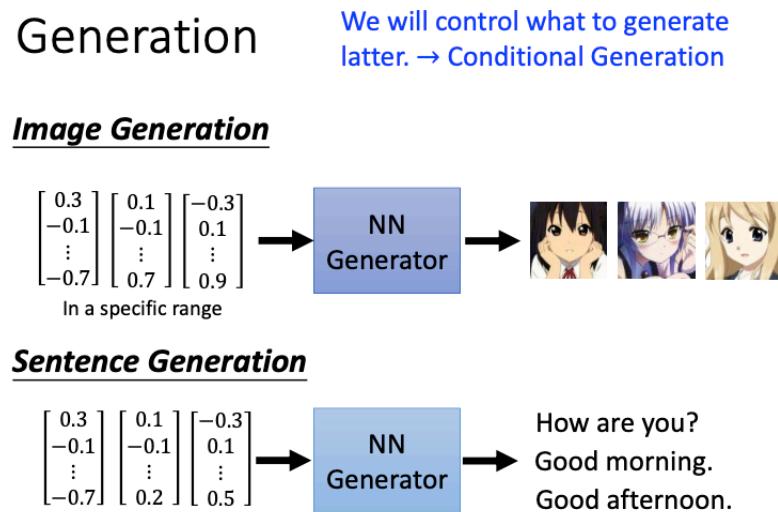


有很多种不同类型GAN，可以在这里查看<https://github.com/hindupuravinash/the-gan-zoo>

Generation

对于Image generation，要实现的是输入一个vector，输出一个image；而对于Sentence generation，则实现的是输入一个vector，输出为一个sentence。那么GAN就是用来实现这个中间的NN Generator。

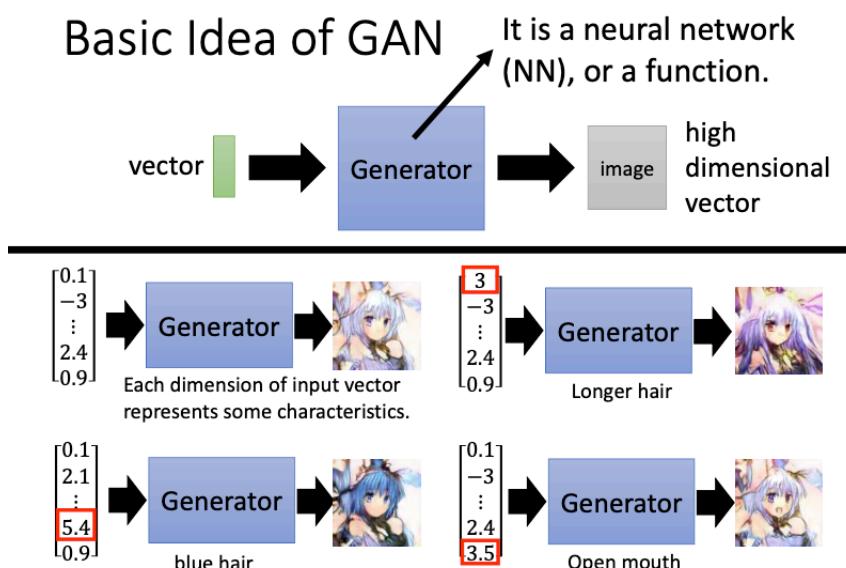


Basic Idea of GAN

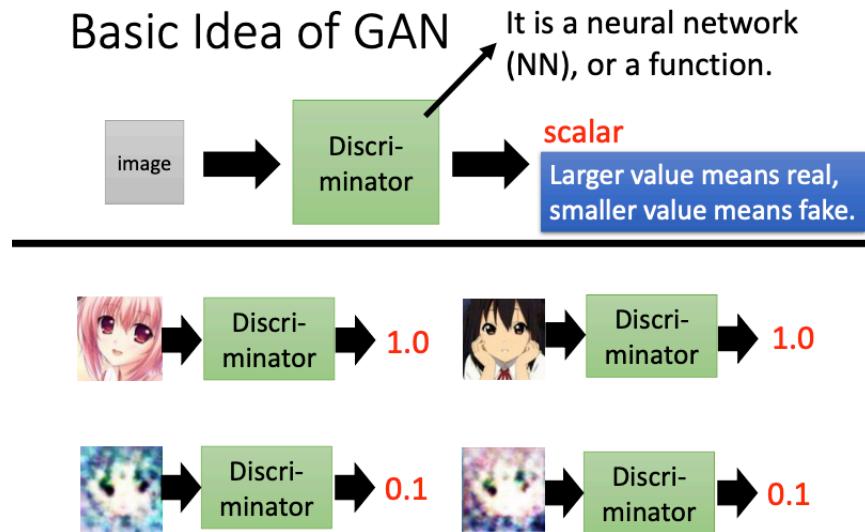
Generator vs Discriminator

这个generator可以是一个神经网络，也可以是一个函数f。

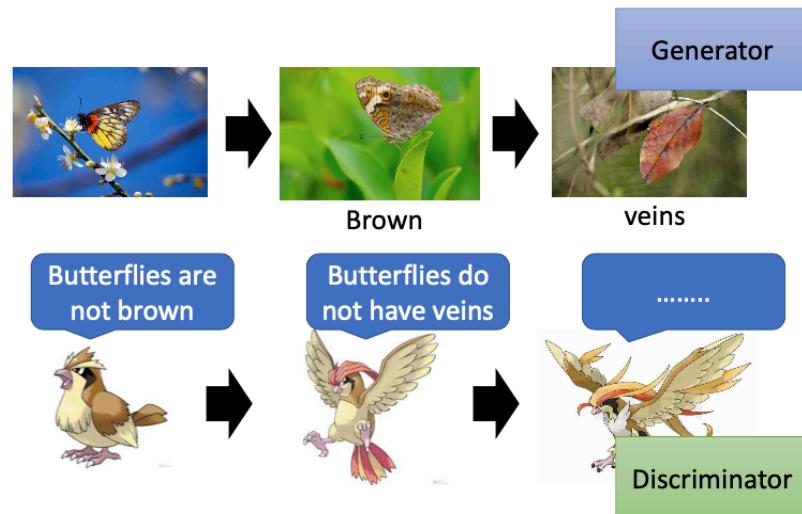
输入的vector表示我们要generate图像的某种特征，比如vector的第一维如果代表头发的长度，我们现在把这个值设得很大，那么就会generate一张头发很长的图像；如果我们改变了vector倒数第二维（头发为蓝色）的值，可以发现generate的新图头发变蓝了，由于此时只改变了头发的颜色，其他特征都是类似的，所以只有头发的颜色发生了变化。



Discriminator可以是一个神经网络，也可以是一个函数f。输入是一张image，输出为scalar，数值越大，表示这个image越接近真实图像。



在大自然中，某种鸟类以枯叶蝶为食。枯叶蝶必须不断地进化，使其看起来越来越像一个枯叶；它的天敌也在不断地进化，如果枯叶蝶看起来并不像叶子，那么它就会被捕食。其中枯叶蝶就相当于一个generator，天敌就相当于discriminator。



第一代的generator不知道怎样产生二次元的头像，所以会产生一些看起来很像是杂讯的图像，再把这个图像输入discriminator，来判断这是不是一张真实的图像，第一代的discriminator可以根据图像是否有颜色，来正确分辨真实图像和生成的图像；

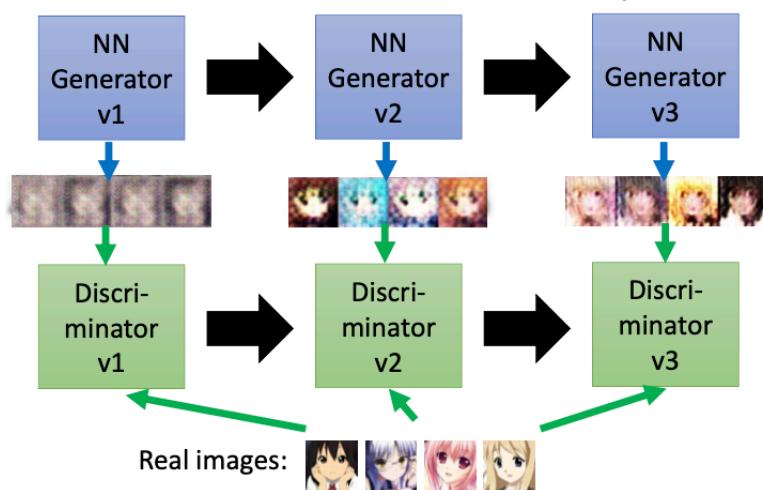
那么第二代的generator的目标就是想办法骗过第一代的discriminator，生成了有颜色的图像，随之discriminator也会发生进化，学习了真实图像和生成图像之间的差异（真实图像是有嘴的）；

第三代的generator生成的图像可以骗过第二代的discriminator，然后discriminator也会继续进化，
.....

generator和discriminator都会不断地进化，因此generator会产生越来越像真实图片的图像。

Basic Idea of GAN

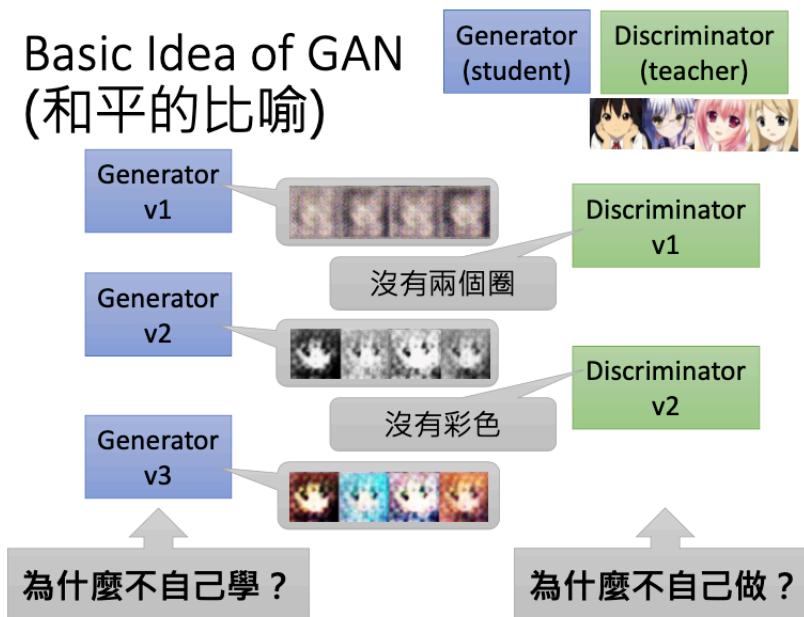
This is where the term
“adversarial” comes from.
 You can explain the process
 in different ways.....



这个过程可以看作是一个对抗的过程，也看一用另外一个和平的比喻来进行解释。generator相当于一个学生，discriminator相当于一个老师，学生并不知道真实的图像长什么样，但老师看过很多真实的图像，就知道真实的图像应该长什么样子。

第一代的generator相当于一年级的学生，重复着上述的过程。学生会画得越来越好，老师会越来越严格，那么学生最后就可以画出很想二次元人物的图像了。

Basic Idea of GAN (和平的比喻)



Algorithm

首先需要随机初始化G和D的参数，

Step 1：我们需要先调整D的参数，就必须先把G的参数固定。首先把随机产生的vector输入G (fix)，生成新的图像之后与从database中sampled出来的图像进行比较。要实现D如果输入真实图像，就会得高分，与1越接近越好，如果输入生成的图像，就会得低分，与0越接近越好。有了这个标准，我们就可以来训练这个discriminator D；

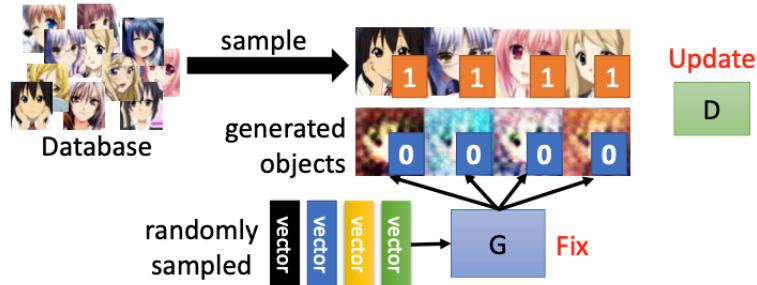
- Initialize generator and discriminator

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.

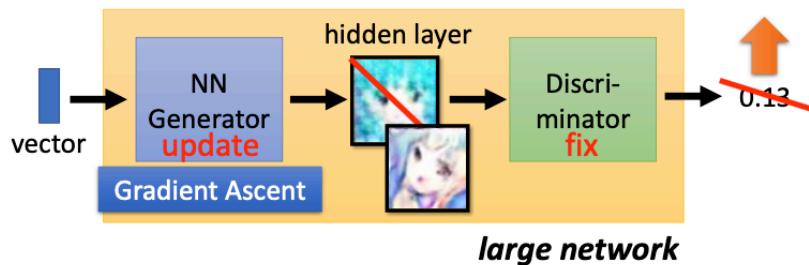
Step2: 训练好discriminator D之后，我们就可以fix D，来调整generator G。一个vector输入第一代G之后，会生成一张图像，再输入D (fix)，就可以得到一个很低的分数 (0.13)。那么G训练的目标就是使生成的图片可以“骗”过D，即生成的图片使D给出一个比较高的分数。由于D看过真实的图像，如果给出了很高的分数，就可以说明G生成的图像和真实图像是很接近的。

在真实的代码实现中，我们通常会把generator和discriminator当成是一个大的network，其中generator的输出就可以看作是一个hidden layer，discriminator所在的层参数是fix的，不用调整，只需要根据整个网络的输出来调整generator的参数。

由于我们希望使discriminator的输出分数值越大越好，因此这里使用了梯度上升算法 **Gradient Ascent**，也就是梯度下降算法前面多乘了一个负号。

Step 2: Fix discriminator D, and update generator G

Generator learns to “fool” the discriminator



现在来叙述一下总的算法流程， θ_d, θ_g 分别表示discriminator和generator的参数。

Learning D: 首先从数据库中取出m个真实图片，再根据一个分布随机产生m个vector作为输入 $\{z^1, z^2, \dots, z^m\}$ ，此时fix G的参数，得到G生成的图像 $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$ ，再输入discriminator D，不断调整 θ_d ，使得得到的分数越大越好，

$$\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(x^i) + \frac{1}{m} \sum_{i=1}^m \log(1 - D(\tilde{x}^i))$$

其中 $D(x^i)$ 表示真实图像所得到的分数，D的目标就是使真实图像获得的分数越大越好；而 $D(\tilde{x}^i)$ 表示G生成的图像所得到的分数，应该越小越好，所以前面加了负号。为了方便求梯度，在式子前面介入了 \log ，求出梯度 $\Delta \tilde{V}(\theta_d)$ ，再更新 θ_d 的值，

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

Algorithm Initialize θ_d for D and θ_g for G

Learning
D

- 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
G

Learning G: 把D训练好之后，我们就可以fix D，来训练generator G的参数。首先也需要从分布中随机生成一些噪声z，再输入G， $G(z^i)$ 再输入D，得到相对应的分数，G的目标是想办法骗过D，不断调整参数 θ_g ，使生成的图像所得到的分数越高越好，

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

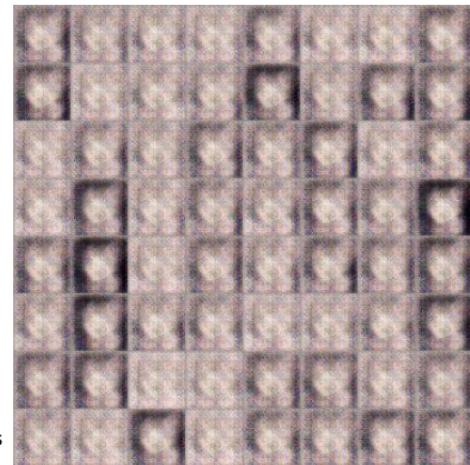
求出梯度 $\Delta \tilde{V}(\theta_g)$ ，再更新 θ_g 的值，

$$\theta_g \leftarrow \theta_g + \eta \Delta \tilde{V}(\theta_g)$$

在每个iteration里，都会进行这个步骤，先训练discriminator，再训练generator；这两个步骤会反复进行。

Anime Face Generation

结果展示



100 updates



1000 updates



2000 updates



20,000 updates

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)

<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr></table>	1	0	0	<table border="1"><tr><td>0</td><td>1</td><td>0</td></tr></table>	0	1	0	<table border="1"><tr><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1
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

Output Sequence $f : X \rightarrow Y$

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

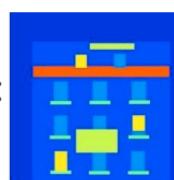
X : 
(speech) Y : 感謝大家來上課”
(transcription)

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

X :  Y : 

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

Colorization:



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, 如果有的类别都没有范例，或者只有很少一部分的范例。

而structured learning是一种极端的One-shot learning, 由于output为一个structure, 比如一个句子, 可能这些句子在training data中从来没出现过, 那么如何学习去输出一个从来没看到的structure, machine必须学会去创造。

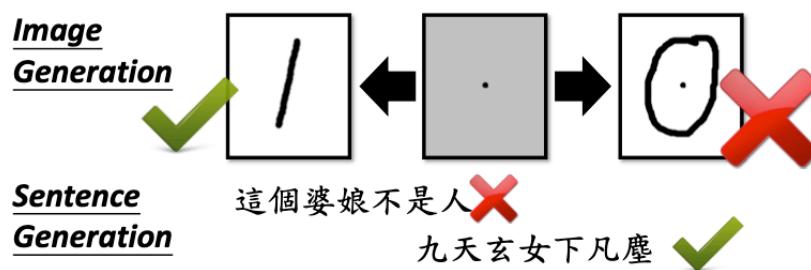
- **One-shot/Zero-shot Learning:**

- 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还必须学会如何去planing, 有全局观; 比如sentence generation中, 如果只看第一句话, 会认为是负面的, 但如果你把整句话都看完, 就会发现这整句话在表达一个正面的意思。

- Machine has to learn to do **planning**

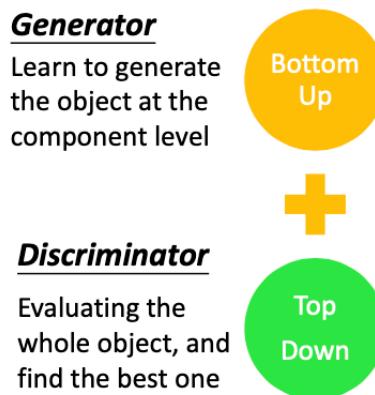
- Machine generates objects component-by-component, but it should have a big picture in its mind.
- Because the output components have dependency, they should be considered globally.



Structured Learning Approach

structured learning有两套方法:

- Bottom up, 机器在生成一个部件时, 会先生成多个component, 这种方法一个很大的问题就是容易失去大局观;
- Top down, 产生一个完整的物件之后, 再去从整体上看产生物件好不好。

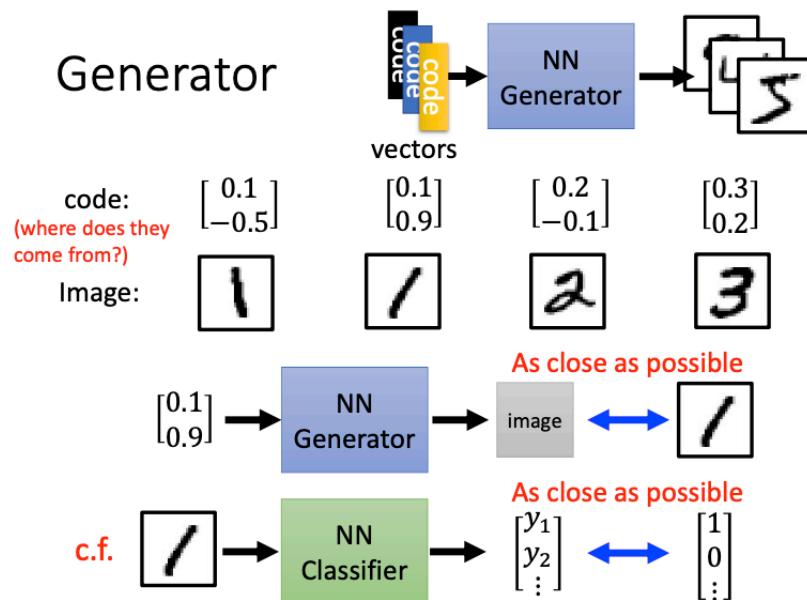


把这两种方法结合起来就是Generator。

Can Generator learn by itself?

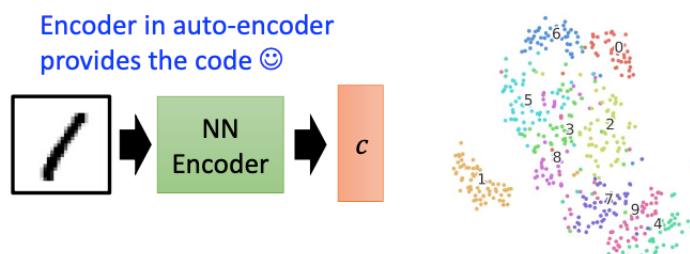
Generator

对于Generator，首先输入不同的vector，就可以输出不同的图片。如果我们现在输入1对应的vector，generator会生成一张image，目标是使image和真实的图像越接近越好，这个真实图像现在generator能看到，那么这不就和一般的supervised learning一摸一样了吗？



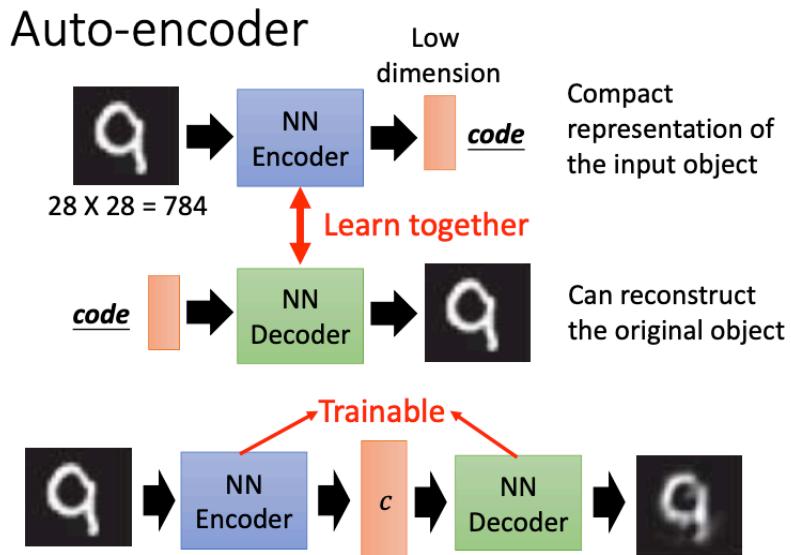
那么我们怎么知道输入的那些vector的数值呢？

我们可以用一个Encoder来表示，把image输入这个NN Encoder，就会输出对应的特征，把图像的特征用vector来表示即可。

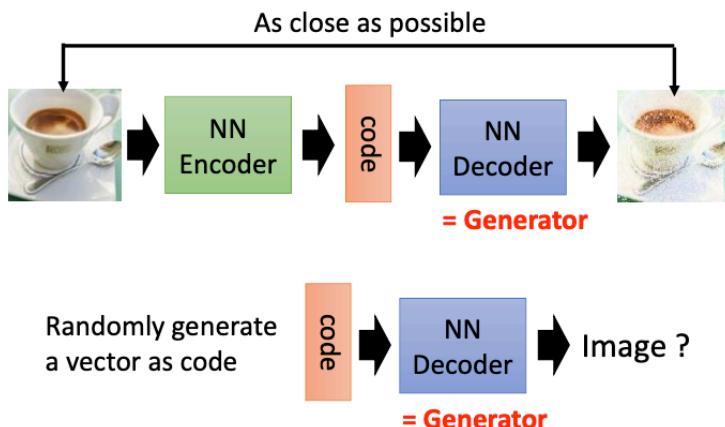


Auto-encoder

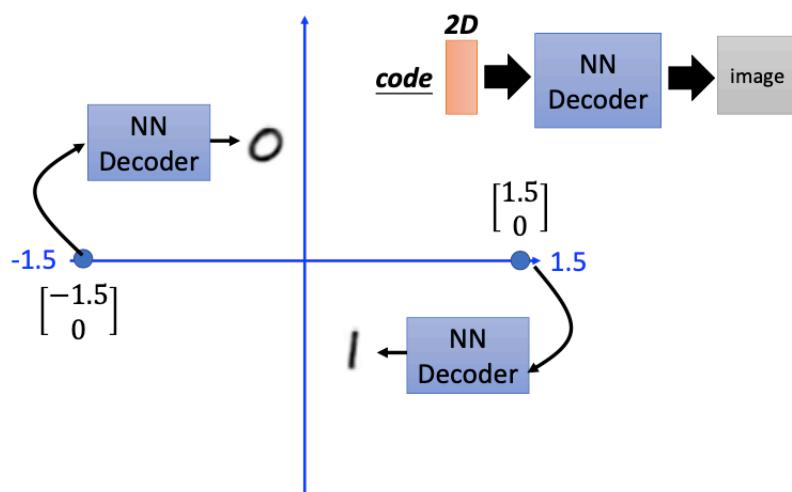
Auto-encoder分为encoder和decoder。对于输入的 28×28 图像，先用encoder使得输入的图像变成code，decoder把这个code再恢复成原来的图像，这两者会一起进行学习。



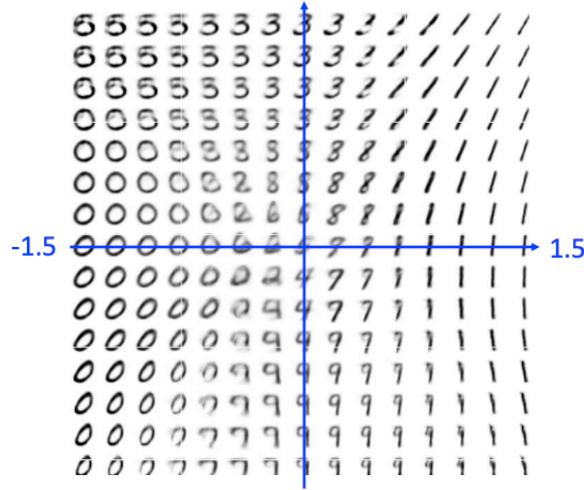
目标是使得input和output越接近越好。这里的decoder就相当于一个generator，我们可以随机输入一个vector，使decoder (generator) 生成一张对应的图像。



这里的code可以训练成二维的。如果code是 $(-1.5 \ 0)^T$, 输入decoder会生成对应的图像, 即图中的0; 如果code是 $(1.5 \ 0)^T$, 输入decoder会生成对应的图像, 即图中的1.



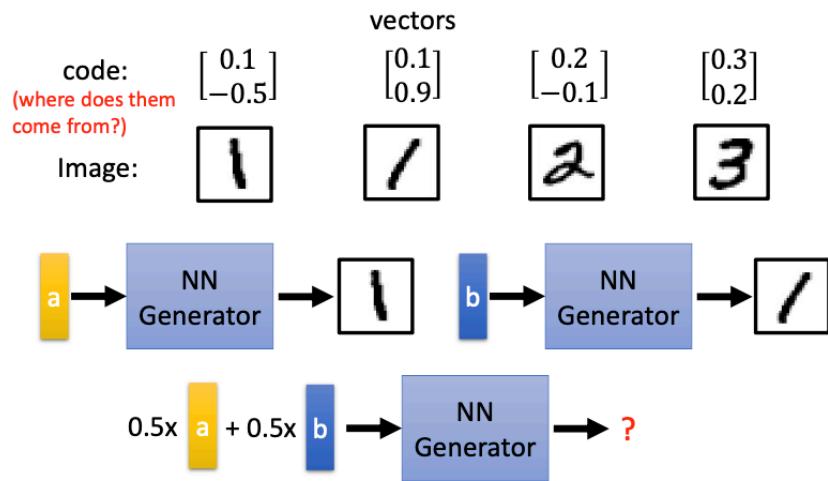
如果我们在这个范围内等距地sample, 就有以下结果,



但使用auto-encoder也会产生一些问题，如果输入vector为a，那么会产生1的图像，如果输入vector为b，会产生斜着的1；

Q：那么 $0.5a+0.5b$ 的vector会产生什么样的结果呢？

A：由于NN Generator是network，不是线性的，很可能产生不是数字的图像。

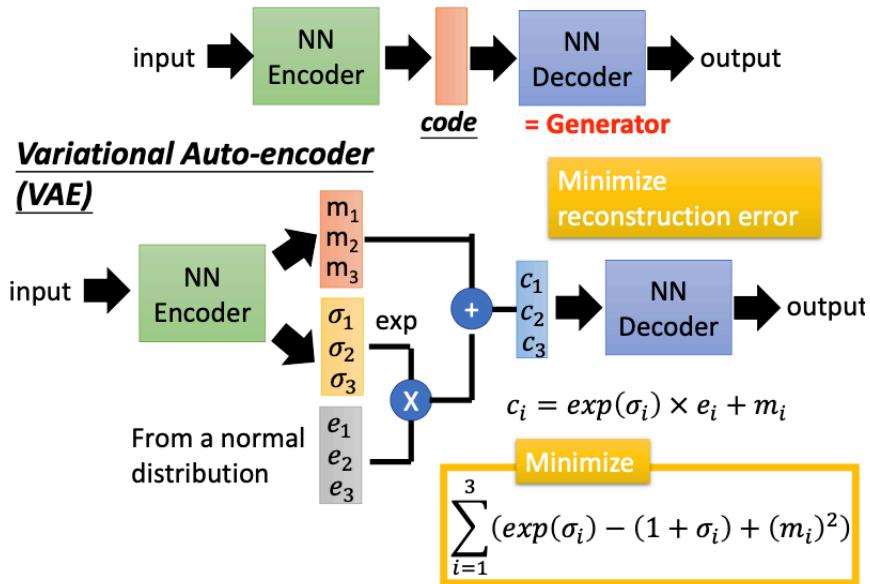


VAE (Variational Auto-encoder)

我们可以用VAE来解决这个问题，输入vector a之后，generator不仅产生code m，还会产生每一维的方差 σ ，还有一个额外的vector e (noise)，最后根据 $c_i = \exp(\sigma_i) \times e_i + m_i$ 得出需要的code；decoder需要根据这个带有noise的code，来还原出和input类似的图像。

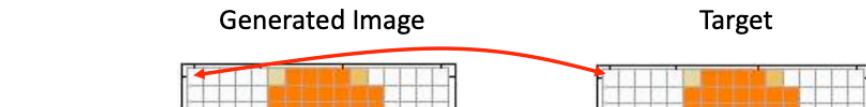
那么现在machine不仅看到vector a会产生数字，看到vector b也会产生数字；看到vector a+noise也会产生数字，看到vector b+noise也会产生数字。

因此，对于现在的generator，如果input是在训练的时候从来没见过的vector，也可能output出合理的object。



What do we miss?

现在有一张真实图像Target，通过generator生成的图像为generated image，目标是使这两者之间的差距越小越好。通常这两者之间的差距，可以通过每个像素点之间的差值来进行计算，比如可以将两张图像都表示成一个vector，再通过L1/L2计算两者之间的distance。

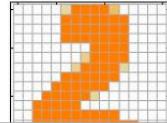


在一种理想的条件下，generator可以生成和Target完全一样的图像。但实际上，generator通常会犯一些错误，在生成的时候会有一些取舍，在一些地方会不得不做出妥协，选择在什么地方做妥协，对结果也会产生至关重要的影响。

在下图中，有一个Target，和4个Generated image。上面两张图像和target之间的差距只有一个pixel，下面两张图像和target之间的差距有6个pixel。但如果根据我们自己的观点，前两张图效果并不好，看起来并不像人写的数字，后两张图效果要好很多，因为后两张图只是对其中的一些笔画进行了延长。

What do we miss?

Target



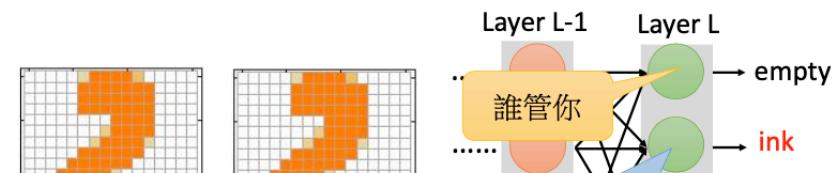
因此，我们并不能单纯地只是让我们的output和target越像越好，否则就会产生上述的生成效果。

Structure learning的输出是一个structure，由多个component组成，每个component之间的关系是非常重要的。对于下图中生成的数字图像，左边的图像有个pixel所在的位置很奇怪，不像是人手写的；但在右图中，我们添加了一些深色的pixel（和原来的pixel相邻）进去，就看起来很像是人手写的了。

但在neural network中，我们很难把component之间的关系放进去。在下图的网络结构中，layer L是output，表示颜色的深浅。如果layer L-1是给定的，那么输出的neural之间其实是independent的，值不会互相影响，也不会互相配合来产生一个相同颜色。为了把不同component之间的关系也考虑进去，我们可以加入多个hidden layer。

What do we miss?

Each neural in output layer corresponds to a pixel.

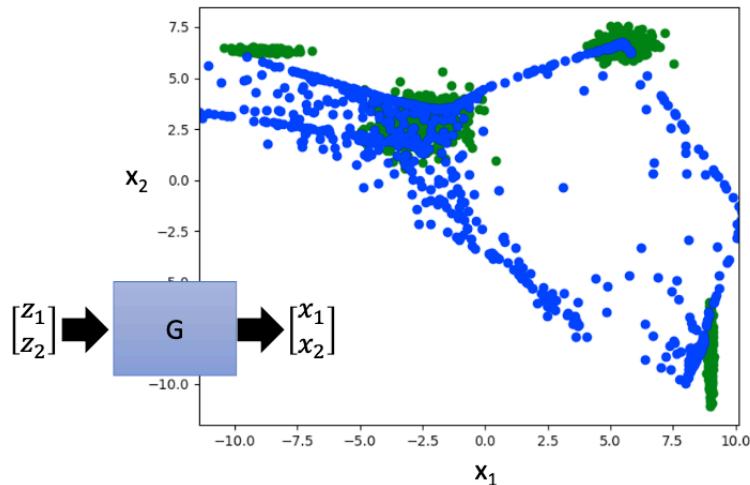


GAN vs VAE

如果现在有相同的network，我们用GAN和VAE都可以用来生成图像，但VAE需要用更大的network（更多的hidden layer）才能达到和GAN接近的结果。

蓝色的点为实验结果，绿色的点表示target，VAE最好能训练的结果也就是图中的蓝色分布了，因为VAE很难考虑不同的dimension之间的关系。

(Variational) Auto-encoder



Can Discriminator generate?

Discriminator

我们再来看看discriminator的概念，input为一张图像 x ，output为分数 $D(X)$ ，数值越大，表示这个image越接近真实图像。

Discriminator Evaluation function, Potential Function, Energy Function ...

- Discriminator is a function D (network, can deep)

$$D : X \rightarrow \mathbb{R}$$

- Input x : an object x (e.g. an image)
- Output $D(x)$: scalar which represents how “good” an object x is



Can we use the discriminator to generate objects?

Yes.

我们也可以用discriminator来生成图像。

我们之前提到用generator也可以用来生成图像，但考虑每个component之间的关系是非常困难的。对于discriminator，考虑每个component之间的关系，就相对而言比较容易了。

Q: 为什么使用discriminator来生成图像就有优势？

A: 在生成的时候，来考虑每个component之间的关系是很麻烦的；但在整个图片都生成完的时候，再来判断图像每个component之间的关系就很容易了。

在下图中，如果用discriminator来生成图像，就可以根据分数来判断生成的图像到底好不好，比如左边的数字2，就可以给一个很低的分数，右边的数字2，就可以给一个很高的分数。

- It is easier to catch the relation between the components by top-down evaluation.

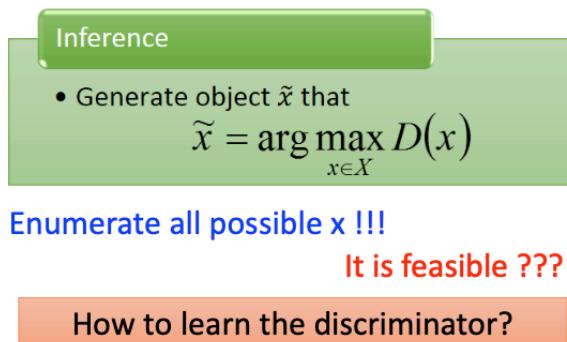


Q: 那么我们怎么来实际来判断分数呢?

A: discriminator很可能是convolution neural network, 对于其中的一个CNN filter (上图所示), 可以来判断某个pixel周围是不是都是empty的, 如果是的话, 就给他低分。

如果我们现在有一个discriminator, 我们可以通过以下方式来生成图像: 穷举所有可能的输入 x , 看哪一个输入可以得到最高的分数。

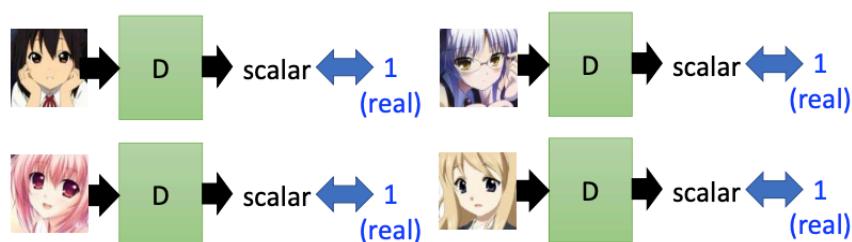
- Suppose we already have a good discriminator $D(x)$...



Discriminator - Training

Discriminator的训练不仅需要真实的图像, 也需要一些negative example。如果只有真实的图像, 从来没有看过negative example, 也不知道negative example长什么样子, D就有可能对所有的输入都输出一个很好的分数1.

- I have some real images

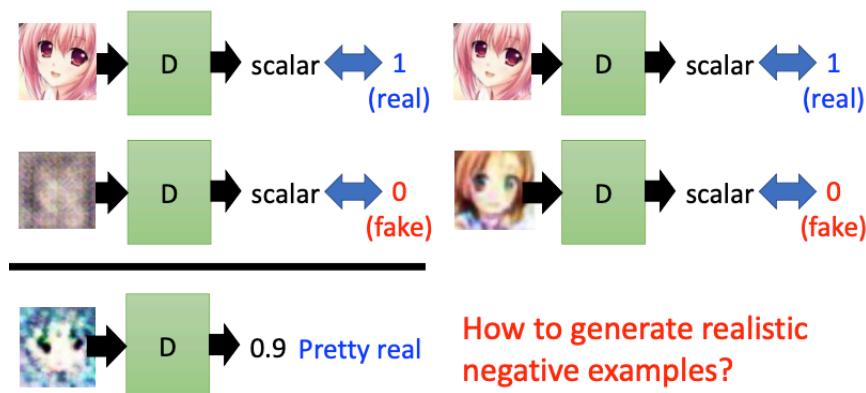


Discriminator only learns to output "1" (real).

Discriminator training needs some negative examples.

如何选择negative example也至关重要。在下图中，（左）如果negative example可以让机器很容易就分别出来 (0, fake) , 那么如果我们输入一个相对比较清晰的图像, D就会得出一个很好的分数0.9, 其实这个图像是fake的, 这并不是我们想看到的结果; （右）如果输入的是非常真实的negative example, D才可以很好地鉴别real or fake image。

- Negative examples are critical.



我们可以通过以下步骤来生成realistic negative examples, 首先给出positive和negative examples。再进行不断地循环。

在循环中, D要给positive example很高的分数, 给negative example很低的分数; 学习出这个discriminator D之后, 再用D来做generation (比如穷举找出 \tilde{x} , 使得分数取得最大值) ;

- 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

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

找出这些D觉得还不错的图像 \tilde{x} 之后, 进入下一次循环, 将 \tilde{x} 与positive examples进行比较, 学习出新的discriminator,

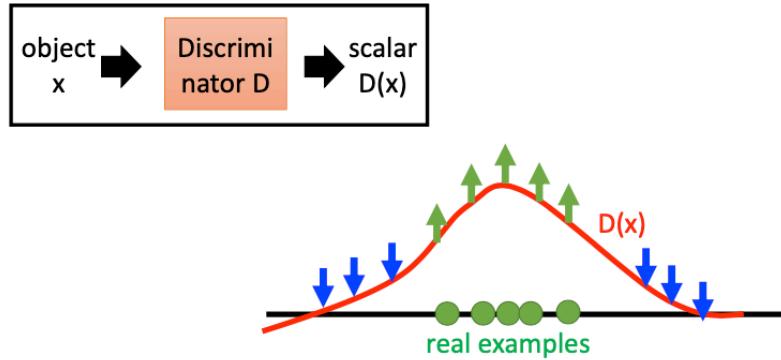
- Learn a discriminator D that can discriminate positive and negative examples.



- Generate negative examples by discriminator D

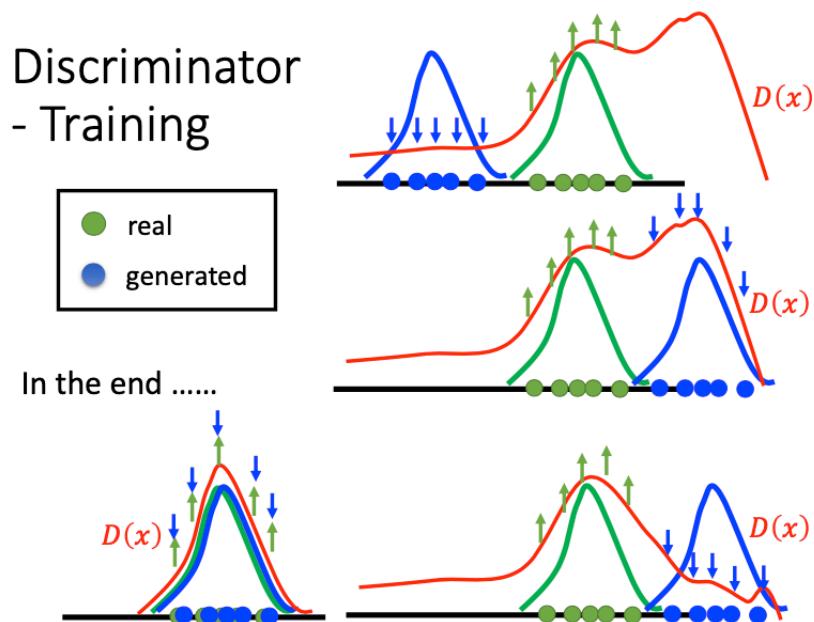
$$\tilde{x} = \arg \max_{x \in X} D(x)$$

我们现在假设object分布在一维空间中, 把object输入D, 得到分数 $D(x)$ 。在下图中, $D(x)$ 在real example区域取得了很不错的分数, 在其他区域则分数相对较低。



In practice, you cannot decrease all the x other than real examples.

但实际上， x 所在的空间是非常高维的，



Generator v.s. Discriminator

Generator:

- 优点：很容易做生成；
- 缺点：不容易考虑component之间的关系。

Discriminator:

- 优点：可以很容易考虑每个component之间的关系；
- 缺点：生成很慢。

- 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
 - Especially when your model is deep
- How to do negative sampling?

Generator + Discriminator

现在我们将Generator和Discriminator结合起来，使用G来生成negative examples，现在D就不用那么费力去寻找对应的negative examples了。

- 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.



Benefit of GAN

1. 从Discriminator的角度来看，现在我们只需要使用generator来生成negative examples，相比于之前的方法，可以更加高效；
2. 从Generator的角度来看，虽然还是生成component-by-component的object，但此时G得到的feedback不再是L1或者L2 loss了，不用再去计算pixel之间的相似度，而是更具有全局观的discriminator给出的分数评价。

- From Discriminator's point of view

- Using generator to generate negative samples

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

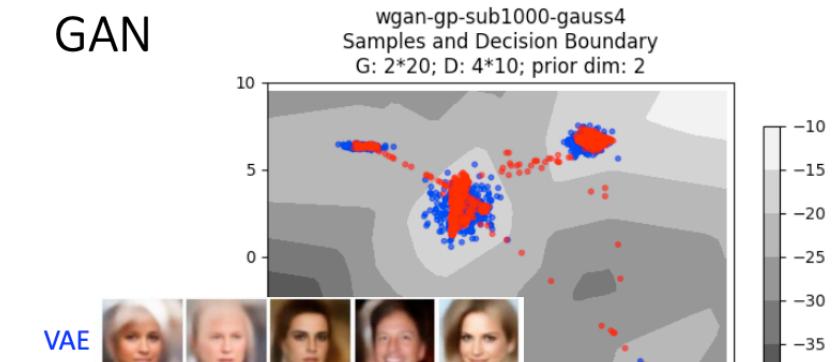
efficient

- From Generator's point of view

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

GAN

这里的数据还使用的上文VAE的数据，只是训练方法调整为了GAN。蓝色点点是generator要学习的目标，前文的VAE没有discriminator，可以发现在GAN比VAE的效果要好很多。在真实的环境中，VAE产生的人脸会比较模糊，GAN生成的人脸就没那么模糊。

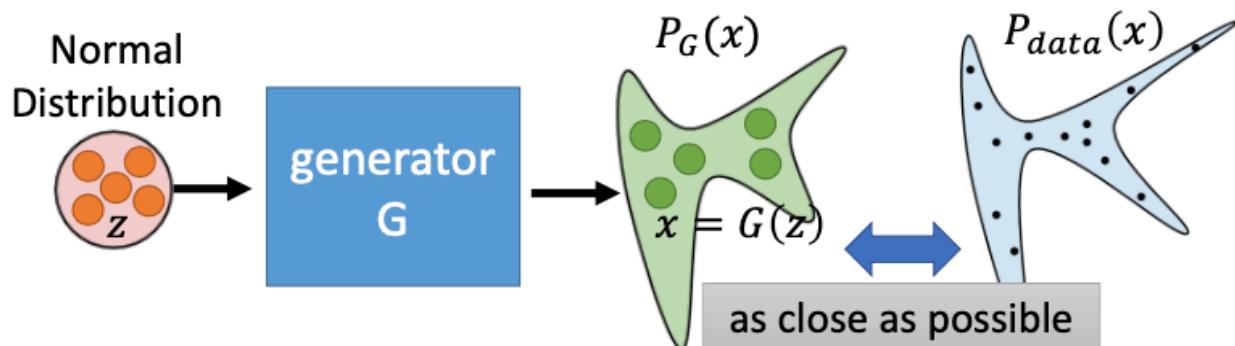


A little bit theory

Generator

x : an image (a high-dimensional vector)

- A generator G is a network. The network defines a probability distribution P_G



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

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

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.



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

The maximum objective value is related to JS divergence.

Discriminator

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

Blue star: data sampled from P_{data}

Orange star: data sampled from P_G

Training:

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



small divergence

train

Discriminator

hard to discriminate
(cannot make objective large)



large divergence

train

Discriminator

easy to discriminate

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

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

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