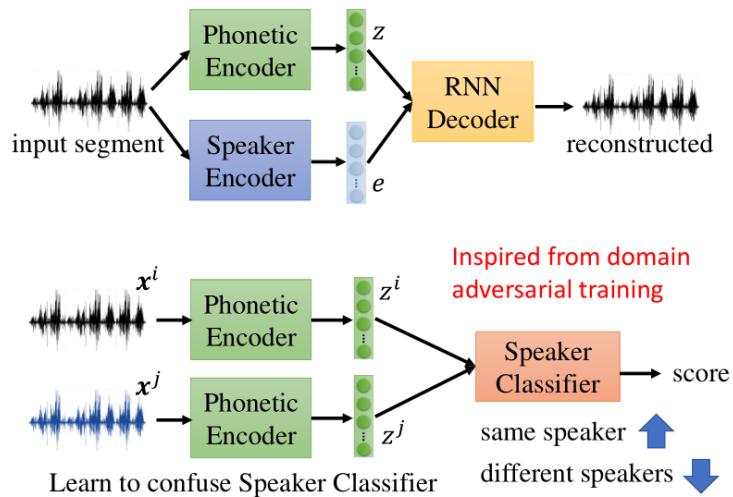
这件事怎么做，怎么让机器自动学到这个 encoder，如果这三个东西 joint learn，当然没有办法保证 Phonetic Encoder 的 output 一定要是发音的信息，Speaker Encoder 的 output 一定要是语者的信息。

于是就需要加一些额外的 constrain，比如说对语者的地方，你可能可以假设现在 input 一段声音讯号在训练的时候，我们知道那些声音讯号是同一个人说的。这个假设其实也还满容易达成的，因为可以假设同一句话就是同一个人说的，同一句话把它切成很多个小块，每一个小块就是同一个人说的。

所以对 Speaker Encoder 来说，给它同一个人说的声音讯号，虽然他们的声音讯号可能不太一样，但是 output 的 vector、output 的 embedding 要越接近越好。

同时假设 input 的两段声音讯号是不同人说的，那 output 的 embedding 就不可以太像，他们要有一些区别。

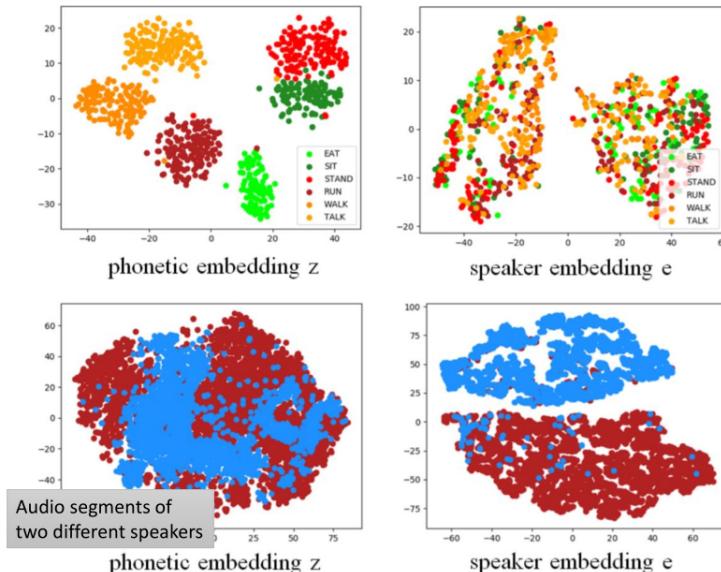
就算是这样做，你只能让 Speaker Encoder 的 output 考虑语者的信息，没有办法保证 Phonetic Encoder output 一定是发音的信息，因为也许语者的信息也会被藏在绿色的 vector 里，所以怎么办？



这边就可以用到 Domain Adversarial Training 的概念，再另外去 train 一个 Speaker Classifier。

Speaker Classifier 作用是给它两个 vector，它去判断这两个 vector 到底是同一个人说的还是不同的人说的，Phonetic Encoder 要做的事情就是去想办法骗过 Speaker Classifier，Speaker Classifier 要尽力去判断给他两个 vector 到底是同一个人说的还是不同人说的，Phonetic Encoder 要想尽办法去骗过 classifier，这个其实就是一个 GAN，后面就是 discriminator，前面就是 generator。

如果 Phonetic Encoder 可以骗过 Speaker Classifier，Speaker Classifier 完全无法从这些 vector 判断到底是不是同一个人说的，那就意味着 Phonetic Encoder 它可以滤掉所有跟语者有关的信息，只保留和语者无关的信息，这个就是 Feature Disentangle 的技术。



这边就是一些真正的实验结果，搜集很多有声书给机器去学，左边是 Phonetic Encoder 的 output，右边是 Speaker Encoder 的 output。

上面两个图每一个点就代表一段声音讯号，这边不同颜色的点代表声音讯号背后对应的词汇是不一样的，但他们都是不同的人讲的。如果看 Phonetic Embedding 的 output 就会发现，同样的词汇它是被聚集在一起的。虽然他们是不同人讲的，但是 Phonetic Encoder 知道它会把语者的信息滤掉，知道不同人讲的声音讯号不太一样，但是这些都是同一个词汇。

Speaker Encoder output 很明显就分成两群，不同的词汇发音虽然不太一样，但是因为现在 Speaker Encoder 已经把发音的信息都滤掉只保留语者的信息，就会发现不同的词汇都是混在一起的。

下面是两个不同颜色的点代表两个不同的 speaker，两个不同的语者他们所发出来的声音讯号。

如果看 Phonetic Embedding，看发音上面的信息，两个不同的人他们很有可能会说差不多的内容，所以这两个 embedding 重迭在一起。

如果看 Speaker Encoding 就会发现这两个人的声音，是很明显的分成两群的，这个就是 Feature Disentangle。

这边是举语音做例子，但是它也可以用在影像等等其他 application 上

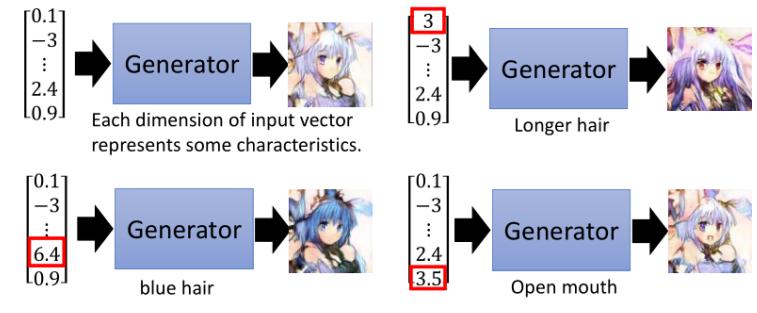
Intelligent Photo Editing by GAN

NVIDIA 自动修图是怎么做的？

我们知道假设 train 一个人脸的 generator，会 input 一个 random vector 然后就 output 一个人脸。

Modifying Input Code

在一开始讲 GAN 的时候跟大家说过 input vector 的每一个 dimension 其实可能对应了某一种类的特征，只是问题是并不知道每一个 dimension 对应的特征是什么，现在要讲的是怎么去反推出现在 input 的这个 vector 每一个 dimension 它对应的特征是什么。



➤ The input code determines the generator output.

➤ Understand the meaning of each dimension to control the output.

现在的问题是这个样子，你其实可以收集到大量的 image，你可以收集到这些 image 的 label，label 说这张 image 里面的人，是金头发的、是男的、是年轻的等等，你可以得到 image，你也可以得到 image 的特征，你也可以得到 image 的 label，但现在的问题是会搞不清楚这张 image 它到底应该是由什么 vector 所生成的。

Connecting Code and Attribute

假设你可以知道生成这张 image 的 vector 长什么样子，你就可以知道 vector 跟 label 之间的关系。

因为你有 image 跟它特征的 label，假设可以知道某一张 image 可以用什么样的 random vector 丢到 generator 就可以产生这张 image，你就可以把这个 vector 跟 label 的特征 link 起来。

现在的问题就是给你一张 image，你其实并不知道什么样的 random vector 可以产生这张 image。

所以这边要做的第一件事情是，假设已经 train 好一个 generator，这个 generator 给一个 vector z 它可以产生一个 image x。

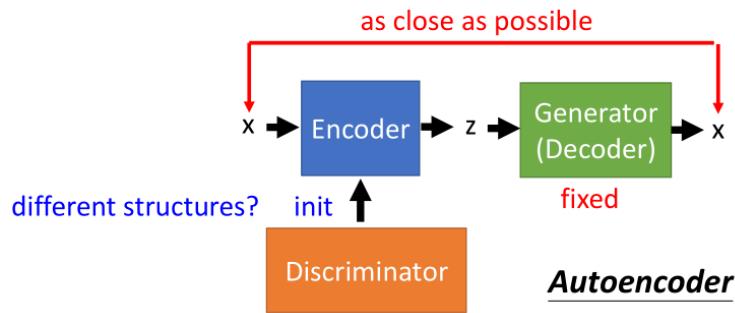
这边要做的事情是去做一个逆向的工程，去反推说如果给你一张现成的 image，什么样的 z 可以生成这张现成的 image。怎么做呢？

GAN + Autoencoder

这边的做法是再 learn 另外一个 encoder，再 learn 一个 encoder，这个 encoder 跟这个 generator 合起来就是一个 Autoencoder。在 train 这个 Autoencoder 的时候 input 一张 image x，它把这个 x 压成一个 vector z，希望把 z 丢到 generator 以后它 output 的是原来那张 image。

在 train 的过程中，generator 的参数是固定不动的，generator 是事先已经训练好的就放在那边，我们要做一个逆向工程猜出，假设 generator 产生某一张 image x 的时候，应该 input 什么样的 z，要作一个反向的工程。这个怎么做？

- We have a generator (input z, output x)
- However, given x, how can we find z?
 - Learn an encoder (input x, output z)



就是 learn 一个 encoder，然后在 train 的时候给 encoder 一张 image，它把这个 image 变成一个 code z，再把 z 丢到 generator 里面让它产生一张 image x，希望 input 跟 output 的 image 越接近越好。

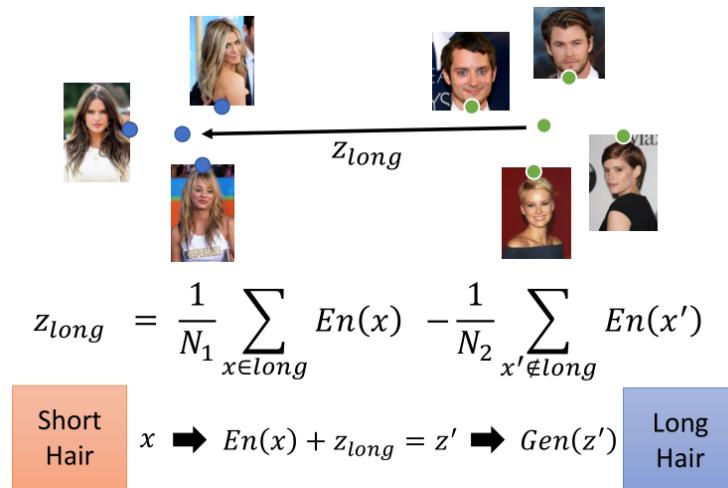
在 train 的时候要记得这个 generator 是不动的，因为我们是要对 generator 做逆向的工程，我们是要反推它用什么样的 z 可以产生什么样的 x，所以这个 generator 是不动的，我们只 train encoder 就是了。

在实现上，这个 encoder 因为它跟 discriminator 很像，所以可以拿 discriminator 的参数来初始化 encoder 的参数，这是一个实验的细节。

接下来假设做了刚才那件事以后就得到一个 encoder，encoder 做的事情就是给一张 image 它会告诉你这个 image 可以用什么样的 vector 来生成。

Attribute Representation

现在你就把 database 里面的 image 都倒出来，然后反推出他们的 vector，就是这个 vector 可以生成这张图。



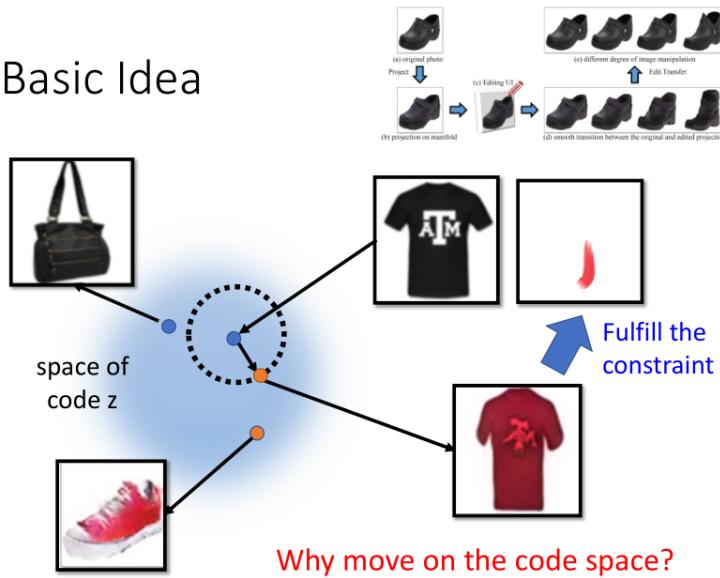
然后我们知道这些 image 他们的特征，这些是短发的人脸，这些是长发的人脸，把短发的人脸它的 code 推出来，再平均就得到一个短发的人脸的代表，把这个长发的人脸的 code 都平均就得到长发人脸的代表。再把他们相减就知道在这个 code space 上面做什么样的变化，就可以把短发的脸变成长发的脸，你把短发的脸加上这个向量 z long 它就会变成长发。

z_{long} 是怎么来的？你就把长发的 image x 它的 code 都找出来，把 x 丢到 encoder 里面，把它 code 都找出来，然后变平均得到这个 z，把这个不是长发的，短发的 code 都找出来平均，得到这个点，这两个点相减，就得到 z_{long} 这个向量。

接下来在生成 image 的时候，给你一张短发，你怎么把它变长发呢？，给你一张短发 image x，你把 x 这张 image 丢到 encoder 里面得到它的 code，再加上 z_{long} 得到新的 vector z' ，再把 z' 丢到 generator 里面就可以产生一张长发的图。

Another Idea

Basic Idea



有另外一个版本的智能的 Photoshop。

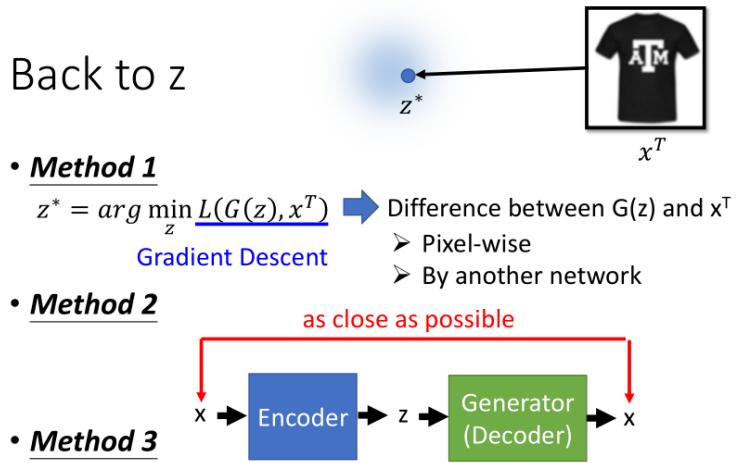
这个做法是这样，首先 train 一个 GAN，train 一个 generator，这个 generator train 好以后，这个 generator 你从它的 latent space 随便 sample 一个点，假设 train 的时候是用商品的图来 train，那你在 latent space 上面、在 z 的 space 上面，随便 sample 一个 vector，丢到 generator 里面它就 output 一个商品。你拿不同位子做 sample 会 output 出不同商品，那接下来刚才看到智能的 Photoshop，给一张图片，然后在这个图片上面稍微做一点修改，结果就会产生一个新的商品，这件事情是怎么做的？

这个做法大概是这个样子，先把这张图片反推出它在 code space 上面的哪一个位子，然后接下来在 code space 上面做一下小小的移动，希望产生一张新的图片，这张新的图片一方面跟原来的图片够像，一方面它跟原来的图片够像，新的图片跟原来的图片够像，但同时又要满足使用者给的指示，比如使用者说这个地方是红色的，所以产生的图片在这个地方是红色的，但它仍然是一件 T-shirt。

假设 GAN train 的够好的话，只要在 code space 上做 sample，你在这 code space 上做一些移动，你的 output 仍然会是一个商品，只是有不同的特征。

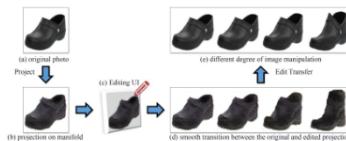
所以你已经推出这张 image 对应的 code 就在这个地方，你把它小小的移动一下，就可以产生一张新的图，然后这张新的图要符合使用者给你的 constrain，接下来实际上怎么做的呢？

实际上会遇到的第一个问题就是要给一张 image，你要反推它原来的 code 长什么样子，怎么做到这件事？



有很多不同的做法，举例来说一个可行的做法是你把它当作一个 optimization 的 problem 来解。你就在这个 code space 上面想要找到一个 vector z^* ， z 可以产生所有的 image X^T ，所以要解的是这样一个 optimization problem，要找一个 z^* 。把这个 z^* 丢到 generator 以后产生一张 image。

这个 $G(z)$ 代表一张产生出来的 image，产生出来的 image 要跟原来的图片 X^T 越接近越好。 L 是一个 Loss Function，它代表的是要衡量这个 $G(z)$ 这张图片跟 X^T 之间的差距。至于怎么衡量他们之间的差距有很多不同的方法，比如说你用 Pixel-wise 的方法，直接衡量 $G(z)$ 这张图片跟 X^T 的 L1 或 L2 的 loss，也有人会说它是用 Perception Loss，所谓 Perception Loss 是拿一个 pretrain 好的 classifier 出来，这个 pretrain 好的 classifier 就吃这张图片得到一个 embedding，再吃 X^T 得到一个 embedding，希望 $G(z)$ 根据 pretrain 的 classifier（比如



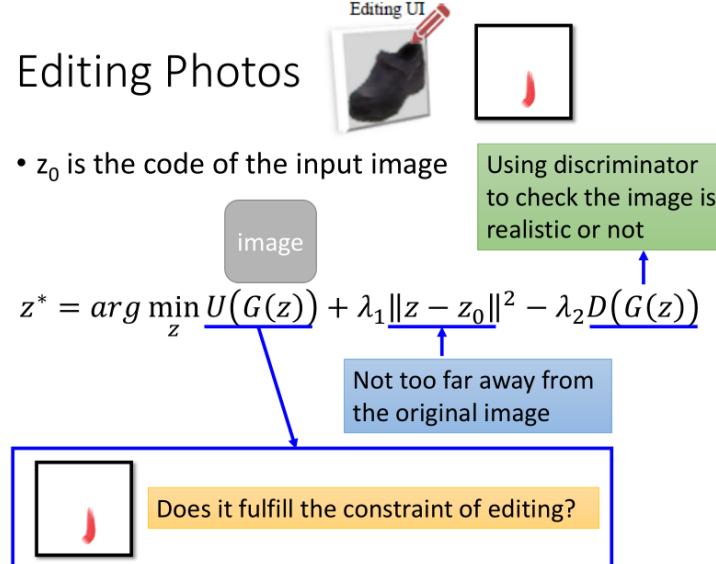
VGG) 得到 embedding, 跟 X^T 得到的embedding, 越接近越好。找一个 z^* , 这个 z^* 丢到 generator 以后, 它跟你目标的图片 X^T 越接近越好, 就得到了 z^* , 这是一个方法, 解这个问题可以用 Gradient Descent 来解。

第二个方法就是我们刚才在讲前一个 demo 的时候用的方法。

learn 一个 encoder, 这个 encoder 要把一张 image 变成一个 code z , 这个 z 丢到 generator 要产生回原来的 image, 这是个 Autoencoder, 你希望 input 跟 output 越接近越好。

还有一个方法, 就是把第一个方法跟第二个方法做结合, 怎么做结合?

因为第一个方法要用 Gradient Descent, Gradient Descent 可能会遇到一个问题就是 Local Minimum 的问题, 所以在不同的地方做 initialization, 给 z 不同的 initialization 找出来的结果是不一样的, 你先用方法 2 得到一个 z , 用方法 2 得到的 z 当作方法 1 的 initialization, 再去 fine tune 你的结果, 可能得到的结果会是最好的。



总之有不同方法可以从 x 反推 z , 你可以从 x 反推 z 以后, 接下来要解另外一个 optimization problem, 这个 optimization problem 是要找一个 z , 这个 z 可以做到什么事情?

这个 z 一方面, 你把 z 丢到 generator 产生一张 image 以后, 这个 image 要符合人给的 constrain, 举例来说是这个地方要是红色的等等。

U 代表有没有符合 constrain, 那至于什么样叫做符合 constrain 这个就要自己去定义, 写智能 Photoshop 的 developer 要自己去定义。

你用 $G(z)$ 产生一张 image, 接下来用 U 这个 function 去算这张 image 有没有符合人定的 constrain。这是第一个要 minimize 的东西。

第二个要 minimize 的东西是你希望新找出来的 z , 跟原来的 z , 假设原来是一只鞋子, 原来这只鞋子, 你反推出它的 z 就是 z_0 , 你希望做一下修改以后, 新的 z 跟原来的 z_0 越接近越好。因为你不希望本来是一张鞋子, 然后你画一笔希望变红色的鞋子, 但它变成一件衣服, 不希望这个样子, 你希望它仍然是一只鞋子, 所以希望新的 vector z 跟旧的 z_0 他们越接近越好。

最后还可以多加一个 constrain, 这个 constrain 是来自于 discriminator, discriminator 会看你把 z 丢到 generator 里面再产生出来的 image 丢到 D 里面, 把 generator output 再丢到 discriminator 里面, discriminator 去 check 这个结果是好还是不好。

你要找一个 z 同时满足这三个条件, 你要找一个 z 它产生出来的 image 符合使用者给的 constrain, 你要找一个 z 它跟原来的 z 不要差太多, 因为你希望 generate 出来的东西跟原来的东西仍然是同一个类型的, 希望找一个 z 它可以骗过 discriminator, discriminator 觉得你产生出来的结果是好的, 就解这样一个 optimization problem, 可以用 Gradient Descent 来解, 就找到一个 z^* , 这个就可以做到刚才讲的智能的 Photoshop, 就是这个做出来的。

Image super resolution

GAN 在影像上还有很多其他的应用, 比如说它可以做 Super-resolution。

你完全可以想象怎么做 Super-resolution, 它就是一个 Conditional GAN 的 problem, input 模糊的图 output 就是清晰的图。

input 是模糊的图, output 是清晰的图就结束了, 要 train 的时候要搜集很多模糊的图跟清晰的图的 pair, 要搜集这种 pair 很简单, 就把清晰的图故意弄模糊就行了, 实作就是这么做, 清晰的图弄模糊比较容易, 模糊弄清晰比较难。

- Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", CVPR, 2016



Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

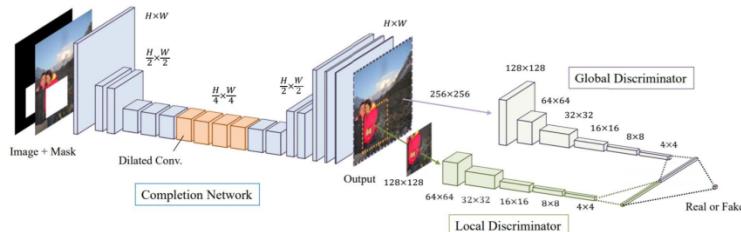
这个是文献上的结果，这个是还满知名的图，如果你有看过 GAN 的介绍，通常都会引用这组图，最左边这个是传统的、不是用 network 的方法得到的结果，产生出来的图是比较模糊的。第二个是用 network 的方法产生出来的图。最右边是原图。第三个是用 GAN 产生的出来的图，你会发现如果用 network 虽然比较清楚，但是在一些细节的地方，比如说衣领的地方，这个头饰的地方还是有些模糊的。但是如果看这个 GAN 的结果的话，在衣领和头饰的地方，花纹都是满清楚的。

有趣的地方是衣领的花纹虽然清楚，但衣领的花纹跟原图的花纹其实不一样，头饰的花纹跟原图的花纹是不一样，机器自己创造出清晰的花纹，反正能骗过 discriminator 就好，未必要跟原来的花纹是一样的，这是 image 的 Super-resolution。

Image Completion

Image Completion

<http://hi.cs.waseda.ac.jp/~iizuka/projects/completion/en/>



现在还会做的事情是 Image Completion，Image Completion 就是给一张图片，然后它某个地方挖空，机器自己把挖空的地方补上去，这个怎么做？

这个就是 Conditional GAN，就是给机器一张有挖空的图，它 output 一张填进去的图就结束了，怎么产生这样的 training data？它很容易产生，就找一堆图片，中间故意挖空就得到这种 training data pair，然后就结束了。

Improving Sequence Generation by GAN

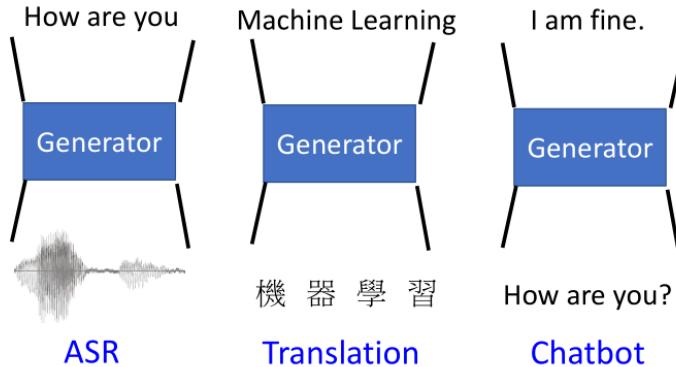
我们今天要讲的是用 GAN，来 improve sequence generation。

那 sequence generation 的 task 它有非常多的应用，我们先讲怎么用 GAN 来 improve conditional sequence generation。

接下来我们会讲，我们还可以做到 Unsupervised conditional sequence generation。

Conditional Sequence Generation

只要是要产生一个 sequence 的 task，都是 conditional sequence generation。



The generator is a typical seq2seq model.

With GAN, you can train seq2seq model in another way.

举例来说语音识别可以看作是一个 conditional sequence generation 的 task。你需要的是一个 generator, input 是声音讯号, output 就是语音识别的结果就是这一段声音讯号所对应到的文字。或者假设你要做翻译, 你要做 translation 的话, 你的 input 是中文, output 就是翻译过的结果, 是一串 word sequence。或者是说 chatbot 也是一个 conditional sequence generation 的 task, 它的 input 是一个句子, output 是另外一个 sequence。

那我们之前有讲过, 其实这些 task, 语音识别, 翻译或 chatbot, 都是用 sequence to sequence 的 model 来解它的, 所以实际上这边这个图上所画的 generator, 它们都是 sequence to sequence 的 model。

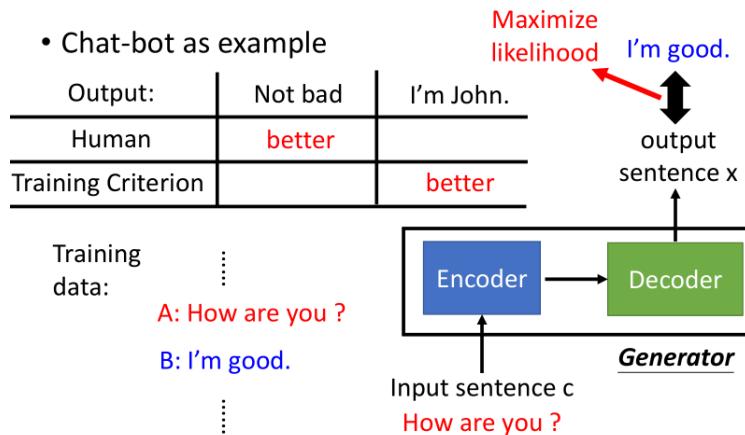
只是今天要讲的是用一个不一样的方法, 用 GAN 的技术来 train 一个 seq2seq model。

那为什么我们会要用到 GAN 的技术或其他的技术来 train seq2seq model 呢。我们先来看看我们 train seq2seq model 的方法有什么不足的地方, 假设我们就 train 了一个 chatbot, 一个 chatbot 它是一个 seq2seq model, 它里面有一个 encoder, 有一个 decoder, 这个 seq2seq model 就是我们的 generator, 那这个 encoder 会吃一个 input 的句子, 这边用 c 来表示, 那它会 output 另外一个句子 x , encoder 吃一个句子, 之于 decoder 会 output 一个句子 x 。

那我们知道要 train 这样子的 chatbot, 你需要收集一些 training data。所谓的 training data 就是人的对话, 所以你今天告诉 chatbot 说, 在这个 training data 里面, A 说 How are you 的时候, B 的响应是 I'm good, 所以 chatbot 必须学到, 当 input 的句子是 How are you 的时候, 它 output 这个 I'm good 的 likelihood 应该越大越好。

意思就是说, 今天假设正确答案是 I'm good, 那你在用 decoder 产生句子的时候, 第一个 time step 产生 I'm 的机率要越大越好, 那在第二个 time step 产生 good 的机率要越大越好。

那这么做显然有一个非常大的问题, 就是我们看两个可能的 output, 假设今天有一个 chatbot, 它 input How are you 的时候, 它 output 是 Not bad, 有另外一个 chatbot, 它 input How are you 的时候, 它 output 是 I'm John。



如果从人的观点来看, Not bad 是一个比较合理的 answer, I'm John 是一个比较奇怪的 answer。但是如果从我们 training 的 criteria 来看, 从我们在 train 这个 chatbot 的时候, 希望 chatbot 要 maximize 的 object 希望 chatbot 学到的结果来看, 事实上 I'm John 是一个比较好的结果, 为什么呢? 因为 I'm John 至少第一个 word 的还是对的, 那如果是 Not bad, 你两个 word 都是错的, 所以从这个 training 的 criteria 来看是这样子的, 假设你 train 的时候是 maximum likelihood。

其实 maximum likelihood 就是 minimize 每一个 time step 的 cross entropy, 这两个其实是 equivalent 的东西, maximum likelihood 就是 minimize cross entropy。这是一个真正的例子, 某人去面试某一个大家都知道的, 全球性的科技公司, 被问了这个问题, 人家问他说, train 这个 classifier 的时候, 有时候我们会说我们是 maximum likelihood, 有时候我们会说我们是在 minimize cross entropy, 这两者有什么不同呢? 如果你答这两个东西有点像, 但他们中间有微妙的不同, 你就错了。这个时候你就是要说, 他们两个就是一模一样的东西,

maximum likelihood 跟 minimize cross entropy, 是一模一样的东西。

Improving Supervised Seq-to-seq Model

RL (human feedback)

我们先讲一下怎么去 improve 这个 seq2seq 的 model。

我们会先讲，怎么用 reinforcement learning 来 improve conditional generation。然后接下来我们才会讲说，怎么用 GAN 来 improve conditional generation。

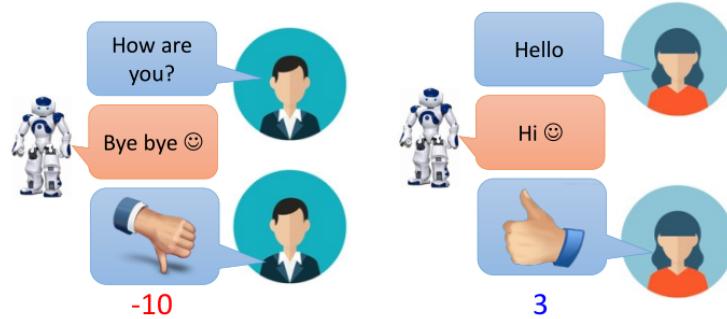
之所以要讲 RL，是因为等一下你会发现，用 GAN 来 improve conditional generation 这件事情，其实跟 RL 是非常像的。你甚至可以说使用 RL，来 improve seq2seq 的 chatbot，可以看作是 GAN 的一个 special case。

假设我们今天要 train 一个 seq to seq 的 model，你不想要用 train maximum likelihood 的方法，来 train seq to seq model，因为我们刚才讲用 maximum likelihood 的方法有很明显的问题。

我们都用 chatbot 来做例子，其实我们讨论的技术，不是只限于 chatbot 而已，任何 seq to seq model，都可以用到等一下讨论的技术。不过我们等一下举例的时候，我们都假设我们是要做 chatbot 就是了。

那今天假设你要 train 一个 chatbot，你不要 maximum likelihood 的方法，你想要 Reinforcement learning 的方法，那你会怎么做呢？

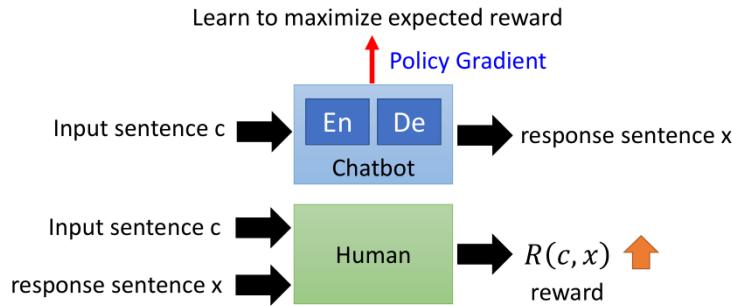
Machine obtains feedback from user



Chat-bot learns to maximize the expected reward

你的做法可能是这样，你就让这个 chatbot 去胡乱在线上跟人讲话，就有一个人说 How are you，chatbot 就回答 bye-bye，人就会给 chatbot 一个很糟的评价，chatbot 就知道说这样做是不好的；再下一次他跟人对话的时候，人说 Hello，chatbot 说 Hi，人就觉得说它的回答是对的，就给它一个 positive 的评价，chatbot 就知道说它做的事情是好的。那 chatbot 在跟人互动的过程中呢，他会得到 reward。把这个问题想的单纯一点，就是人说一个句子，然后 chatbot 就做一个响应，人就会给 chatbot 一个分数，chatbot 要做的事情，就是希望通过互动的过程，它去学习怎么 maximize 它可以得到的分数，

Maximizing Expected Reward



[Li, et al., EMNLP, 2016]

我们现在的问题是，有一个 chatbot，它 input 一个 sentence c ，要 output 一个 response x ，它就是一个 seq to seq model，接下来有一人，人其实也可以看作是一个 function，人这个 function 做的事情就是，input 一个 sentence c ，还有 input 一个 response x ，然后给一个评价，给一个 reward，这个 reward 我们就写成 $R(c, x)$ ，但如果你熟悉 conditional generation 的话，你会发现这个图，跟用 GAN 做 conditional generation，其实是非常像的。唯一的不同是，如果用 GAN 做 conditional generation 的话，这个绿色的方块，它是一个 discriminator。切记 discriminator 它不要只吃 generator 的 output，它要同时吃 generator 的 input 跟 output，才能给与评价。今天人

也是一样，人来取代那个 discriminator，人就不用 train，或者是说你可以说人已经 train 好了，人有一个脑，然后在数十年的成长历程中其实已经 train 好了，所以你不用再 train。给一个 input sentence c ，给一个 response x ，然后你可以给一个评价。

我们接下来要做的事情，chatbot 要做的事情就是，它调整这个 seq to seq model 里面内部的参数，希望去 maximize 人会给它的评价，这边写成 $R(c, x)$ ，这件事情怎么做呢？我们要用的技术就是 policy gradient。policy gradient 我们其实在 machine learning 的最后几堂课其实是有说过的。

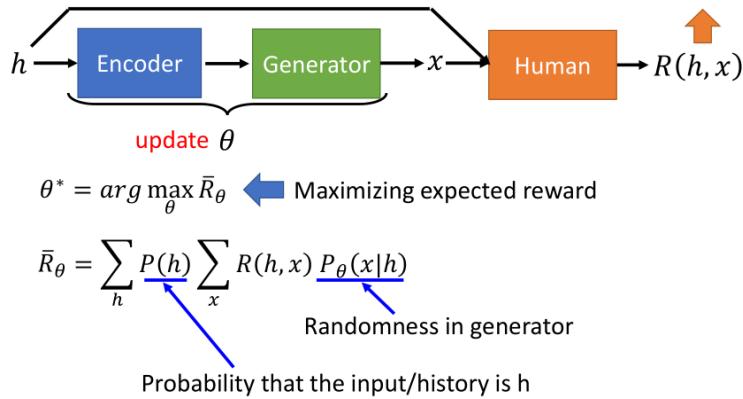
(文章中的 c ，与 slides 中的 h 是同一个东西)

我们这边以 chatbot 做例子，来很快地复习一下，policy gradient 是怎么做的。

那我们有一个 seq to seq model，它的 input 是 c output 是 x 。接下来我们有另外一个 function，这个 function 是人，人吃 c 跟 x ，然后 output 一个 R 。

那我们现在要做的事情是什么呢？我们要去调 encoder 跟 generator 的参数，这个 encoder 跟 generator 合起来是一个 seq to seq model，他们合起来的参数，我们叫做 θ 。我们希望调这个 θ 去 maximize human 这个 function 的 output。

那怎么做呢，我们先来计算给定某一组参数 θ 的时候，这个时候这个 chatbot，会得到的期望的 reward 有多大。假设这个 θ 是固定的，然后计算一下这个 seq to seq model，它会得到的期望的 reward 是有多大。



怎么算呢？首先我们先 summation over 所有可能的 input c ，然后乘上每一个 c 出现的机率，因为 c 可能有各种不同的 output，比如说人可能说 How are you，人可能说 Good morning，人可能说 Good evening，你有各种各样的 input，你有各种各样的 c ，但是每一个 input 出现的机率，可能是不太一样的，比如说 How are you 相较于其他的句子，也许它出现的机率是特别大的，因为人特别常对 chatbot 说这个句子。接下来，summation over 所有可能的回应 x 。当你有一个 c 的时候，当你有一个 input c ，再加上假设这个 chatbot 的参数 θ 我们已经知道的时候，接下来你就可以算出一个机率，这个机率是在 given c 这组参数的情况下，chatbot 会回答某一个答复 x 的机率有多少。给一个 input c ，为什么 output 会是一个机率呢？

你想想看，我们今天在 train seq to seq model 的时候，每一个 time step 我们不是其实要做一个 sampling 嘛，我们 train 一个 seq to seq model 的时候，每一次给同样的 input，它的 output，不见得是一样的，假设你在做 sampling 的时候，我们的 decoder 的 output 是一个 distribution，你要把 distribution 变成一个 token 的时候，如果你是采取 sampling 的方式，那你 chatbot 的每一次 output 都会是不一样的。所以今天给一个 c ，每一次 output 的 x ，其实是不一样的，所以给一个 c ，我们其实得到的是一个 x 的机率。

假设你不是用 sampling 的方式，你是用 argmax 的方式呢？其实也可以，如果是用 argmax 的方式，给一个 c ，那你一定会得到一模一样的 x ，但我们可以这么说，那个 x 出现的机率就是 1，其他的 response 出现的机率都是 0，其他的 x 出现机率都是 0。

总之给你一个 c ，在参数 x, θ 知道的情况下，你可以把 chatbot 可能的 output 看作是一个 distribution，写成 $P_{\theta}(x | c)$ 。当给一个 c ，chatbot 产生一个 x 的时候，接下来人就会给一个 reward $R(c, x)$ 。

这一整项 summation over 所有的 c ，summation over 所有的 x ，这边乘上 c 的机率，这边乘上 x 出现的机率，再 weighted by 这个 reward，其实就是 reward 的期望值。

接下来我们要做的事情就是，我们要调这个 θ ，要调这个 chatbot 的参数 θ ，让 reward 的期望值，越大越好，那这件事情怎么做呢？

$$\theta^* = \arg \max_{\theta} \bar{R}_{\theta} \quad \text{Maximizing expected reward}$$

$$\begin{aligned} \bar{R}_{\theta} &= \sum_h P(h) \sum_x R(h, x) P_{\theta}(x|h) = E_{h \sim P(h)} [E_{x \sim P_{\theta}(x|h)} [R(h, x)]] \\ &= E_{h \sim P(h), x \sim P_{\theta}(x|h)} [R(h, x)] \approx \frac{1}{N} \sum_{i=1}^N R(h^i, x^i) \end{aligned}$$

Where
is θ ?

Sample: $(h^1, x^1), (h^2, x^2), \dots, (h^N, x^N)$

我们把这个 reward 的期望值稍微做一下整理，就是我们从 P of c 里面 sample 出一个 c 来，我们从这个机率里面 sample 出一个 x 来，然后取 R(c, x) 的期望值。

然后接下来的问题就是，这个期望值要怎么算？

你要算这个期望值，theoretical 做法要 summation over 所有的 c, summation over 所有的 x, 但是在实作上，你根本无法穷举所有 input，你根本无法穷举所有可能 output。

所以实作上就是做 sampling，假设这两个 distribution 我们知道。P(c)，人会说什么句子，你就从你的 database 里面 sample 看看，从 database 的句子里面 sample，就知道人常输入什么句子，那 $P_\theta(x | c)$ ，你只要知道参数，他就是给定的。

所以我们根据这两个机率，去做一些 sample，我们去 sample 大 N 笔的 c 跟 x 的 pair，比如说上百笔的 c 跟 x 的 pair。

所以来这边应该是要取一个期望值，但实际上我们并没有办法真的去取期望值，我们真正做法是，做一下 sample，sample 出大 N 笔 data，这大 N 笔 data，每一笔都去算它的 reward，把这大 N 笔 data 的 reward 全部平均起来，我们用 $\sum_{i=1}^N R(c^i, x^i)$ 来 approximate 期望值， $\frac{1}{N} \sum_{i=1}^N R(c^i, x^i)$ 就是期望的 reward 的 approximation。

那我们现在要对 θ 做 optimization，我们要找一个 θ 让 \bar{R}_θ 这一项越大越好，那意味着说我们要拿 θ 去对 \bar{R}_θ 算它的 gradient。

但是问题是在 $\frac{1}{N} \sum_{i=1}^N R(c^i, x^i)$ 里面，我们说 \bar{R}_θ 就等于这项，这项里面没有 θ 啊，没有 θ 你根本没有办法对 θ 算 gradient，不知不觉间，它就不见了，它到哪里去了呢？

它被藏到 sampling 的这个 process 里面去了，当你改变 θ 的时候，你会改变 sample 到的东西，但在这式子里面， θ 就不见了，你根本就不知要怎么对这个式子算 θ 的 gradient，所以怎么办呢？

Policy Gradient

实作上的方法是这个样子的，这一项如果把它 approximate 成 $\frac{1}{N} \sum_{i=1}^N R(c^i, x^i)$ 的话，就会没有办法算 gradient 了，所以怎么办？

先把对 \bar{R}_θ 算 gradient，再做 approximation，这一项算 gradient 是怎么样呢？只有 $P_\theta(x | c)$ 跟 θ 是有关的，所以你对 \bar{R}_θ 取 gradient 的时候，那你只需要把 gradient 放到 P_θ 的前面就好了。

接下来，唯一的 trick 是对这一个式子，分子和分母都同乘 $P_\theta(x | c)$ ，分子分母同乘一样的东西，当然对结果是没有任何影响的。

那我们知道右上角的式子，微分告诉我们反正就是这个样子。

所以今天这个式子，其实蓝框里面的这两项是一样的。

Policy Gradient	$\frac{d \log(f(x))}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx}$
$\bar{R}_\theta = \sum_h P(h) \sum_x R(h, x) P_\theta(x h) \approx \frac{1}{N} \sum_{i=1}^N R(h^i, x^i)$	
$\nabla \bar{R}_\theta = \sum_h P(h) \sum_x R(h, x) \nabla P_\theta(x h) \approx \frac{1}{N} \sum_{i=1}^N R(h^i, x^i) \nabla \log P_\theta(x^i h^i)$	
$= \sum_h P(h) \sum_x R(h, x) P_\theta(x h) \frac{\nabla P_\theta(x h)}{P_\theta(x h)}$	
$= \sum_h P(h) \sum_x R(h, x) P_\theta(x h) \nabla \log P_\theta(x h)$	
$= E_{h \sim P(h), x \sim P_\theta(x h)} [R(h, x) \nabla \log P_\theta(x h)]$	

Sampling

那接下来呢，变成期望值的形式。

所以这一项，当你要对 \bar{R}_θ 做 gradient 的时候，你要去 approximate $\nabla \bar{R}_\theta$ 的话，你是怎么算的呢？

这一项就是，把 summation 换做 sampling，你就 sample 大 N 项，每一项都去算 $R(c^i, x^i) \nabla \log P_\theta(x^i | c^i)$ ，把它们平均起来就是 expectation 的 approximation。

所以我们实际上是怎么做的呢？你 update 的方法是，原来你的参数叫做 θ^{old} ，然后你用 gradient ascent 去 update 它，加上某一个 gradient 的项，你得到新的 model θ^{new} ，gradient 这一项怎么算？gradient 这一项算法就是，去 sample N 个 pair 的 c^i 跟 x^i 出来，然后计算 $R(c^i, x^i) \nabla \log P_\theta(x^i | c^i)$ ，就结束了。

其实这一项它是非常的直觉的，怎么说它非常的直觉呢？这个 gradient 所代表的意思是说，假设今天 given c_i, x_i ，也就是说有人对 machine 说了 c_i 这个句子，machine 回答 x_i 这个句子，然后人给的 reward 是 positive 的，那我们就要增加 given c_i 的时候， x_i 出现的机率。反之如果 $R(c_i, x_i)$ 是 negative 的，当人对 chatbot 说 c_i ，chatbot 回答 x_i ，然后得到负面的评价的时候，这个时候我们就应该调整参数 θ ，让 given c_i ，回答 x_i 的这个机率呢，越小越好。

Gradient Ascent

$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

$$\nabla \bar{R}_{\theta} \approx \frac{1}{N} \sum_{i=1}^N R(h^i, x^i) \nabla \log P_{\theta}(x^i | h^i)$$

$R(h^i, x^i)$ is positive

→ After updating θ , $P_{\theta}(x^i | h^i)$ will increase

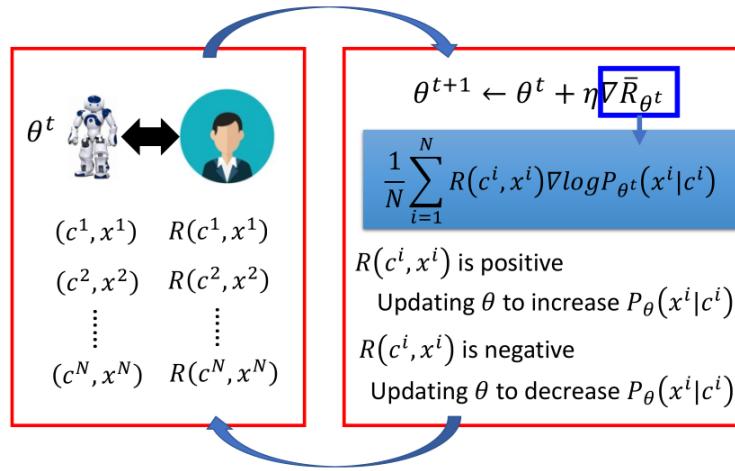
$R(h^i, x^i)$ is negative

→ After updating θ , $P_{\theta}(x^i | h^i)$ will decrease

所以实作上的时候，如果你要用 policy gradient 这个技术，来 implement 一个 chatbot，让它在 reinforcement learning 的情境中，可以去学习怎么和人对话的话，实际上你是怎么做的呢？

Implementation

实际上你的做法是这个样子，你有一个 chatbot 它的参数叫做 $\theta(t)$ ，然后你把你的 chatbot 拿去跟人对话，然后他们就讲了很多，这个是一个 sampling 的 process。



你先用 chatbot 跟人对话，做一个 sampling 的 process，在这个 sampling 的 process 里面，当人说 c_1 chatbot 回答 x_1 的时候，会得到 reward $R(c_1, x_1)$ ，当输入 c_2 回答 x_2 的时候，会得到 reward $R(c_2, x_2)$ ，那你会 sample 出 N 笔 data，每一笔 data 都会得到一个 reward， N 笔 data N 个 reward。

接下来你做的事情是这样，你有一个参数 $\theta(t)$ ，你要 update 这个参数，让它变成 $\theta(t+1)$ 。那怎么 update 呢？你要把它加上对这个 \bar{R}_{θ} 的 gradient，那这个 \bar{R}_{θ} 的 gradient 这一项到底怎么算呢？这一项式子就列在这边，那这个式子的直观解释我们刚才讲过说，如果 R of (c_i, x_i) 是正的，那就增加这一项的机率，如果 R of (c_i, x_i) 是负的，就减少这一项的机率。

但是你要注意，每次你 update 完参数以后，你要从头回去，再去 sample data，因为这个 \bar{R}_{θ} 它是在 given 参数是 θ 的情况下，所算出来的结果，一旦 update 你的参数，从 $\theta(t)$ 变成 $\theta(t+1)$ ，gradient 这一项就不对了，你本来参数 $\theta(t)$ ，一旦你 update 变成 $\theta(t+1)$ 以后，你就要回过头去再重新收集参数。

所以这跟一般的 gradient decent 非常不同，因为一般 gradient decent，你就算 gradient，然后就可以 update 参数，然后就可以马上再算下一次 gradient，再 update 参数。

但是如果你 apply reinforcement learning 的时候，你的做法是，每次你 update 完参数以后，你就要去跟使用者再互动，然后才能再次 update 参数，所以每次 update 参数的时间呢，需要的 effort 是非常大的。每 update 一次参数，你就要跟使用者互动 N 次，才能 update 下一次参数，所以在 policy gradient 里面，update 参数这件事情，是非常宝贵的，就这一步是非常宝贵的，绝对不能够走错这样子，你一走错，你就要你要重新再去跟人互动，才能够走回来，那你也有可能甚至就走不回来。所以之后会讲到一些新的技术，来让这一步做得更好，不过这是我们之后才要再讲的东西。

Comparison

	Maximum Likelihood	Reinforcement Learning
Objective Function	$\frac{1}{N} \sum_{i=1}^N \log P_\theta(\hat{x}^i c^i)$	$\frac{1}{N} \sum_{i=1}^N R(c^i, x^i) \log P_\theta(x^i c^i)$
Gradient	$\frac{1}{N} \sum_{i=1}^N \nabla \log P_\theta(\hat{x}^i c^i)$	$\frac{1}{N} \sum_{i=1}^N R(c^i, x^i) \nabla \log P_\theta(x^i c^i)$
Training Data	$\{(c^1, \hat{x}^1), \dots, (c^N, \hat{x}^N)\}$ $R(c^i, \hat{x}^i) = 1$	$\{(c^1, x^1), \dots, (c^N, x^N)\}$ obtained from interaction weighted by $R(c^i, x^i)$

那这边是把 reinforcement learning 跟 maximum likelihood 呢，做一下比较，在做 maximum likelihood 的时候，你有一堆 training data，这些 training data 告诉我们说，今天假设人说 c_1 ，chatbot 最正确的回答是 x_1 hat，我们就会有 labeled 的 data 嘛，就你有 input c_1 output x_1 hat，，input c_N 正确答案就是 x_N hat，这是 training data 告诉我们的，在 training 的时候，你就是 maximize 你的 likelihood，怎么样 maximize 你的 likelihood 呢？

你希望 input c_i 的时候，output x_i hat 的机率越大越好，input 某个 condition，input 某个 input 的时候，input 某个输入的句子的时候，你希望正确的答案出现的机率越大越好，那算 gradient 的时候很单纯，你就把这个 $\log P_\theta$ 前面呢，加上一个 gradient，你就算 gradient 了，这个是 maximum likelihood。

那我们来看一下 reinforcement learning，在做 reinforcement learning 的时候呢，你也会得到一堆 c 跟 x 的 pair，但这些 c 跟 x 的 pair，它并不是正确的答案，这些 x 并不是人去标的答案，这些 x 是机器自己产生的，就人输入 c_1 到 c_N ，机器自己产生了 x_1 到 x_N ，所以有些答案是对的，有些答案有可能是错的。

接下来呢我们说，我们在做 reinforcement learning 的时候，我们是怎么计算 gradient 的呢？我们是用这样的式子来计算 gradient，所以我们实际上的作法呢，我们这个式子的意思就是把这个 gradient $\log P_\theta$ 前面乘上 $R(c, x)$ 。

就如果你比较这两个式子的话，你会发现说他们唯一的差别是，在做 reinforcement learning 的时候，你在算 gradient 的时候，每一个 x 跟 c 的 pair 前面都乘上 $R(c, x)$ ，如果你觉得这个 gradient 算起来不太直观，那没关系，我们根据这个 gradient，反推 objective function。我们反推说什么样的 objective function，在取 gradient 的时候，会变成下面这个式子。那如果你反推了以后，你就会知道说，什么样的 objective function，取 gradient 以后会变成下面这个式子呢？你的 objective function 就是，summation over 你 sample 到的 data，每一笔 sample 到的 data，你都乘上 $R(c, x)$ ，然后你去计算每一笔 sample 到的 data 的 log 的 likelihood，你去计算每一笔 sample 到的 data 的 $\log P_\theta$ ，再把它乘上 $R(c, x)$ ，就是你的 objective function。

把这个 objective function，做 gradient 以后，你就会得到这个式子。我们在做 reinforcement learning 的时候，我们每一个 iteration，其实是在 maximize 这样一个 objective function，那如果你把这两个式子做比较的话，那就非常清楚了。右边这个 reinforcement learning 的 case，可以想成是每一笔 training data 都是有 weight，而在 maximum likelihood 的 case 里面，每一笔 training data 的 weight 都是一样的，每一笔 training data 的 weight 都是 1，在 reinforcement learning 里面，每一笔 training data 都有不同的 weight，这一个 weight 就是那一笔 training data 得到的 reward。

也就是说今天输入一个 c_i ，机器回答一个 x_i ，如果今天机器的回答正好是好的，这个 x_i 是一个正确的回答，那我们在 training 的时候就给那笔 data 比较大的 weight。如果今天 x_i 是一个不好的回答，代表这笔 training data 是错的，我们 even 会给它一个 negative 的 weight，这个就是 maximum likelihood，和 reinforcement learning 的比较。

reward 理论上并没有特别的限制，你用 policy gradient，都可以去 maximize objective function，但是在实作上，会有限制，我们刚才不是讲到说，如果 R 是正的，你就要让机率越大越好，那你会不会遇到一个问题就是，假设 R 永远都是正的，今天这个 task R 就是正的，你做的最差，也只是得到的分数比较小而已，它永远都是正的，那今天不管你采取什么样的行为，machine 都会说我要让机率上升，听请来有点怪怪的。但是在理论上这样未必会有问题。

为什么说理论上这样未必会有问题呢？你想看，你要 maximize 的这一项，是一个机率，它的和是 1，所以今天就算是所有不同的 x_i ，他前面乘的 R 是正的，他终究是有大有小的，你不可能让所有的机率都上升，因为机率的和是 1，你不可能让所有机率都上升，所以变成说，如果 weight 比较大的，就比较 positive 的，就上升比较多，如果 weight 比较小的，比较 negative 的，它就可能反而是会减少的，就算是正的，但如果值比较小，它可能也是会减小，因为 constrain 就是它的和要是 1。

但是你今天在实作上并没有那么容易，因为在实作上会遇到的问题是，你不可能 sample 到所有的 x ，所以到时候就会变成说，假设一笔 data 你没有 sample 到，其他人只要有 sample 到都是 positive 的 reward。没 sample 到的，反而就会机率下降，而 sample 到的都会机率上升。这个反而不是我们要的。所以其实今天在设计那个 reward 的时候，你其实会希望那个 reward 是有正有负的，你 train 起来会比较容易，那假设你的 task reward 都是正的，实际上你会做的一件事情是，把 reward 通通都减掉一个 threshold，让它变成是有正有负，这样你 train 起来会容易很多。

这个是讲了 maximum likelihood, 跟 reinforcement learning 的比较。

Alpha GO style training

但是你知道实作上要做什么 reinforcement learning 根本就是不太可能的，有一个人写一篇网络文章说，当有人问他说某一个 task 用 reinforcement learning 好不好的时候，他的回答都是不好，多数的时候他都是对的。

要做 reinforcement learning 一个最大的问题就是，机器必须要跟人真的互动很多次，才能够学得起来。

你不要看今天google 或者是Deep mind或者是 OpenAI 他们在玩那些什么 3D 游戏都玩得很好这样，那个 machine 跟环境互动的次数都可能是上千万次，或者是上亿次，那么多互动的次数，除了在电玩这种 simulated 的 task 以外，在真实的情境，几乎是不可能发生。

所以如果你要用 reinforcement learning 去 train 一个 chatbot，几乎是不可能的。

因为在现实的情境中，人没有办法花那么多力气，去跟 chatbot 做互动，所以来就有人就想了一个 Alpha Go style training。也就是说我们 learn 两个 chatbot，让它们去互讲，例如有一个 bot 说 How are you。另外一个说 see you，然后它再说 see you，它说 see you，然后陷入一个无穷循环永远都跳不出来。它们有时候可能也会说出比较正确的句子，因为我们知道说机器在回应的时候其实是有随机性的。所以问它同一个句子，每次的回答不见得是一样的。

接下来你再去定一个 evaluation 的 function，因为你还是不可能说让两个 chatbot 互相对话，然后产生一千万则对话以后，人再去一千万则对话每一个去给它 feedback 说，讲得好还是不好，你可能会设计一个 evaluation function，这个就是人订一个 evaluation function，给一则对话，然后看说这则对话好不好，但是这种 evaluation function 是人订的，你其实没有办法真的定出太复杂的 function，就只能定义一些很简单的。就是，比如说陷入无穷循环，就是得到负的 reward，说出 I don't know，就是得到负的 reward，你根本没有办法真的订出太复杂的 evaluation function，所以用这种方法还是有极限的。

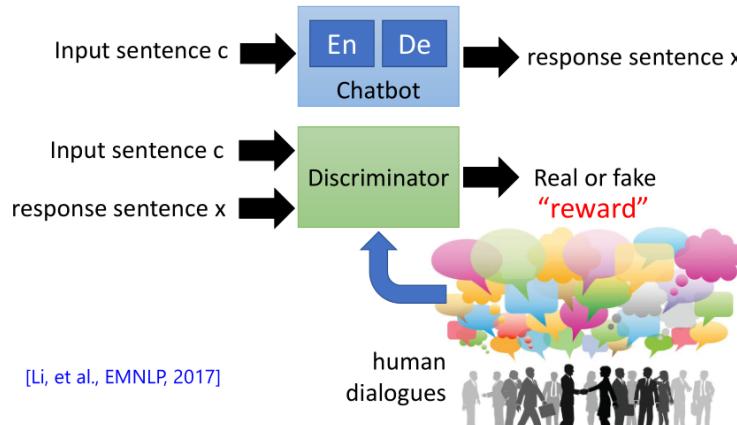
所以接下来要解这个问题，你可以引入 GAN 的概念。

GAN (discriminator feedback)

GAN 和 RL 有什么不同呢？在 RL 里面，你是人给 feedback，在 GAN 里面，你变成是 discriminator 来给 feedback。

我们一样有一个 chatbot，一样吃一个句子，output 另外一个句子，现在有一个 discriminator，这个 discriminator，其实就是取代了人的角色，它吃 chatbot 的 input 跟 output，然后吐出一个分数。

Conditional GAN



那这个跟 typical 的 conditional GAN 就是一样的，我们知道就算是别的 task，什么 image 的生成，你做的事情也是一样的。你就是有一个 discriminator，它吃你的 generator 的 input 跟 output，接下来给你一个评价，那在 chatbot 里面也是一样，你有一个 discriminator，它吃 chatbot input 的 sentence，跟 output sentence，然后给予一个评价。那这个 discriminator 呢，你要给它大量人类的对话，让它知道说真正的人类的对话，真正的当这个chatbot 换成一个人的时候，它的 c 跟 x 长什么样子，那这个 discriminator 就会学着鉴别这个 c 跟 x 的 pair，是来自于人类，还是来自于 chatbot。然后 discriminator 会把他学到的东西，feedback 给 chatbot，或者是说 chatbot 要想办法骗过这个 discriminator。那这跟 conditional GAN就是一模一样的事情了。

Algorithm

那这个 algorithm 是什么样子呢？

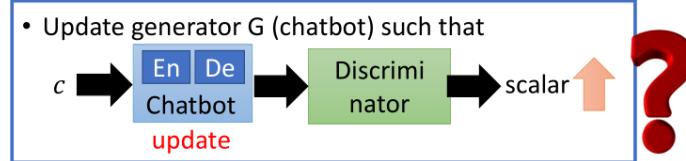
其实这个 discriminator 的 output，就可以想成是人在给 reward，你要把这个 discriminator，想成是一个人，只是这个 discriminator 和人不一样的地方是，它不是完美的，所以要去更新它自己的参数。那整个 algorithm 其实就跟传统的 GAN是一样的，传统 conditional GAN 是一样的。

Algorithm

Training data:

Pairs of conditional input c
and response x

- Initialize generator G (chatbot) and discriminator D
- In each iteration:
 - Sample input c and response x from training set
 - Sample input c' from training set, and generate response \tilde{x} by $G(c')$
 - Update D to increase $D(c, x)$ and decrease $D(c', \tilde{x})$



你有 training data, 这些 training data, 就是一大堆的正确的 c 跟 x 的 pair。

然后你一开始你就 initialize 一个 G, 其实你的 G 就是你的 generator 你的 chatbot, 然后 initialize 你的 discriminator D。

在每一个 training 的 iteration 里面, 你从你的 training data 里面, sample 出正确的 c 跟 x 的 pair。

你从你的 training data 里面 sample 出一个 c prime, 然后把这个 c prime 丢到你的 generator 也就是 chatbot 里面, 让它回一个句子 x tilde, 那这个 c prime, x tilde 就是一个 native 的一个 example。

接下来discriminator 要学着说, 看到正确的 c 跟 x, 给它比较高的分数, 看到错误的 c prime 跟 x tilde, 给它比较低的分数。

至于怎么 train 这个 discriminator, 你可以用传统的方法, 量 js divergence 的方法, 你完全也可以套用 WGAN, 都是没有问题的。

那接下来的问题是说, 我们知道在 GAN 里面你 train discriminator 以后, 接下来你就要 train 你的 chatbot, 也就是 generator。

那 train generator 他的目标是什么呢? 你要 train 你的 generator, 这个 generator 的目标就是要去 update 参数, 然后你 generator 产生出来的 c 跟 x 的 pair, 能让 discriminator 的 output 越大越好, 那这个就是 generator 要做的事情。

这边要做的事情, 跟我们之前看到的 conditional GAN, 其实是一模一样的, 我们说 generator 要做的事情, 其实就是要去骗过 discriminator。

但是这边我们会遇到一个问题。什么样的问题呢? 如果你仔细想一想你的 chatbot 的 network 的架构的话, 我们的 chatbot 的 network 的架构它是一个 seq to seq 的 model, 它是一个 RNN 的 generator。

我们看 chatbot 在 generate 一个 sequence 的时候, 它 generate sequence 的 process 是这样子的。一开始你给它一个 condition, 这个 condition 可能是从 attention based model 来的, 给它一个 condition, 然后它 output 一个 distribution, 那根据这个 distribution 它会去做一个 sample, 就 sample 出一个 token, sample 出一个 word, 然后接下来你会把这个 sample 出来的 word, 当作下一个 time step 的 input, 再产生新的 distribution, 再做 sample, 再当做下一个 time step 的 input, 再产生 distribution。

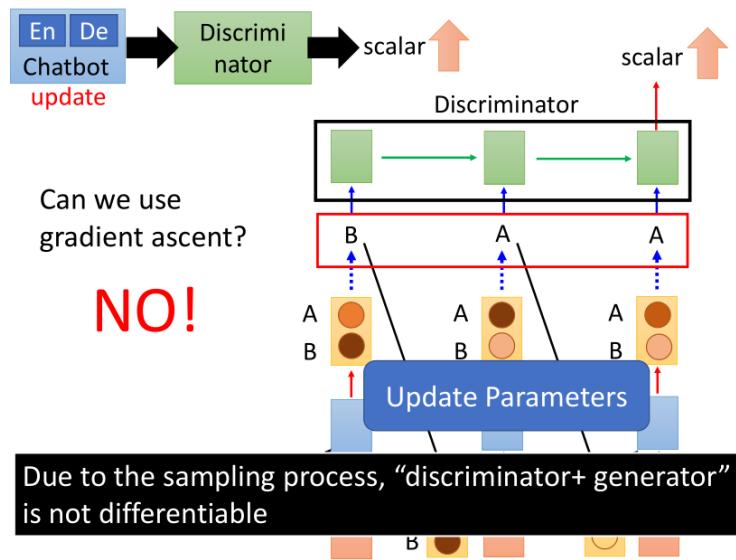
然后我们说我们要把 generator 的 output, 丢给 discriminator, 你对这个 discriminator 的架构, 你也是自己设计, 反正只要可以吃两个 sequence。注意一下这个 discriminator, 前一页的图, 只有画说它吃chatbot 的 output, 但它不能只吃 chatbot 的 output, 它是同时吃 chatbot 的 input 和 output。

在做 conditional GAN 的时候, 你的 discriminator 要同时吃你的 generator 的 input 和 output。所以其实这个 discriminator, 是同时吃了这个 chatbot 的 input 跟 output, 就是两个 word sequence。

那至于这个 discriminator network 架构要长什么样子, 这个就是看你高兴。你可以说你就 learn 一个 RNN, 然后你把 chatbot input 跟 output 把它接起来, 变成一个很长的 sequence。然后 discriminator 把这个很长的 sequence 就读过, 然后就吐出一个数值, 这样也是可以的。有人说我可以用 CNN, 反正只要吃两个 sequence, 可以吐出一个分数, 怎么样都是可以的。

那反正 discriminator 就吃一个 word sequence, 接下来他吐出一个分数。当我们知道说假设我们今天要, train generator 去骗过 discriminator, 我们要做的事情是, update generator 的参数, update 这个 chatbot seq to seq model 的参数, 让 discriminator 的 output 的 scalar 越大越好, 这件事情你仔细想一下, 你有办法做吗? 你想说这个很简单啊, 就是把 generator 跟 discriminator 串起来就变成一个巨大的 network, 然后我们要做的事情就是, 调这个巨大的 network 的前面几个 layer 让, 这个 network 最后的 output 越大越好。

但是你会遇到的问题是, 你发现这个 network 其实是没有办法微分的, 为什么它没有办法微分? 这整个 network 里面有一个 sampling 的 process, 这跟我们之前在讲 image 的时候, 是不一样的。



我觉得这个其实是要用 GAN 来做 natural language processing，跟你用 GAN 来做 image processing 的时候，一个非常不一样的地方。在 image 里面，当你用 GAN 来产生一张影像的时候，你可以直接把产生的影像，丢到 discriminator 里面，所以你可以把 generator 跟 discriminator 合起来，看作是一个巨大的 network。

但是今天在做文字的生成的时候，你生成出一个 sentence，这个 sentence 是一串 sequence，是一串 token，你把这串 token 丢到 discriminator 里面，你要得到这个 token 的时候，这中间有一个 sampling 的 process。

当一整个 network 里面有一个 sampling 的 process 的时候，它是没有办法微分的。一个简单的解释是，你想想看所谓的微分的意思是什么？微分的意思是你把某一个参数小小的变化一下，看它对最后的 output 的影响有多大。这两个相除，就是微分。那今天假设一个 network 里面有 sampling 的 process，你把里面的参数做一下小小的变化，对 output 的影响是不确定的，因为中间有个 sampling 的 process，所以你每次得到的 output 是不一样的。你今天对你整个 network 做一个小小的变化的时候，它对 output 的影响是不确定的，所以你根本就没有办法算微分出来。

另外一个更简单的解释就是，你回去用TensorFlow 或 PyTorch implement 一下，看看如果 network 里面有一个 sampling 的 process，你跑不跑得动这样子，你应该是会得到一个 error，应该是跑不动的，结果就是这样。

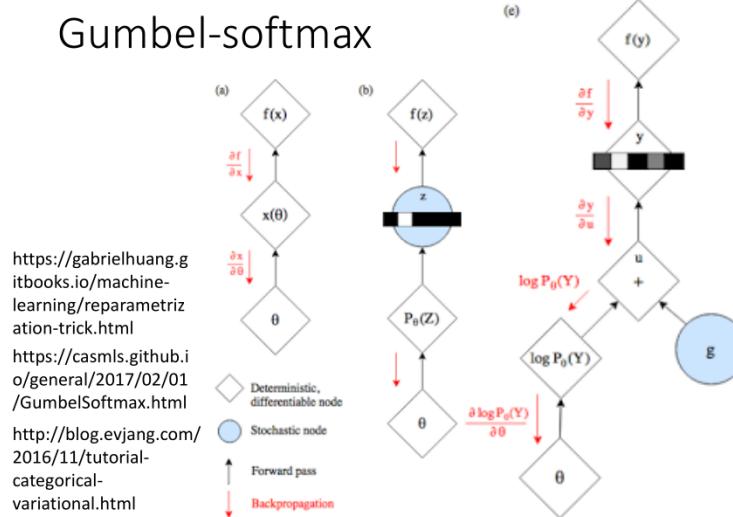
反正无论如何，今天你把这个 seq to seq model，跟你的 discriminator 接起来的时候，你是没有办法微分的。所以接下来真正的难点就是，怎么解这个问题。

Three Categories of Solutions

那我在文献上看到，大概有三类的解法，一个是 Gumbel-softmax，一个是给 discriminator continuous input，另外一种方法就是做 reinforcement learning。

Gumbel-softmax

Gumbel-softmax 我们就不解释，那它其实 implement 也是蛮简单的，但是我发现用在 GAN 上目前没有那么多，所以我们就不解释。总之 Gumbel-softmax 就是想了一个 trick，让本来不能微分的东西，somehow 变成可以微分，如果你有兴趣的话，你再自己研究 Gumbel-softmax 是怎么做的。

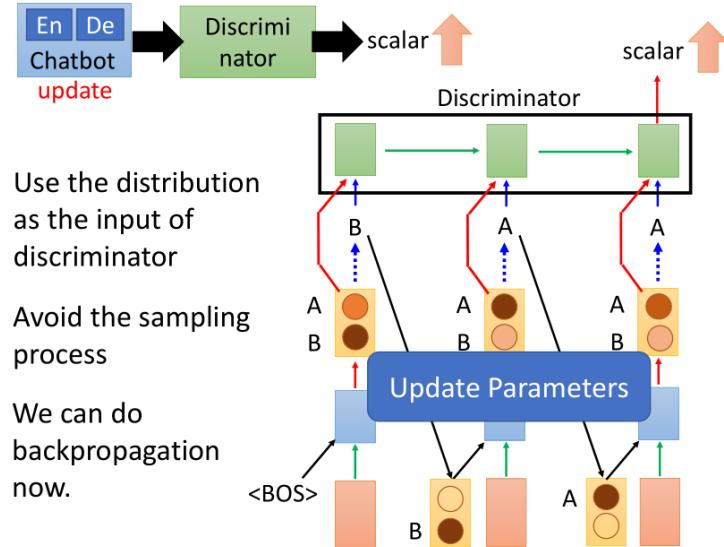


Continuous Input for Discriminator

那另外一个很简单的方法就是，给 discriminator continuous 的 input。

你说今天如果问题是在这一个 sampling 的 process，那我们何不就避开 sampling process 呢。discriminator 不是吃 word sequence，不是吃 discrete token，来得到分数，而是吃 word distribution，来得到分数。

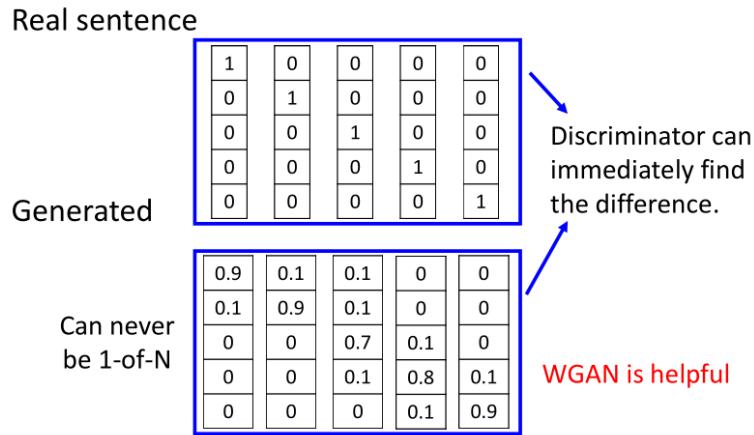
那今天如果我们把这一个 seq to seq model，跟这个 discriminator 串在一起，你就会发现说它变成一个是可以微分的 network 了，因为现在没有那一个 sampling process 了，问题就解决了。



但是实际上问题并没有这么简单，仔细想想看当你今天给你的 discriminator一个 continuous input 的时候，你会发生什么样的问题。

你会发生的问题是这样，Discriminator 会看 real data 跟 fake data，然后去给它一笔新的 data 的时候，它会决定它是 real 还是 fake 的。

当你今天给 discriminator word distribution 的时候，你会发现说，real data 跟 fake data 它在本质上就是不一样的。



因为对 real data 来说，它是 discrete token，或者是说每一个 discrete token，我们其实是用一个 1 one-hot 的 vector 来表示它。对一个 discrete token，我们是用 1 one-hot vector 来表示它。

而对 generator 来说，它每次只会 output 一个 word distribution，它每次 output 的都是一个 distribution。

所以对 discriminator 来说，要分辨今天的 input 是 real 还是 fake 的，太容易了，他完全不需要管这个句子的语义，它完全不管句子的语义，它只要一看说，是不是 one-hot，就知道说它是 real 还是 fake 的。

所以如果你直接用这个方法，来 train GAN 的话，你会发现会遇到什么问题呢？你会发现，generator 很快就会发现说 discriminator，判断一笔 data 是 real 还是 fake 的准则，是看说今天你的每一个 output，是不是 one-hot 的，所以 generator 唯一会学到的事情就是，迅速的变成 one-hot，它会想办法赶快把某一个，随便选一个 element 谁都好，也不要在意语意了，因为就算你考虑语意，也很快会被 discriminator 发现，因为 discriminator 就是要看说是不是 one-hot。

所以今天随便选一个 element，想办法赶快把它的值变到 1，其他都赶快压成 0，然后产生的句子完全不 make sense，然后就结束了。

你会发现所以今天直接让 discriminator，吃 continuous input 是不够的，是没有办法真的解决这个问题。

那其实还有一个解法是，也许用一般的 GAN，train 不起来，但是你可以试试看用 WGAN。为什么在这个 case 用 WGAN，是有希望的呢？

因为 WGAN 在 train 的时候，你会给你的 model 一个 constrain，你要去 constrain 你的 discriminator 一定要是 1-Lipschitz function。因为你有这个 constrain，所以你的 discriminator 它的手脚会被绑住，所以它就没有办法马上分别出 real sentence，跟 generated sentence 的差别。它的视线是比较模糊的，它是比较看不清楚的。因为它有一个 1-Lipschitz function constrain，所以它是比较 fuzzy 的，所以它就没有办法马上分别这两者的差别。

所以今天假设你要做 conditional generation 的时候呢，如果你是要做这种 sequence generation，然后你要用的方法是让 discriminator 吃 continuous input，WGAN 是一个可以的选择。

如果你没有用 WGAN 的话，应该是很难把它做起的，因为 generator 其实学不到语意相关的东西，它只学到说，output 必须要像是 one-hot，才能够骗过 discriminator。

所以这个是第二个 solution，给它 continuous input。

Reinforcement Learning

第三个 solution 呢，就是套用 RL。

我们刚才已经讲过说，假设这个 discriminator，换成一个人的话，你知道怎么去调你 chatbot 的参数，去 maximize 人会给予 chatbot 的 reward。

那今天把人换成 discriminator，solution 其实是一模一样的。怎么解这个问题呢？

也就是说现在呢，discriminator 就是一个 human，我们说人其实就是一个 function 嘛，然后看 chatbot 的 input output 给予分数，所以 discriminator 就是我们的人，它的 output，它的 output 那个 scalar，discriminator output 的那个数值，就是 reward，然后今天你的 chatbot 要去调它的参数，去 maximize discriminator 的 output，也就是说本来人的 output 是 $R(c, x)$ ，那我们只是把它换成 discriminator 的 output $D(c, x)$ ，就结束了。



Consider the output of discriminator as reward

- Update generator to increase discriminator = to get maximum reward
- Using the formulation of policy gradient, replace reward $R(c, x)$ with discriminator output $D(c, x)$

Different from typical RL

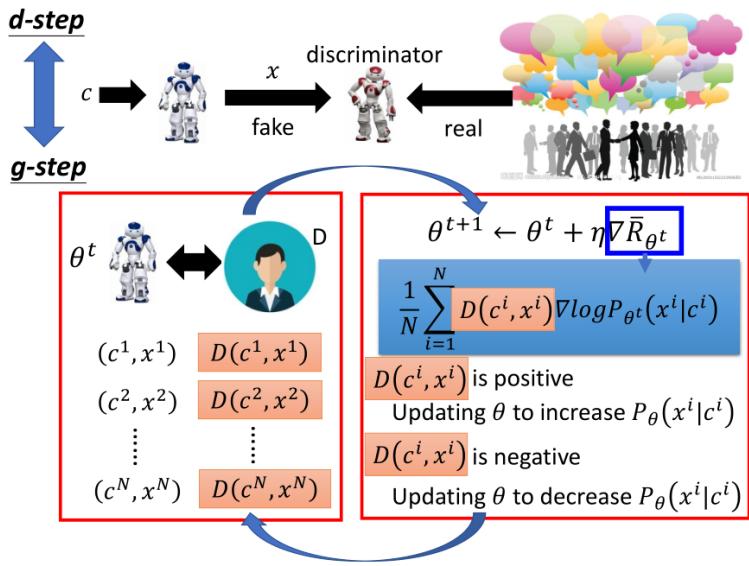
- The discriminator would update

接下来怎么 maximize $D(c, x)$ ，你在 RL 怎么做，在这边就怎么做。

所以呢，我们说在这个 RL 里面是怎么做的呢？你让 θ 去跟人互动，然后得到很多 reward，接下来套右边这个式子，你就可以去 train 你的 model。

现在我们唯一做的事情，是把人呢，换成另外一个机器，就是换成 discriminator，本来是人给 reward，现在换成 discriminator 给 reward。

我们唯一做的事情，就是把 R 换成 D ，所以右边也是一样，把 R 换成 D 。



当然这样跟人互动还是不一样，因为人跟机器互动很花时间嘛，那如果是 discriminator，它要跟 generator 互动多少次，反正都是机器，你就可以让它们真的互动非常多次。

但是这边只完成了 GAN 的其中一个 step 而已，我们知道说在 GAN 的每一个 iteration 里面，你要 train generator，你要 train discriminator 再 train generator，再 train discriminator，再 train generator。

今天这个 RL 的 step 只是 train 了 generator 而已，接下来你还要 train discriminator，怎么 train discriminator 呢？

你就给 discriminator 很多人真正的对话，你给 discriminator 很多，现在你的这个 generator 产生出来的对话，你给 discriminator 很多 generator 产生出来的对话，给很多人的对话，然后 discriminator 就会去学着分辨说这个对话是 real 的，是真正人讲的，还是 generator 产生的。

那其实还有很多的 tip，那这边也稍跟大家讲一下，那如果我们看这个式子的话，你会发现有一个问题，什么样的问题呢？

Reward for Every Generation Step

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{i=1}^N D(c^i, x^i) \nabla \log P_\theta(x^i | c^i)$$

$c^i = \text{"What is your name?"}$	$D(c^i, x^i)$ is negative
$x^i = \text{"I don't know"}$	Update θ to decrease $\log P_\theta(x^i c^i)$
$\log P_\theta(x^i c^i) = \log P(x_1^i c^i) + \log P(x_2^i c^i, x_1^i) + \log P(x_3^i c^i, x_{1:2}^i)$	
$P(\text{"I"} c^i)$?

$c^i = \text{"What is your name?"}$	$D(c^i, x^i)$ is positive
$x^i = \text{"I am John"}$	Update θ to increase $\log P_\theta(x^i c^i)$
$\log P_\theta(x^i c^i) = \log P(x_1^i c^i) + \log P(x_2^i c^i, x_1^i) + \log P(x_3^i c^i, x_{1:2}^i)$	
$P(\text{"I"} c^i)$	↑
	↑
	↑

这个式子跟刚才那个 RL 看到的式子是一样的，我们只是把 R 换成了 D。今天假设 c_i 是 what is your name，然后 x_i 是 I don't know，这可能不是一个很好的回答，所以你得到的 discriminator 给它的分数是负的。当 discriminator 给它的分数是负的时候，我们希望调整我们的参数 θ ，让 $\log P_\theta(x^i | c^i)$ 的值变小，那我们再想想看， $P_\theta(x^i | c^i)$ ，到底是什么样的东西呢？它其实是一大堆 term 的连乘。

也就是说，我们今天实际上在做 generation 的时候，我们每次只会 generate 一个 word 而已。我们假设 I don't know 这边有三个 word，第一个 word 是 x_1 ，第二个 word 是 x_2 ，第三个 word 是 x_3 。那你说让这个机率下降，你希望他们每一项都下降。

但是我们看看 $P(c_i, \text{given } x_1)$ 是什么 is what is your name 的时候，产生 I 的机率，那如果输入 what is your name? 一个好的答案其实可能是比如说 I am John。所以今天问 What is your name 的时候，你其实回答 I 当作句子的开头是好的，但是你在 training 的时候，你却告诉 chatbot 说，看到 What is your name 的时候，回答 I 这个机率，应该是下降的。

看到 What is your name? 你已经产生 I, 产生 don't 的机率要下降, 这项是合理的, 产生 I don't 再产生 know 的机率要下降是合理的, 但是 given What is your name? 产生 I 的机率要下降, 其实是不合理的。

那这个 training 不是有问题吗? 理论上这个 training 不会有问题, 因为今天你的 output, 其实是一个 sampling 的 process, 所以今天在另外一个 case, 当你输入 What is your name 的时候, 机器的回答可能是 I am John, 这个时候机器就会得到一个 positive 的 reward, 也就是 discriminator 会给机器一个 positive 的评价。这个时候 model 要做的事情就是 update 它的参数, 去 increase $\log P_\theta(x^i | c^i)$, 那 $P_\theta(x^i | c^i)$, 是这三个项的相乘, 而第一项是 $P(I | c^i)$, 我们会希望它越大越好, 当你输入 What is your name? sample 到 I don't know 的时候, $P(I | c^i)$ 要减小, 当你 sample 到 I am John 的时候, 你希望这个机率上升, 那如果你今天 sample 的次数够多, 这两项就会抵消, 那就没事了。

但问题就是在实作上, 你永远 sample 不到够多的次数, 所以在实作上这个方法是会造成一些问题的, 所以怎么办呢?

Reward for Every Generation Step

$$\begin{aligned}
 h^i &= \text{"What is your name?"} & x^i &= \text{"I don't know"} \\
 \log P_\theta(x^i | h^i) &= \log P(x_1^i | c^i) + \log P(x_2^i | c^i, x_1^i) + \log P(x_3^i | c^i, x_{1:2}^i) \\
 &\quad \swarrow \qquad \swarrow \qquad \swarrow \\
 P("I" | c^i) && P("don't" | c^i, "I") && P("know" | c^i, "I don't") \\
 \nabla \bar{R}_\theta &\approx \frac{1}{N} \sum_{i=1}^N D(c^i, x^i) \nabla \log P_\theta(x^i | c^i) \\
 \nabla \bar{R}_\theta &\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T (Q(c^i, x_{1:t}^i) - b) \nabla \log P_\theta(x_t^i | c^i, x_{1:t-1}^i)
 \end{aligned}$$

Method 1. Monte Carlo (MC) Search [Yu, et al., AAAI, 2017]

Method 2. Discriminator For Partially Decoded Sequences

[Li, et al., EMNLP, 2017]

今天的 solution 是这个样子, 我们今天希望当输入 What is your name? sample 到 I don't know 的时候, machine 可以自动知道说, 在这三个机率里面, 虽然 I don't know 整体而言是不好的, 但是造成 I don't know 不好的原因, 并不是因为在开头 sample 到了 I, 在开头 sample 到 I, 是没有问题的, 是因为之后你产生了 don't 跟 know, 所以才做得不好。所以希望机器可以自动学到说, 今天这个句子不好, 到底是哪里不好, 是因为产生这两个 word 不好, 而不是产生第一个 word 不好。

那所以你今天会改写你的式子, 现在你给每一个 generation step, 都不同的分数, 今天在给定 condition c_i , 已经产生前 $t-1$ 个 word 的情况下, 产生的 word x_t , 它到底有多好或多不好。

我们换另外一个 measure 叫做 Q , 来取代 D , 这个 Q 它是对每一个 time step 做 evaluation, 它对这边每一次 generation 的 time step 做 evaluation, 而不是对整个句子去做 evaluation。

这件事情要怎么做呢? 你如果想知道的话, 你就自己查一下文献, 那有不同的作法, 这其实是一个还可以尚待研究中的问题。

一个作法就是做 Monte Carlo, 跟 Alpha Go 的方法非常像, 你就想成是在做 Alpha Go, 你去 sample 接下来会发生到的状况, 然后去估测每一个 generation, 每一个 generation 就像是在棋盘上下一个子一样, 可以估测每一个 generation 在棋盘上落一个子的胜率。那这个方法最大的问题就是, 它需要的运算量太大, 所以在实作上你会很难做。

那有另外一个运算量比较小的方法, 这个方法它的缩写叫做 REGS, 不过这个方法, 在文献上看到的结果就是它不如 Monte Carlo, 我自己也有实作过, 觉得它确实不如 Monte Carlo。但 Monte Carlo 的问题是, 它的运算量太大了, 所以这个仍然是一个目前可以研究的问题。

那还有另外一个技术可以 improve 你的 training, 这个方法, 叫做 RankGAN。

Tips: RankGAN

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, Ming-Ting Sun,
"Adversarial Ranking for Language Generation", NIPS 2017

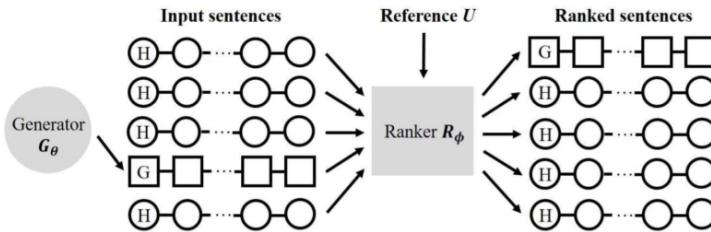


Image caption generation:

Method	BLEU-2	BLEU-3	BLEU-4	Method	Human score
MLE	0.781	0.624	0.589	SeqGAN	3.44
SeqGAN	0.815	0.636	0.587	RankGAN	4.61
RankGAN	0.845	0.668	0.614	Human-written	6.42

那这边是讲一些我们自己的Experimental Results，今天到底把 maximum likelihood, 换到 GAN 的时候，有什么样的不同呢？

事实上如果你有 train 过 chatbot 的话，你会知道说，今天 train 完以后，chatbot 非常喜欢回答一些没有很长，然后非常 general 的句子，通常它的回答就是 I'm sorry, 就是 I don't know, 这样讲来讲去都是那几句。我们用一个 benchmark corpus 叫 Open subtitle 来 train 一个 end to end 的 chatbot 的时候，其实有 1/10 的句子，它都会回答 I don't know 或是 I'm sorry, 这听起来其实是没有非常 make sense。

那如果你要解这个问题，我觉得 GAN 就可以派上用场，为什么今天会回答 I'm sorry 或 I don't know 呢？我的猜测是，这些 I'm sorry 或 I don't know 这些句子，对应到影像上，就是那些模糊的影像。

我们有讲过说，为什么我们今天在做影像生成的时候要用 GAN，而不是传统的 supervised learning 的方法，是因为，今天在做影像的生成的时候，你可能同样的 condition，你有好多不同的对应的 image，比如说火车有很多不同的样子，那机器在学习的时候，它是会产生所有火车的平均，然后看起来是一个模糊的东西。

那今天对一般的 training 来说，假设你没有用 GAN 去 train 一个 chatbot 来说，也是一样的，因为输入同一个句子，在你的 training data 里面，有好多个不同的答案，对 machine 来说他学习的结果就是希望去，同时 maximize 所有不同答案的 likelihood。但是同时 maximize 所有答案的 likelihood 的结果，就是产生一些奇怪的句子。那我认为这就是导致为什么 machine，今天用 end to end 的方法，用 maximum likelihood 的方法，train 完一个 chatbot 以后它特别喜欢说 I'm sorry，或者是 I don't know。

那用 GAN 的话，一个非常明显你可以得到的结果是，用 GAN 来 train 你的 chatbot 以后，他比较喜欢讲长的句子，那它讲的句子会比较有内容，就这件事情算是蛮明显的。

Input	We've got to look for another route.
MLE	I'm sorry.
GAN	You're not going to be here for a while.
Input	You can save him by talking.
MLE	I don't know.
GAN	You know what's going on in there, you know what I mean?

- MLE frequently generates "I'm sorry", "I don't know", etc. (corresponding to fuzzy images?)
- GAN generates longer and more complex responses (however, no strong evidence shows that they are better)

Find more comparison in the survey papers.

[Lu, et al., arXiv, 2018][Zhu, et al., arXiv, 2018]

那一个比较不明显的地方是我们其实不确定说，产生比较长的句子以后，是不是一定就是比较好的对话。

但是蛮明显可以观察到说，当你把原来 MLE 换成 GAN 的时候，它会产生比较长的句子。

More Applications

- Supervised machine translation [Wu, et al., arXiv 2017][Yang, et al., arXiv 2017]
- Supervised abstractive summarization [Liu, et al., AAAI 2018]
- Image/video caption generation [Rakshit Shetty, et al., ICCV 2017][Liang, et al., arXiv 2017]

If you are using seq2seq models,
consider to improve them by GAN.

那其实各种不同的 seq to seq model 都可以用上 GAN 的技术，如果你今天在 train seq to seq model 的时候，你其实可以考虑加上 GAN，看看 train 的会不会比较好。

Unsupervised Conditional Sequence Generation

刚才讲个 conditional sequence generation，那还是 supervised 的，你要有 seq to seq model 的 input 跟 output。接下来要讲 Unsupervised conditional sequence generation。

Text Style Transfer

那我们先讲 Text style transformation，那我们今天已经看过满坑满谷的例子是做image style transformation。

那其实在文字上，你也可以做 style 的 transformation，什么叫做文字的 style 呢？我们可以把正面的句子算做是一种 style，负面的句子算做是另一种 style，接下来你只要 apply cycle GAN 的技术，把两种不同 style 的句子，当作两个 domain，你就可以用 unsupervised 的方法。

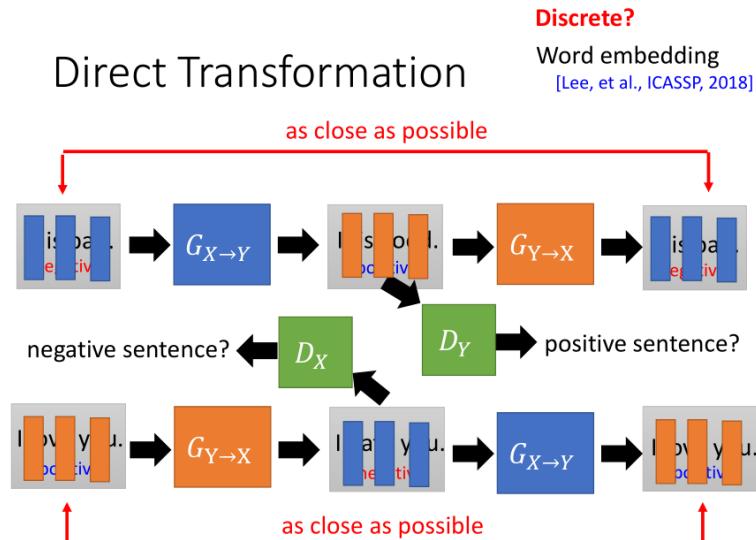
你并不需要两个 domain 的文字句子的 pair，你并不需要知道说这个 positive 的句子应该对应到哪一个 negative 的句子，你不需要这个信息，你只需要两堆句子，一堆 positive，一堆 negative，你就可以直接 train 一个 style transformation。

那我们知道说其实要做这种你要知道，一个句子是不是 positive 的，其实还蛮容易的，因为我们在 ML 的作业 5 里面，你就会 train 一个 RNN，那你就把你 train 过那个 RNN 拿出来，然后给他一堆句子，然后如果很 positive，就放一堆，很 negative 就放一堆，你就自动有 positive 跟 negative 的句子了。那这个技术怎么做呢？

我们不需要多讲，image style transformation 换成 text style transfer，唯一做的事情就是影像换成文字。

所以我们就把 positive 的句子算是一个 domain，negative 的句子算是另外一个 domain，用 cycle GAN 的方法 train 下去就结束了。

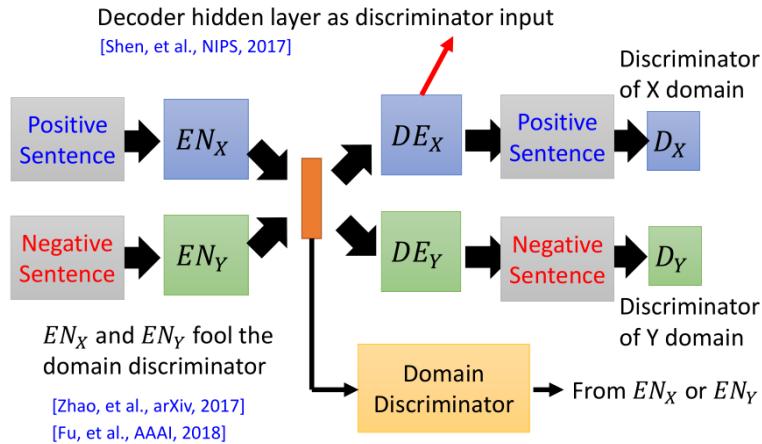
那你这边可能会遇到一个问题是，我们刚才有讲到说，如果今天你的 generator 的 output，是 discrete 的，你没有办法直接做 training，假设你今天你的 generator output 是一个句子，句子是一个 discrete 的东西，你用一个 sampling 的 process，你才能够产生那个句子，当你把这两个 generator，跟这个 discriminator 全部串在一起的时候，你没办法一起 train，那怎么办呢？



有很多不同的解法，我们刚才就讲说有三个解法，一个是用 Gumbel-softmax，一个是给 discriminator continuous 的东西，第三个是用 RL，那就看你爱用哪一种。

在我们的实验里面，我们是用 continuous 的东西，怎么做呢？其实就是把每一个 word，用它的 word embedding 来取代，你把每一个 word，用它的 word embedding 来取代以后。每一个句子，就是一个 vector 的 sequence，那 word embedding 它并不是 one-hot，它是 continuous 的东西，现在你的 generator，是 output continuous 的东西，这个 discriminator 跟这个 generator，就可以吃这个 continuous 的东西，当作 input，所以你只要把 word 换成 word embedding，你就可以解这个 discrete 的问题。

那我们上次讲到说这种 unsupervised 的 transformation 有两个做法，一个就是 cycle GAN 系列的做法，那我们刚才看到哪个 Text style transfer，是用 cycle GAN 系列的做法。那也可以有另外一个系列的做法 Projection to Common Space，就是你把不同 domain 的东西，都 project 到同一个 space，然后再用不同 domain 的 decoder，把它解回来。



Text style transfer 也可以用这样子的做法。你唯一做的事情，就只是把本来你的 x domain 跟 y domain，可能是真人的头像，跟二次元人物的头像，把他们换成正面的句子，跟负面的句子。

当然我们有说，今天如果是产生文字的时候，你会遇到一些特别的问题就是因为，文字是 discrete 的，所以今天这个 discriminator 没有办法吃 discrete 的 input，如果它吃 discrete 的 input 的话，它会没有办法跟 decoder jointly trained，所以怎么解呢？

在文献上我们看过的一个作法是，当然你可以用 RL，Gumbel-softmax 等不同的解法，但我在文献上看到 MIT CSAIL lab 做的一个有趣的解法是，有人说这 discriminator 不要吃 decoder output 的 word，它吃 decoder 的 hidden state，就 decoder 也是一个 RNN 嘛，那 RNN 每一个 time step 就会有一个 hidden vector，这个 decoder 不吃最终的 output，它吃 hidden vector，hidden vector 是 continuous 的，所以就没有那个 discrete 的问题，这是一个解法。

然后我们说这个今天你要让这两个不同的 encoder，可以把不同 domain 的东西 project 到同一个 space，你需要下一些 constrain，我们讲了很多各式各样不同的 constrain，那我发现说那些各式各样不同的 constrain，还没有被 apply 到文字的领域，所以这是一个未来可以做的事情。

我现在看到唯一做的技术只有说有人 train 了一个 classifier，那这个 classifier，就吃这两个 encoder 的 output，那这两个 encoder 要尽量去骗过这个 classifier，这个 classifier 要从这个 vector，判断说这个 vector 是来自于哪一个 domain，我把文献放在这边给大家参考。

Unsupervised Abstractive Summarization

那接下来我要讲的是说，用 GAN 的技术来做，Unsupervised Abstractive summarization。

那怎么 train 一个 summarizer 呢？怎么 train 一个 network 它可以帮你做摘要呢？那所谓做摘要的意思是说，假设你收集到一些文章，那你有没有时间看，你就把那些文章直接丢给 network，希望它读完这个文章以后，自动地帮你生成出摘要。

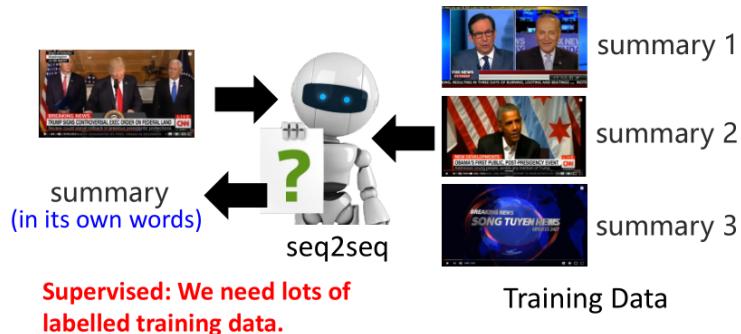
当然做摘要这件事，从来不是一个新的问题，因为这个显然是一个非常有应用价值的东西，所以他从来不是一个新的问题，五六十年前就开始有人在做了，只是在过去的时候，machine learning 的技术还没有那么强，所以过去你要让机器学习做摘要的时候，通常机器学做的事情是 extracted summarization，这边 title 写的是 abstractive summarization，还有另外一种作摘要的方法叫做 extracted summarization，extracted summarization 的意思就是说，给机器一篇文章，那每一篇文章机器做的事情就是判断这篇文章的这个句子，是重要的还是不重要的，接下来他把所有判断为重要的句子接起来，就变成一则摘要了。

那你可能会说用这样的方法，可以产生好的摘要吗？那这种方法虽然很简单，你就是 learn 一个 binary classifier，决定一个句子是重要的还是不重要的，但是你没有办法用这个方法，产生真的非常好的摘要。

为什么呢？你要用自己的话，来写摘要，你不能够把课文里面的句子就直接抄出来，当作摘要，你要自己 understanding 这个课文以后，看懂这个课文以后，用自己的话，来写出摘要。那过去 extracted summarization，做不到这件事，但是今天多数我们都可以做 abstractive summarization。

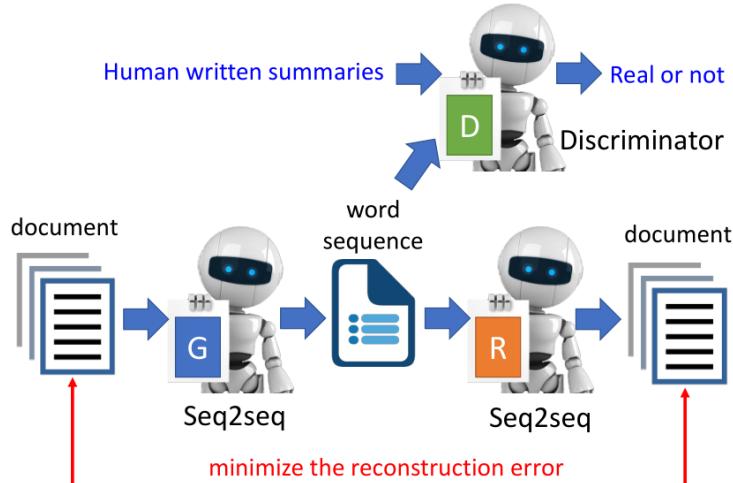
Abstractive Summarization

- Now machine can do **abstractive summary** by seq2seq (write summaries in its own words)



怎么做？learn 一个 seq2seq model，收集一大堆的文章，每一篇文章都有人标的摘要，然后 seq2seq model 硬 train 下去，train 下去就结束了，给它一个新的文章，它就会产生一个摘要，而且这个摘要是机器用自己话说出来的，不见得是文章里面现有的句子。但是这整套技术最大的问题就是，你要 train 这个 seq2seq model，你显然需要非常大量的数据。到底要多少数据才够呢？很多同学会想要自己 train 一个 summarizer，然后他去网络上收集比如说 10 万篇文章，10 万篇文章它通通有标注摘要，他觉得已经很多了，train 下去结果整个坏掉。为什么呢？你要 train 一个 abstractive summarization 系统，通常至少要一百万个 examples 才做得到，没有一百万个 examples，机器可能连产生符合文法的句子都做不到。如果有上百万个 examples，对机器来说，要产生合文法的句子，其实不是一个问题。但是这个 abstractive summarization 最大的问题就是，要收集大量的资料，才有办法去训练。

Unsupervised Abstractive Summarization



所以怎么办呢？我们就想要提出一些新的方法，我们其实可以把文章视为是一种 domain，把摘要视为是另外一种 domain。现在如果我们有了 GAN 的技术，我们可以在两个 domain 间直接用 unsupervised 的方法互转，我们并不需要两个 domain 间的东西的 pair。所以今天假设我们把文章视为一个 domain，摘要视为另外一个 domain，我们不需要文章和摘要的 pair，只要收集一大堆文章，收集一大堆摘要当作范例告诉机器说，摘要到底长什么样子，这些摘要不需要是这些文章的摘要，只要收集两堆 data，机器就可以自动在两个 domain 间互转，你就可以自动地学会怎么做摘要这件事。而这个 process 是 unsupervised 的，你并不需要标注这些文章的摘要，你只需要提供机器一些摘要，作为范例就可以了。那这个技术怎么做的呢？

这个技术就跟 cycle GAN 是非常像的，我们 learn 一个 generator，这个 generator 是一个 seq2seq model。这个 seq2seq model 吃一篇文章，然后 output 一个比较短的 word sequence，但是假设只有这个 generator，你没办法 train，因为 generator 根本不知道说，output 什么样的 word sequence，才能当作 input 的文章的摘要。所以接下来，你就要 learn 一个 discriminator，这个 discriminator 的工作是什么呢？这个 discriminator 的工作就是，他看过很多人写的摘要，这些摘要不需要是这些文章的摘要，他知道人写的摘要是什么样子，接下来他就可以给这个 generator feedback，让 generator output 出来呢 word sequence，看起来像是摘要一样。

就跟我们之前讲说什么风景画转梵高画一样，你需要一个 discriminator，看说一张图是不是梵高的图，把这个信息 feedback 给 generator，generator 就可以产生看起来像是梵高的画作。那这边其实一样，你只需要一个 generator，一个 discriminator，discriminator 给这个 generator feedback，就可以希望它 output 出来的句子，看起来像是 summary。

但是在讲 cycle GAN 的时候我们有讲过说，光是这样的架构是不够的。因为 generator 可能会学到产生看起来像是 summary 的句子，就人写的 summary 可能有某些特征，比如说它都是比较简短的，也许 generator 可以学到产生一个简短的句子，但是跟输入是完全没有关系的。那怎么解这个问题呢？就跟 cycle GAN 一样，你要加一个 reconstructor，在做 cycle GAN 的时候我们说，我们把 x domain 的东西转到 y domain，接下来要 learn 一个 generator，把 y domain 的东西转回来，这样我们就可以迫使，x domain 跟 y domain 的东西，是长得比较像的。我们希望 generator output，跟 input 是有关系的，所以在做 unsupervised abstractive summarization 的时候，我们这边用的概念，跟 cycle GAN 其实是一模一样的。你 learn 另外一个 generator，我们这边称为 reconstructor，他的工作是，吃第一个 generator output 的 word sequence，把这个 word sequence，转回原来的 document，那你在 train 的时候你就希望，原来输入的文章被缩短以后要能被扩写回原来的 document。这个跟 cycle GAN 用的概念是一模一样。

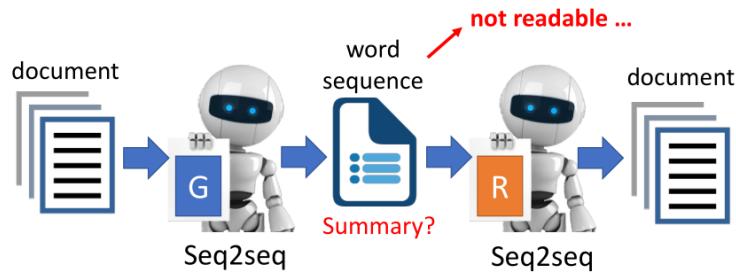
Unsupervised Abstractive Summarization

Only need a lot of documents to train the model



This is a seq2seq2seq auto-encoder.

Using a sequence of words as latent representation.



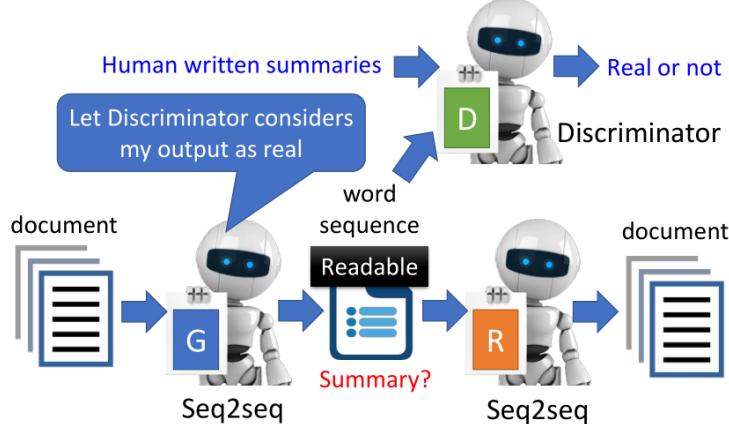
那你其实可以用另外一个方法来理解这个 model，你说我有一个 generator，这个 generator 把文章变成简短的句子，那你有另外一个 reconstructor 它把简短的句子变回原来的文章。如果这个 reconstructor 可以把简短的句子，变回原来的文章，代表说这个句子，有原来的文章里面重要的信息，因为这个句子有原来的文章里面重要的信息，所以你就可以把它当作一个摘要。在 training 的时候，这个 training 的 process 是 unsupervised，因为你只需要文章就好，你只需要输入和输出的文章越接近越好，所以并不需要给机器摘要，你只需要提供给机器文章就好。那这个整个 model，这个 generator 跟 reconstructor 合起来，可以看作是一个 seq2seq2seq auto-encoder，你就一般你 train auto-encoder 就 input 一个东西，把它变成一个 vector，把这个 vector 变回原来的 object，比如说是个 image 等等，那现在是 input 一个 sequence，把它变成一个短的 sequence，再把它解回原来长的 sequence，这样一个 seq2seq2seq auto-encoder。

那一般的 auto-encoder 都是用一个 latent vector 来表示你的信息，那我们现在不是用一个人看不懂的 vector 来表示信息，我们是用一个句子来表示信息，这个东西希望是人可以读的。

但是这边会遇到的问题是，假设你只 train 这个 generator 跟这个 reconstructor，你产生的出来的 word sequence 可能是人没有办法读的，他可能是人根本就没办法看懂的，因为机器可能会自己发明奇怪的暗语，因为 generator 跟 reconstructor，他们都是 machine，所以他们可以发明奇怪的暗语，反正只要他们彼此之间看得懂就好，那人看不懂没有关系，比如说台湾大学，它可能就缩写成湾学，而不是台大，反正只要 reconstructor 可以把湾学解回台湾大学其实就结束了。

Unsupervised Abstractive Summarization

REINFORCE algorithm is used.

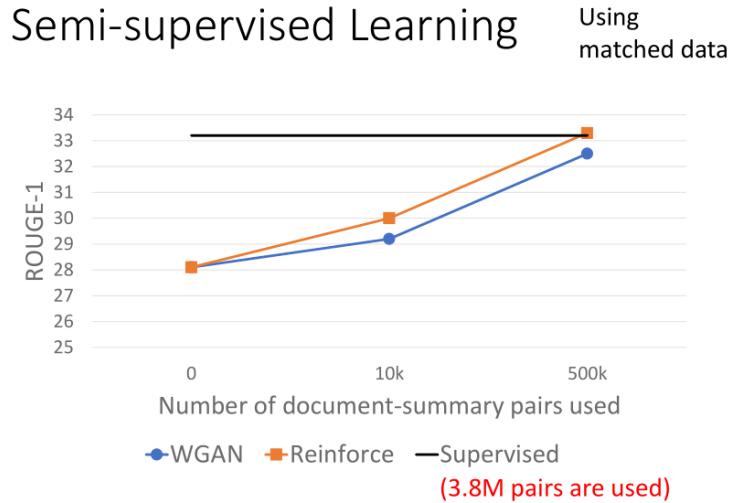


所以为了希望 generator 产生出来的句子是人看得懂的，所以我们要加一个 discriminator，这个 discriminator 就可以强迫说，generator 产生的句子，一方面要是一个 summary 可以对reconstructor 解回原来的文章，同时 generator output 的这个句子，只要是 discriminator 可以看得懂的，觉得像是人类写的 summary。那这个就是 unsupervised abstractive summarization 的架构。

这边可以跟大家讲一下就是说，在 training 的时候，因为这边 output 是 discrete 的嘛，所以你当然是需要有一些方法来处理这种 discrete output，那我们用的就是 reinforced algorithm。

那有人可能会想说用 unsupervised learning 有什么好处，因为你用 unsupervised learning，永远赢不过 supervised learning，supervised learning 就是，unsupervised learning 的 upper bound，unsupervised learning 的意义何在。

那所以我们用这个实验来说明一下 unsupervised learning 的意义。



那这边这个纵轴是 ROUGE 的分数，总之就是用来衡量摘要的一个方法，值越大，代表我们产生的摘要越好。黑色的线是 supervised learning 的方法，今天在做 supervised learning 的时候，需要 380 万笔 training example，380 万篇文章跟它的摘要，你才能够 train 出一个好的 summarization 的系统，是黑色的这一条线。那这边我们用了不同的方法来做这个，来 train 这个 GAN，我们有用 WGAN 的方法，有用 reinforcement learning 的方法，分别是蓝线跟橙线。得到的结果其实是差不多的，WGAN 差一点，用 reinforcement learning 的结果是比较好的，那今天如果在完全没有 label 情况下，得到的结果是这个样子。那当然跟 supervised 的方法，还是差了一截。

但是今天你可以用少量的 summary，再去 fine tune unsupervised learning 的 model，就是你先用 unsupervised learning 的方法把你的 model 练得很强，再用少量的 label data 去 fine tune，那它的进步就会很快。

举例来说，我们这边只用 50 万笔的 data，得到的结果就已经跟 supervised learning 的结果一样了，所以这边你只需要原来的 1/6 或者更少的 data，其实就可以跟用全部的 data 得到一样好的结果。

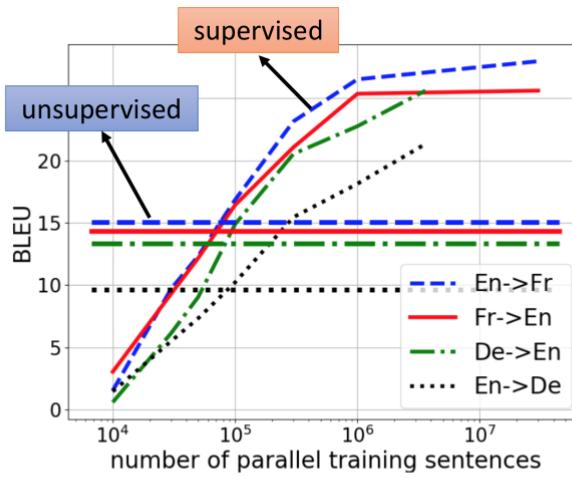
所以 unsupervised learning 带给我们的好处就是，你只需要比较少的 label data，就可以跟过去大量 label data 的时候得到的结果也许是同样好的。那这就是 unsupervised learning 的妙用。

Unsupervised Machine Translation

这边举最后一个例子是 unsupervised machine translation，我们今天可以把不同的语言视为是不同的 domain，就假设你要英文转法文，你就把英文视为一个 domain，法文视为另外一个 domain，然后就可以用 unsupervised learning 的方法把英文转成法文，法文转成英文，做到翻译就结束了，所以你就可以做 unsupervised 的翻译。

那这个方法听起来还蛮匪夷所思的，真的能够做得到吗？其实 facebook 在 ICLR2018 就发了两篇这种 paper，看起来还真的是可以的。

细节我们就不讲，细节你可以想象就很像那个 cycle GAN 这样，只是前面我们有说拿两种不同 image 当作两个不同的 domain，两种不同的语音当作两个不同的 domain，现在只是把两种语言当作两个不同的 domain，然后让机器去学两种语言间的对应，硬做看看做不做的起来。



Unsupervised learning with 10M sentences = **Supervised learning with 100K sentence pairs**

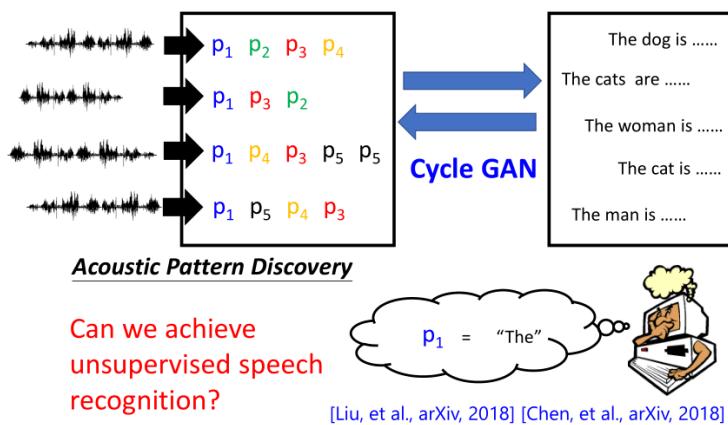
这是文献上的结果，这个虚线代表 supervised learning 的方法，纵轴是 BLEU score，是拿来衡量摘要好坏的方法，BLEU 越高，代表摘要做得越好，横轴是训练资料的量，从 10^4 一直到 10^7 。

如果 supervised learning 的方法，这边是不同语言的翻译，英文转法文，法文转英文，德文转英文，英文转德文，四条线代表四种不同语言的 pair，语言组合间的翻译。

那你说发现训练资料越多，当然结果就越好，这个没有什么特别稀奇的，横线是这个横线是什么？横线用 10^7 的 data 去 train 的 unsupervised learning 的方法，但是你并不需要两个语言间的 pair。做 supervised learning 的时候，你需要两个语言间的 pair。但做 unsupervised learning 的时候，就是两堆句子，不需要他们之间的 pair，得到的结果，只要 unsupervised learning 的方法有 10 million 的 sentences，你的 performance 就可以跟 supervised learning 的方法，只用 10 万笔 data，是一样好的。

所以假设你手上没有 10 万笔 data pair，unsupervised 方法其实还可以赢过 supervised learning 的方法，这个结果是我觉得还颇惊人的。

Unsupervised Speech Recognition



既然两种不同的语言可以做，那语音跟文字间可不可以做呢？把语音视为是一个 domain，把文字视为是另外一个 domain，然后你就可以 apply 类似 GAN 的技术，在这两个 domain 间，互转，这样看看机器能不能够学得起来。如果假设今天机器可以学会说，给它一堆语音给它一堆文字，它就可以自动学会怎么把声音转成文字的话，你就可以做 unsupervised 的语音识别了。未来机器可能在日常生活中，听人讲话，然后它自己再去网络上，看一下人写的文章，就自动学会，语音识别了。

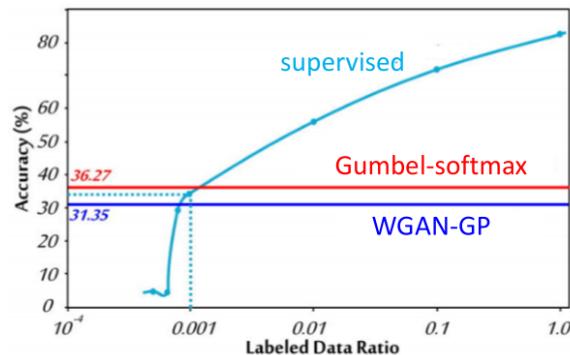
有人可能会想说，这个听起来也是还蛮匪夷所思的，这个东西到底能不能够做到呢？我觉得是有可能的，如果翻译可以做到，这件事情也是有机会的，unsupervised 语音识别也是有机会的。

这边举一个非常简单的例子，假如说所有声音讯号的开头，都是某个样子，比如说都有 P1 这个 pattern，我们用 P 代表一个 pattern，就 P1 这个 pattern，那机器在自己去读文章以后发现说，所有的文章都是 The 开头，它就可以自动 mapping 到说 P1 这种声音讯号，这种声音讯号的 pattern，就是 The 这样，那这是一个过度简化的例子。

Unsupervised Speech Recognition

- Phoneme recognition

Audio: TIMIT
Text: WMT



实际上做不做得起来呢？这个是实际上得到的结果，我们用的声音讯号来自于 TIMIT 这个 corpus，用的文字来自于 WMT 这个 corpus。

那这两个 corpus 是没有对应关系的，一堆语音讲自己的，文字讲自己的，两堆不相关的东西，用类似 cycle GAN 的技术，看能不能够把声音讯号硬是转成文字。

这是一个实验的结果，纵轴是辨识的正确率，那其实是 Phoneme recognition，不是辨识出文字，你是辨识出音标而已，辨识出文字还是比较难，直接辨识出音标而已。

那这个横轴代表说训练资料的量，如果是 supervised learning 的方法，当然训练数据的量越多，performance 越好。这两个横线就是用 unsupervised 的方法硬做得到的结果，那硬做其实有得到 36% 的正确率，你会想 36% 的正确率，这么低，这个 output 结果应该人看不懂吧，是的人看不懂，但是它是远比 random 好的，所以就算是在完全 unsupervised 的情况下，只给机器一堆文字，一堆语音，它还是有学到东西的。

Concluding Remarks

Conditional Sequence Generation

- RL (human feedback)
- GAN (discriminator feedback)

Unsupervised Conditional Sequence Generation

- Text Style Transfer
- Unsupervised Abstractive Summarization
- Unsupervised Translation

GAN Evaluation

这个投影片就是 GAN 的最后要跟大家讲的东西，就是怎么做 Evaluation。

Evaluation 是要做什么？我们要讲的是，怎么 evaluate 用 GAN 产生的 object 的好坏。怎么知道你的 image 是好还是不好。我觉得最准的方法就是人来看，但是在人来看往往不一定是很客观，如果你在看文献上的话，很多 paper 只是秀几张它产生的图，然后加一个 comment 说你看到我今天产生的图，我觉得这应该是我在文献上看过最清楚的图，然后就结束了，你也不知道是真的还是假的。

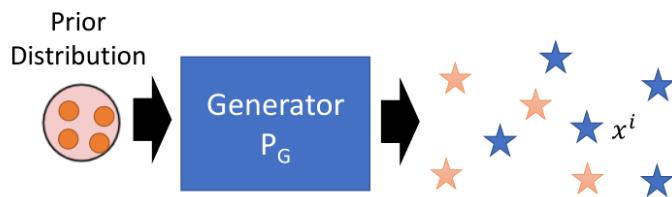
今天要探讨的就是有没有哪一些比较客观的方法，来衡量产生出来的 object 到底是好还是不好。

Likelihood

在传统上怎么衡量一个 generator？传统衡量 generator 的方法是算 generator 产生 data 的 likelihood，也就是说 learn 了一个 generator 以后，接下来给这 generator 一些 real data，假设做 image 生成，已经有一个 image 的生成的 generator，接下来拿一堆 image 出来，这些 image 是在 train generator 的时候 generator 没有看过的 image，然后去计算 generator 产生这些 image 的机率，这个东西叫做 likelihood。

Likelihood

★ : real data (not observed during training)
 ★ : generated data



$$\text{Log Likelihood: } L = \frac{1}{N} \sum_i \log P_G(x^i)$$

We cannot compute $P_G(x^i)$. We can only sample from P_G .

其实是你的 testing data 的 image 的 likelihood 通通算出来做平均，就得到一个 likelihood，这个 likelihood 就代表了 generator 的好坏，因为假设 generator 它有很高的机率产生这些 real data，就代表这个 generator 可能是一个比较好的 generator。

但是如果是 GAN 的话，假设你的 generator 是一个 network 用 GAN train 出来的话，会遇到一个问题就是没有办法计算 $P_G(x)$ ，为什么？

因为 train 完一个 generator 以后，它是一个 network，这个 network 你可以丢一些 vector 进去，让它产生一些 data，但是你无法算出它产生某一笔特定 data 的机率。

它可以产生东西，但你说指定你要产生这张图片的时候，它根本不可能产生你指定出来的图片，所以根本算不出它产生某一张指定图片的机率是多少。所以如果是一个 network 所构成的 generator，要算它的 likelihood 是有困难。

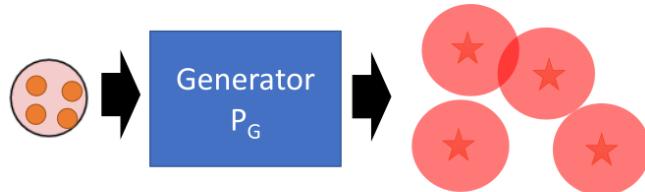
假设这个 generator 不是 network 所构成的，举例来说这个 generator 就是一个 Gaussian Distribution，或是这个 generator 是一个 Gaussian Mixture Model，给它一个 x ，Gaussian Mixture Model 可以推出它产生这个 x 的机率，但是因为那是 Gaussian Mixture Model，它是个比较简单的 model。如果 generator 不是一个简单的 model，是一个复杂的 network，你求不出它产生某一笔 data 的机率。

但是我们又不希望 generator 就只是 Gaussian Mixture Model，我们希望我们的 generator 是一个比较复杂的模型。所以遇到的困难就是如果是一个复杂的模型，我们就不知道怎么去计算 likelihood，不知道怎么计算这个复杂的模型，产生某一笔 data 的机率。

Kernel Density Estimation

怎么办？在文献上一个 solution 叫做，Kernel Density Estimation。

Estimate the distribution of $P_G(x)$ from sampling



Each sample is the mean of a Gaussian with the same covariance.

Now we have an approximation of P_G , so we can compute $P_G(x^i)$ for each real data x^i
 Then we can compute the likelihood.

也就是把你的 generator 拿出来，让你的 generator 产生很多很多的 data，接下来再用一个 Gaussian Distribution 去逼近你产生的 data。什么意思？

假设有一个 generator 你让它产生一大堆的 vector 出来，假设做 Image Generation 的话，产生出来的 image 就是 high dimensional 的 vector，你用你的 generator 产生一堆 vector 出来，接下来把这些 vector 当作 Gaussian Mixture Model 的 mean，然后每一个 mean 它有一个固定的 variance，然后再把这些 Gaussian 通通都叠在一起，就得到了一个 Gaussian Mixture Model。有了这个 Gaussian Mixture Model 以后，你就可以去计算这个 Gaussian Mixture Model 产生那些 real data 的机率，就可以估测出这个 generator 它产生出那些 real

data 的 likelihood 是多少。

我们现在要做的事情是，我们先让 generator 先生一大堆的 data，然后再用 Gaussian 去 fit generator 的 output，到底要几个 Gaussian？32 个吗？64 个吗？还是一个点一个？问题是不知道，所以这就是一个难题。

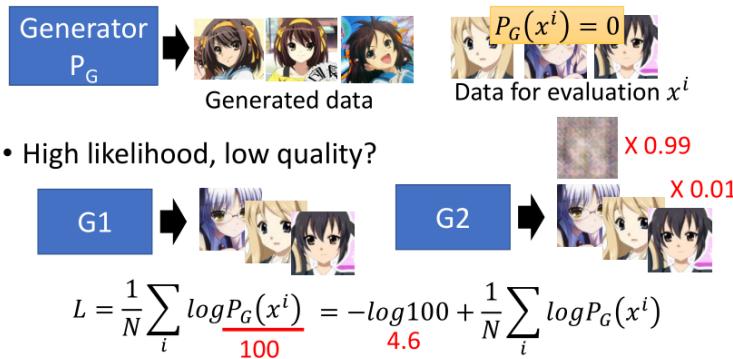
而且另外一个难题是你不知道 generator 应该要 sample 多少的点，才估的准它的 distribution，要 sample 600 个点还是 60,000 个点你不知道。所以这招在实作上也是有问题的。在文献上你会看到有人做这招，就会出现一些怪怪的结果，举例来说，你可能会发现你的 model 算出来的 likelihood 比 real data 还要大。总之这个方法也是怪怪的，因为里面问题太多了，你不知道要 sample 几个点，然后你不知道要怎么估测 Gaussian 的 Mixture，有太多的问题在里面了。

Likelihood v.s. Quality

还有接下来还有更糟的问题，我们就算退一步讲说你真的想出了一个方法，可以计算 likelihood，likelihood 本身也未必代表 generator 的 quality。

- Low likelihood, high quality?

Considering a model generating good images (small variance)



为什么这么说？因为有可能第一个 likelihood 确有高的 quality，举例来说有一个 generator，它很厉害，它产生出来的图都非常的清晰。

所谓 likelihood 的意思是计算这个 generator，产生某张图片的机率，也许这个 generator 虽然它产生的图很清晰，但它产生出来都是凉宫春日的头像而已。如果是其它人物的头像，它从来不会生成，但是 testing data 就是其它人物的头像。所以如果是用 likelihood 的话，likelihood 很小，因为它从来不会产生这些图，所以 likelihood 很小。但是又不能说它做得不好，它其实做得很好，它产生的图是 high quality 的，只是算出来 likelihood 很小。所以 likelihood 并不代表 quality，它们俩者是不等价。

反过来说，高的 likelihood 也并不代表你产生的图就一定很好，你有一个 model 它 likelihood 很高，它仍然有可能产生的图很糟，怎么说？

这边举一个例子，里面有一个 generator 1，generator 1 很厉害，它的 likelihood 很大，假设我们不知道怎么回事，somehow 想了一个方法可以估测 likelihood，虽然之前我们在前期的投影片已经告诉你，估测 likelihood 也是很麻烦，不知道怎么做，现在 somehow 想了一个方法可以估测 likelihood。现在有个很强的 generator，它的 likelihood 是大 L，它产生这些图片的机率很高，现在有另外一个 generator，generator 2 它有 99% 的机率产生 random noise，它有 1% 的机率，它做的事情跟 generator 1 一样。如果我们今天计算 generator 2 的 likelihood，generator 2 它产生每一张图片的机率是 generator 1 的 1/100。假设 generator 1 产生某张图片 x_i 的机率是 $P_G(x_i)$ ，generator 2 产生那张图片的机率，就是 $P_G(x_i) * 100$ ，因为 generator 2 有两个 mode，它有 99% 的机率会整个坏掉，但它有 1% 的机率会跟 generator 1 一样，所以 generator 1 产生某张图片的机率如果是 P_G ，那 generator 产生某张图片的机率就是 $P_G / 100$ 。

现在问题来了，假设把这个 likelihood 每一项都除以 100，你会发现你算出来的值，也差不了多少，因为除一百这项把它提出来，就是 $-\log(100)$ ，才减 4.65 而已，如果看文献 likelihood 算出来都几百，差了 4 你可能会觉得没什么差别。

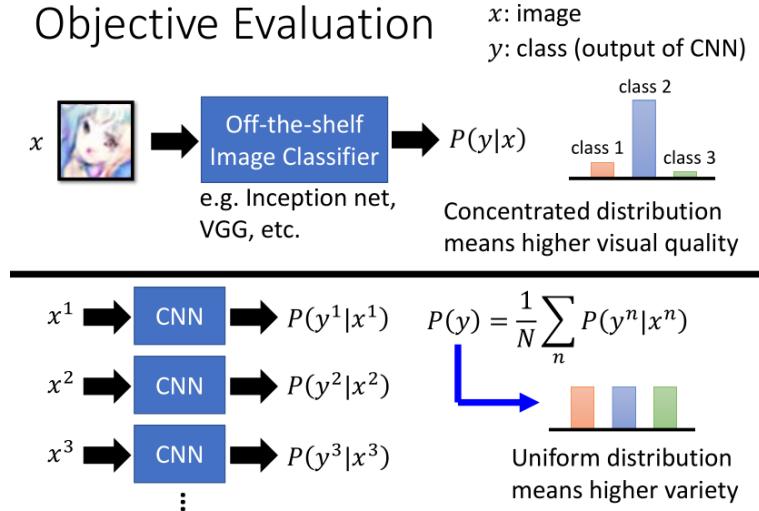
但是如果看实际上的 generator 2 跟 generator 1 比的话，generator 1 你会觉得它应该是比 generator 2 好一百倍的，只是你看不出来而已，数字上看不出来。

所以 likelihood 跟 generator 真正的能力其实也未必是有关系的。

Objective Evaluation

今天这个文献上你常常看到一种 Evaluation 的方法，常常看到一种客观的 Evaluation 的方法，是拿一个已经 train 好的 classifier 来评价现在，产生出来的 object。

Objective Evaluation



假设要产生出来的 object 是影像的话，你就拿一个影像的 classifier 来判断这个 object 的好坏，就好像我们是拿一个人脸的辨识系统，来看你产生的图片，这个人脸辨识系统能不能够辨识的出来，如果可以就代表你产生出来的是还可以的，如果不行就代表你产生出来的真的很弱。

今天这个道理是一样的，假设你要分辨机器产生出来的一张影像好还是不好，你就拿一个 Image Classifier 出来，这 Image Classifier 通常已经是事先 train 好的，举例来说它是个 VGG，它是个 Inception Net。

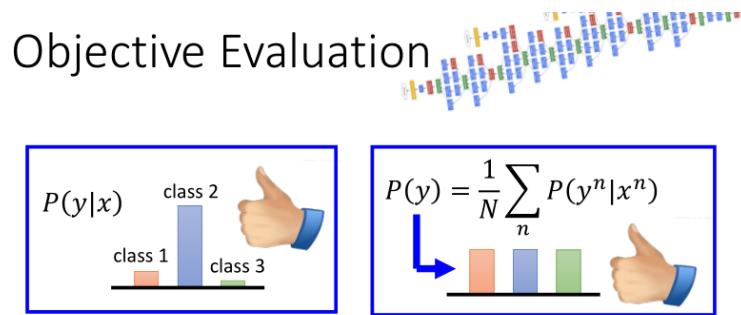
把这个 Image Classifier 丢一张机器产生的 image 给它，它会产生一个 class 的 distribution，它给每一个 class 一个机率，如果产生的机率越集中，代表产生的图片的质量越高。因为这个 classifier 它可以轻易的判断，现在这个图片它是什么样的东西。所以它给某一个 class 机率特别高，代表产生的图片是这个 model 看得懂的。

但这个只是一个衡量的方向而已，你同时还要衡量另外一件事情，因为我们知道在 train GAN 会遇到一个问题就是，**Mode collapse** 的问题，你的机器可能可以产生某张很清晰的图，但它就只能产生那张图而已，这个不是我们要的。

所以在 evaluate GAN 的时候还要从另外一个方向，还要从 diverse 的方向去衡量它，什么叫从 diverse 的方向去衡量它呢？你让你的机器产生一把，这边举例就产生三张，把这三张图通通丢到 CNN 里面，让它产生三个 distribution，接下来把这三个 distribution 平均起来，如果平均后的 distribution 很 uniform 的话，这个 distribution 平均完以后，它仍然很平均的话，那就意味着每一种不同的 class 都有被产生到，代表产生的 output 是比较 diverse。如果平均完发现某一个 class 分数特别高，就代表它的 output，你的 model 倾向于产生某个 class 的东西，就代表它产生的 output 不够 diverse。

所以我们可以从两个不同的方向，用某一个事先 train 好的 Image Classifier 来衡量 image，可以只给它一张图，然后看产生的图清不清楚，接下来给它一把图，看看是不是各种不同的 class，都有产生到。

Inception Score



Inception Score

$$\begin{aligned}
 &= \sum_x \sum_y P(y|x) \log P(y|x) \quad \text{Negative entropy of } P(y|x) \\
 &\quad - \sum_y P(y) \log P(y) \quad \text{Entropy of } P(y)
 \end{aligned}$$

有了这些原则以后，就可以定出一个 Score，现在一个常用的 Score，叫做 Inception Score。

那至于为什么叫做 Inception Score，当然是因为它用 Inception Net 去 evaluate，所以叫做 Inception Score。

我们之前有讲怎样的 generator 叫做好，好的 generator 它产生的单一的图片，丢到 Inception Net 里面，某一个 class 的分数越大越好，它是非常的 sharp。把所有的 output 都掉到 classifier 里面，产生一堆 distribution，把所有 distribution 做平均，它是越平滑越好。

根据这两者就定一个 Inception Score，把这两件事考虑进去，在 Inception Score 里面第一项要考虑的是，summation over 所有产生出来的 x ，每一个 x 丢到 classifier 去算它的 distribution，然后就计算 Negative entropy。Negative entropy 就是拿来衡量这个 distribution 够不够 sharp，每一张 image 它 output 的 distribution 越 sharp 的话，就代表产生的图越好。同时要衡量另外一项，另外一项就是把所有的 distribution 平均起来，如果平均的结果它的 entropy 越大也代表越好。同时衡量这两项，把这两项加起来，就是 Inception Score。

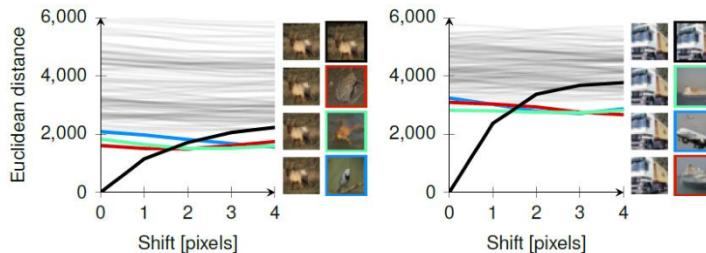
其实还有其它衡量的方法，但一个客观的方法就是拿一个现成的 model 来衡量你的 generator。

We don't want memory GAN.

还有另外一个 train GAN 要注意的问题，有时候就算 train 出来的结果非常的清晰，也并不代表你的结果是好的，为什么？因为有可能 generator 只是硬记了 training data 里面的，某几张 image 而已。这不是我们要的，因为假设 generator 要硬记 image 的话，那直接从 database sample 一张图不是更好吗？干嘛还要 train 一个 generator。

所以我们希望 generator 它是有创造力的，它产生出来的东西不要是 database 里面本来就已经现成的东西。但是怎么知道现在 GAN 产生出来的东西，是不是 database 已经现存的东西呢？这是另外一个 issue，因为没有办法把 database 里面每张图片都一个一个去看。database 里面图片有上万张，根本没办法一张一张看过，所以根本不知道 generator 产生出来的东西是不是 database 里面的。

- Using k-nearest neighbor to check whether the generator generates new objects



GAN 产生一张图片的时候，就把这张图片拿去跟 database 里面每张图片都算 L1 或 L2 的相似度，但光算 L1 或 L2 的相似度是不够的，为什么？以下是文献上举的一个例子，这个例子是想要告诉大家，光算相似度，尤其是只算那种 pixel level 的相似度，是非常不够的。为什么这么说？这个例子是这样，假设有一只羊的图，这个羊的图跟谁最像，当然是跟自己最像，跟 database 里面一模一样的那张图最像。黑色这条线代表的是羊这张图片，羊这张图片跟自己的距离当然是 0，跟其它图片的距离是比较大的，这边每一条横线就代表一张图片。把羊那张图的 pixel 都往左边移一格，还是跟自己最像。但是如果往左边移两格，会发现最像的图片，就变成红色这张，移三格就变绿色这张，移四格就变蓝色的这张。假设 generator 学到怪的东西就是，把所有的 pixel 都往左移两格，这个时候就算它 copy 了 database 你也看不出来。因为检测的方法检测不出这个 case。右边也是一样，把卡车的图片往左移一个 pixel 跟自己最像，移三个 pixel 就变跟飞机最像，移四个 pixel 就变跟船最像。

因为很难算两张图片的相似度，所以 GAN 产生一个图片的时候，你很难知道它是不是 copy 了 database 里面的 specific 的某一张图片，这个也都是尚待解决的问题。

所以有时候 GAN 产生出来结果很好，也不用太得意，因为它搞不好只是 copy 某一张图片而已。

Mode Dropping

Mode Collapse 的意思是说你的 real data 的 distribution 是比较大的，但是你 generate 出来的 example，它的 distribution 非常的小。

Mode dropping 意思是说你的 distribution 其实有很多个 mode，假设你 real distribution 是两群，但是你的 generator 只会产生同一群而已，他没有办法产生两群不同的 data。

假设 GAN 产生出来的是人脸的话，它产生人脸的多样性不够。

怎么检测它产生出来的东西它的多样性够不够，假设 train 了一个 DCGAN，DCGAN 是 Deep Convolutional GAN 的缩写，它的 training 的方法跟 Ian Goodfellow 一开始提出来的办法是一样的，只是在 DCGAN 里面那个作者爆搜了各种不同的参数，然后告诉你怎么样 train GAN 的时候结果会比较好，有不同 network 的架构，不同的 Activation Function，有没有加 batch，各种方法都爆搜一遍，然后告诉你怎么样做比较好。

怎么知道 DCGAN，train 一个产生人脸的 DCGAN，它产生的人脸的多样性是够的呢？一个检测方法是从 DCGAN 里面 sample 一堆 image，叫 DCGAN 产生一堆 image，然后确认产生出来的 image 里面有没非常像的，有没有人会觉得是同一个人。

怎么知道是不是同一个人，结果来自于 ICLR 2018 叫 "Do GANs learn the distribution?"，里面的做法是让机器产生一堆的图片，接下来先用 classifier 决定有没有两张图片看起来很像，再把长的很像的图片拿给人看，问人说：你觉得这两个是不是同一个人，如果是，就代表 DCGAN 产生重复的图了，虽然产生图片每张都略有不同，但人可以看出这个看起来算不算是同一个人。