

GAN笔记——理论与实现

Anime GAN DCGAN WGAN ConditionalGAN

GAN这一概念是由 Ian Goodfellow 于2014年提出，并迅速成为了非常火热的研究话题，GAN的变种更是有上千种，深度学习先驱之一的 Yann LeCun 就曾说，"GAN及其变种是数十年来机器学习领域最有趣的idea"。那么什么是GAN呢？GAN的应用有哪些呢？GAN的原理是什么呢？怎样去实现一个GAN呢？本文将一一阐述。具体大纲如下：

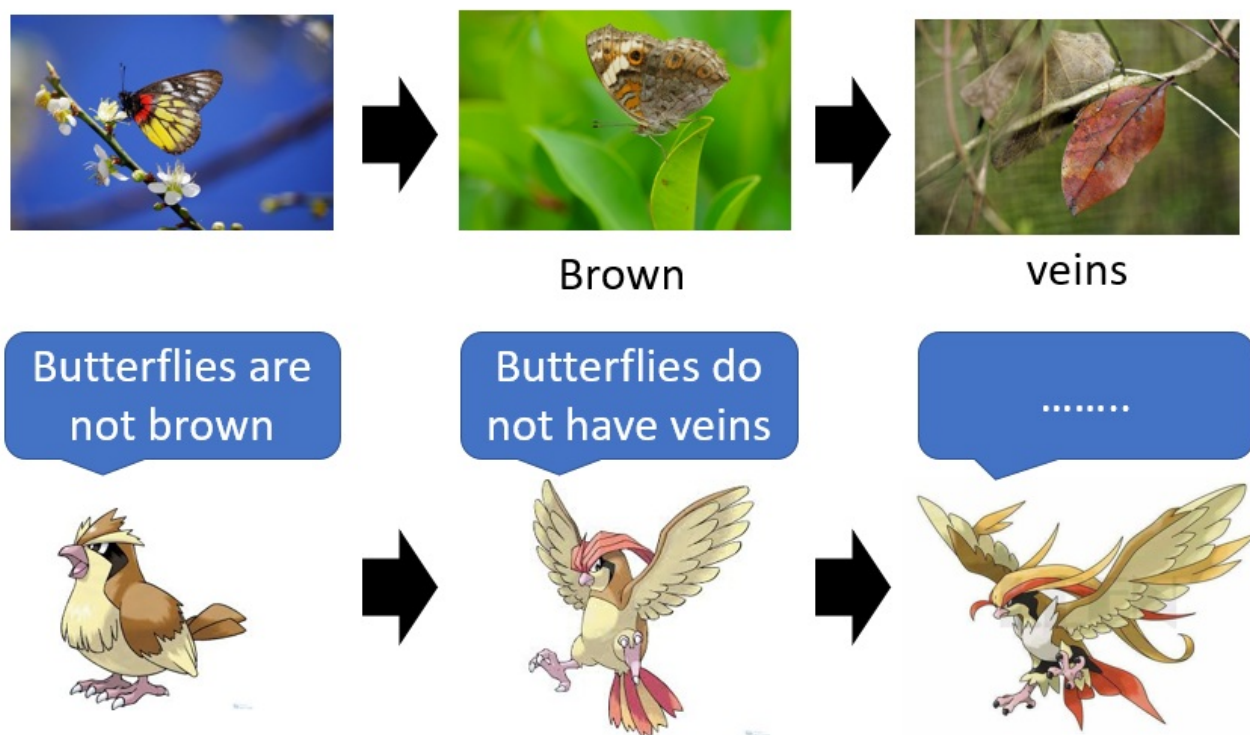
- 1.什么是GAN？
 - 1.1 对抗思想——破破鸟与枯叶蝶
 - 1.2 GAN思想——画画的演变
 - 1.3 零和博弈 (zero-sum game)
 - 1.4 小结
- 2. GAN的应用
- 3. GAN的原理
 - 3.1 生成器是否可以自我训练？
 - 3.2 鉴别器是否可以自我训练？
 - 3.3 生成器、鉴别器和GAN的优缺点
 - 3.4 GAN背后的理论
- 4.实现DCGAN [Github链接]
 - 4.1 model.py
 - 4.2 AnimeDataset.py
 - 4.3 utils.py
 - 4.4 main.py
- 5. 实现WGAN[Github链接]
- 6.实现ConditionalGAN [Github链接]
 - 6.1 CGAN.py
 - 6.2 utils.py
 - 6.3 main.py
 - 6.4 GUI.py
- 7. GAN小技巧

- 8. 参考

1. 什么是GAN？

GAN的英文全称是 `Generative Adversarial Network`，中文名是生成对抗网络，它由两个部分组成，一个是生成器（generative），还有一个是鉴别器，与生成器是敌对（Adversarial）关系。对GAN有了初步了解，知道它有两个模块组成，下面通过事例来理解这两个模块的产生思想？

1.1 对抗思想——啾啾鸟与枯叶蝶

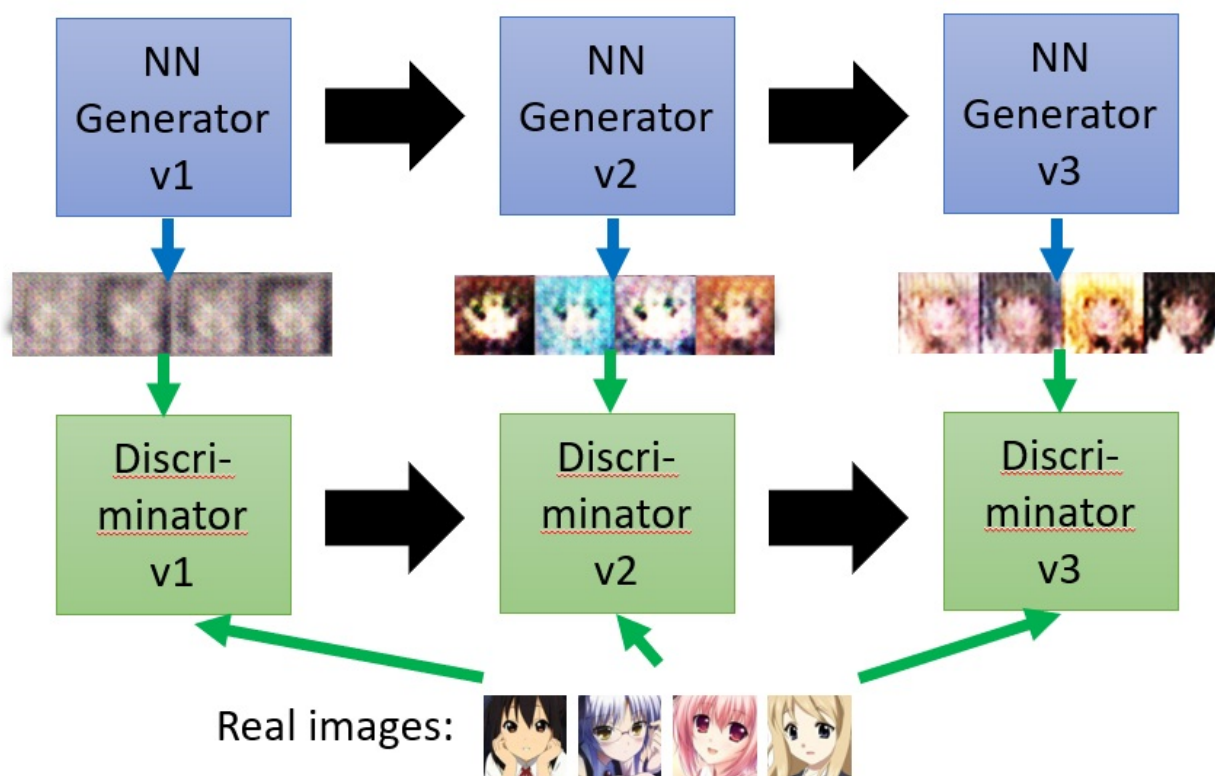


在生物进化的过程中，被捕食者会慢慢演化自己的特征，从而达到欺骗捕食者的目的，而捕食者也会根据情况调整自己对被捕食者的识别，共同进化，上图中的啾啾鸟和枯叶蝶就是这样的一种关系。生成器代表的是枯叶蝶，鉴别器代表的是啾啾鸟。它们的对抗思想与GAN类似，但GAN却有所不同。

1.2 GAN思想——画画的演变

GAN之所以有所不同，这里的原因是**GAN所作的工作与自然界的生物进化不同，它是已经知**

道最终鉴别的目标是什么样子的，不知道假目标是什么样子的，它会对生成器所产生的假目标做惩罚和对真目标进行奖励，这样鉴别器就知道什么目标是不好的假目标，什么目标是好的真目标，而生成器则是希望通过进化，产生比上一次更好的假目标，使鉴别器对自己的惩罚更小。以上是一个轮回，下一个轮回，鉴别器通过学习上一个轮回进化的假目标和真目标，再次进化对假目标的惩罚，而生成器不屈不挠，再次进化，直到以假乱真，与真目标一致，至此进化结束。



以上图为例，我们最开始画人物头像只知道有一个头的大致形状，有眼睛有鼻子等等，但画得不精致，后来通过找老师学习，画得更好了，有模有样，直到，我们画得与专门画头像的老师一样好。这里的我们就像是生成器，一步步进化（对应生成器不同的等级），这里的老师就像是鉴别器（这里只是比喻说明，现实世界的老师已经是一个成熟的鉴别器，不需要通过假样本进行学习，这里有那个意思就行）

1.3 零和博弈 (zero-sum game)

玩过纸牌的人知道，赢家的快乐是建立在输家的痛苦之上，收益和损失的总和始终为0。生成器和鉴别器也是这样一对博弈关系：鉴别器惩罚生成器，鉴别器收益，生成器损失；生成器进

化，使鉴别器对自己惩罚小，生成器收益，鉴别器损失。

1.4 小结

什么是GAN？GAN是由生成器和鉴别器两个部分组成，生成器的目的是生成假的目标，企图彻底骗过鉴别器的识别。而鉴别器通过学习真目标和假目标，提高自己的鉴别能力，不让假目标骗过自己。两者相互进化，相互博弈，一方进化，另一方损失，最后直到假目标与真目标很相似则停止进化。

2. GAN的应用

首先，我们要知道 **结构化学习**（**Structured Learning**），GAN也是结构化学习的一种。与分类和回归类似，结构化学习也是需要找到一个 $X \rightarrow Y$ 的映射，但结构化学习的输入和输出多种多样，可以是序列（sequence）到序列，序列到矩阵（matrix），矩阵到图（graph），图到树（tree）等等。这样，GAN的应用就十分广泛了。例如，**机器翻译**（machine translation）可以用GAN去做，如下图所示

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)

还有**语音识别**（speech recognition）以及**聊天机器人**（chat-bot）

Speech Recognition

X ：
(speech)

Y ：感謝大家來上課”
(transcription)

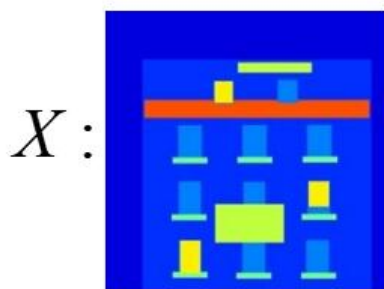
Chat-bot

X ：“How are you?”
(what a user says)

Y ：“I’m fine.”
(response of machine)

在图像方面，我们可以做**图像转图像**(image-to-image)，**彩色化**（colorization），还有**文本转图像**（text-to-image）

Image to Image



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>

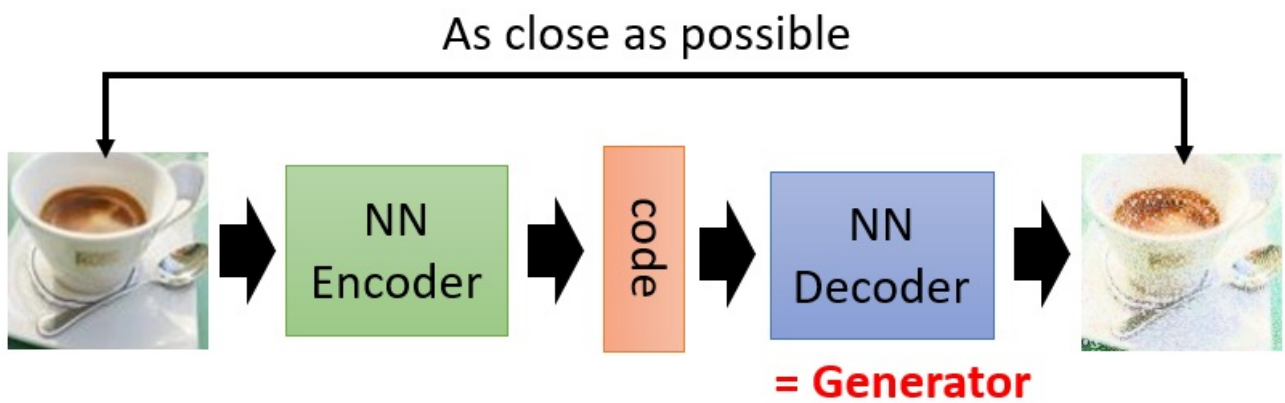
当然，GAN的应用远不止这么些，有非常有趣的变脸，图像自动打马赛克，自动生成多表情图像，年轻转年老等等，更多cool又skr的应用静待各位挖掘！

3. GAN原理

GAN的最终目的是为了生成能够产生以假乱真的目标的生成器。那么，是不是一定要用GAN呢？生成器可不可以自己训练得到目标？鉴别器可不可以自己训练得到目标？我们先来看这两个问题，然后再深入讨论GAN。

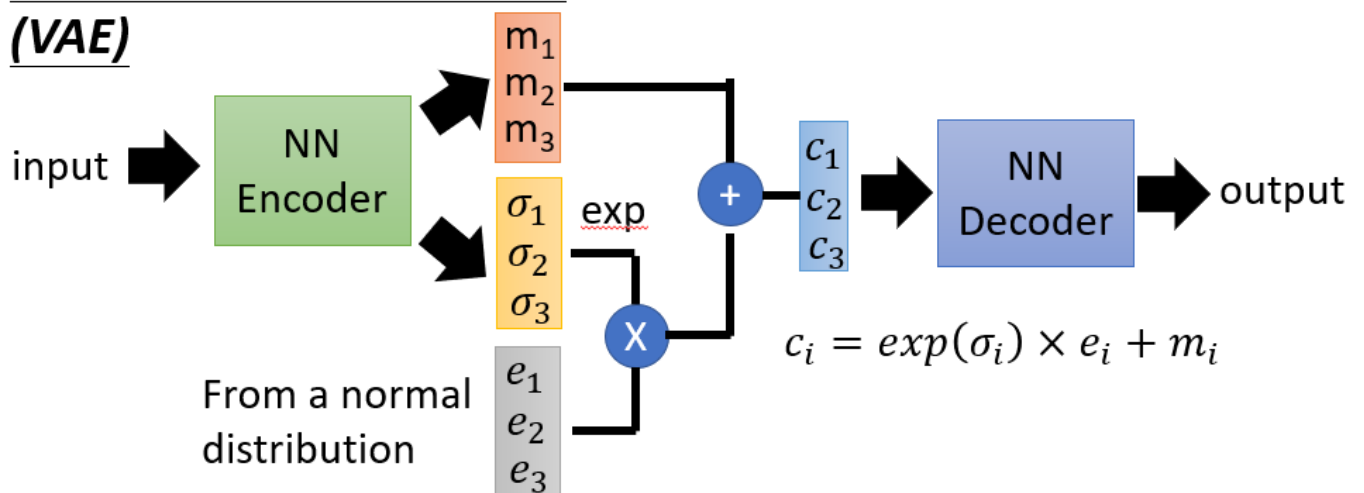
3.1 生成器是否可以自我训练？

答案是肯定的，我们所熟知的自编码器 (Auto-Encoder) 以及变分自编码器 (Variational Auto-Encoder) 都是典型的生成器。输入通过Encoder编码成code，然后code通过Decoder重建原图，其中自编码器中的Decoder就是生成器，code可随机取值，产生不同的输出。自编码器的结构如下：



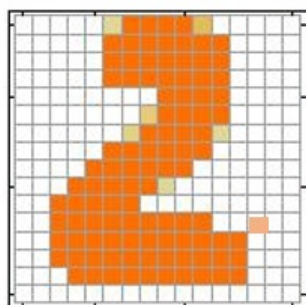
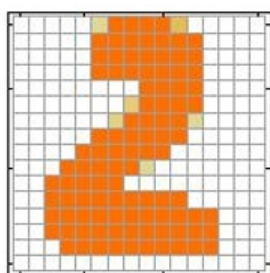
变分自编码器的结构如下

Variational Auto-encoder (VAE)



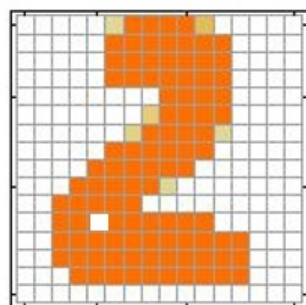
然后自编码器存在着问题，我们来看看下面这张图

Target



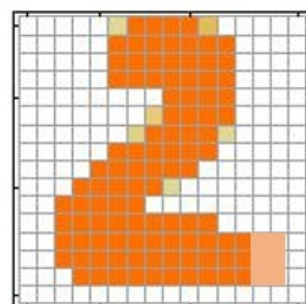
1 pixel error

我覺得不行



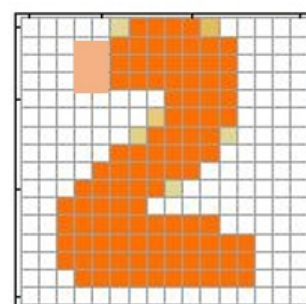
1 pixel error

我覺得不行



6 pixel errors

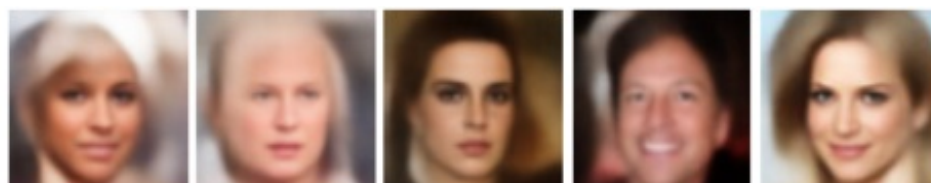
我覺得其實
可以



6 pixel errors

我覺得其實
可以

生成器的问题:由于自编码器的目标是让重建误差越来越小，但从上图中，我们可以看出，其中1个pixel的error，自编码器是觉得ok的，我们是觉得不行，另外6个pixel的误差我们觉得能接受的，自编码器不能接受，误差所在的位置很重要，而生成器并不知道这一点，自编码器缺少理解像素点之间的空间相关性的能力。还有一点，就是自编码器所产生的图像是模糊的，不能够产生十分清晰的图像，如下图所示



所以说目前单凭生成器是很难生成非常高质量的图像的。

3.2 鉴别器是否可以自我训练？

答案也是肯定的。鉴别器是给定一个输入，输出一个[0,1]的置信度，越接近1则置信越高，越接近0则置信度越低，如图所示：

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

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

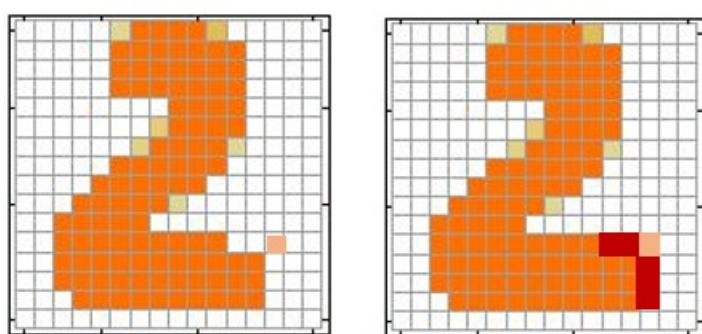
- Input x : an object x (e.g. an image)
- Output $D(x)$: scalar which represents how “good” an object x is



鉴别器的优势在于它可以很轻易地捕捉到元素之间的相关性，例如自编码器中出现的像素问题就不会在鉴别器中出现，如图所示，用一个滤波器就解决了。

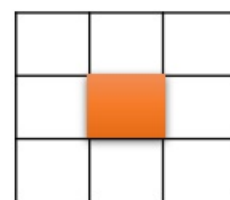
Discriminator

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



我覺得不行

我覺得其實 OK



This CNN filter is good enough.

现在来说说鉴别器要怎么样产生样本，参考下图：

Discriminator - Training

- **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.



v.s.



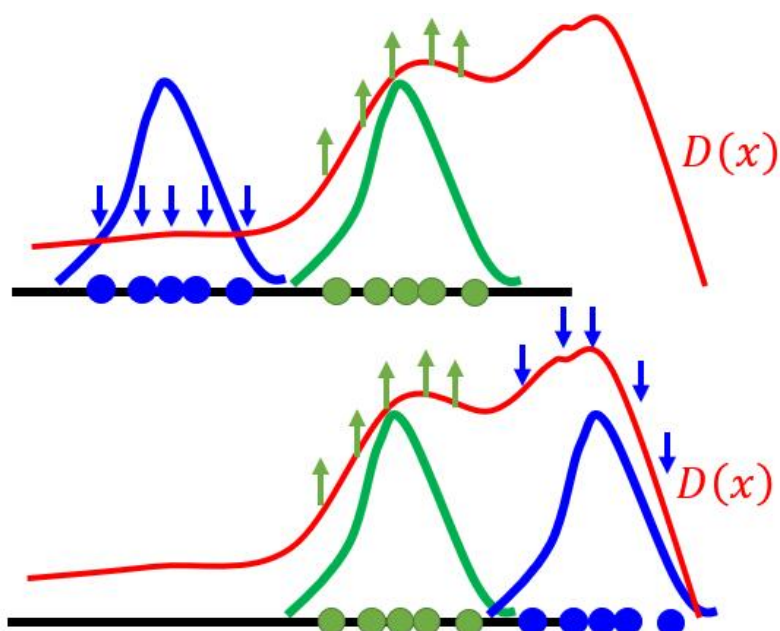
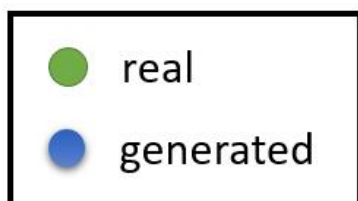
- Generate negative examples by discriminator D



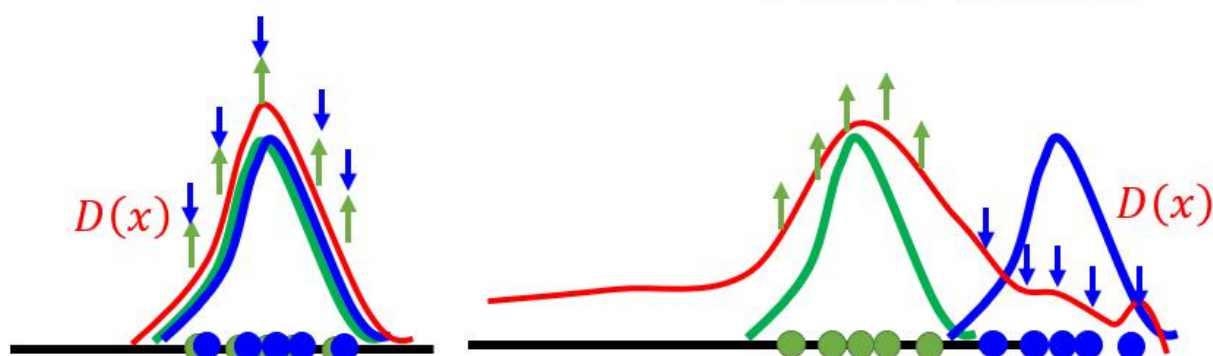
$$\tilde{x} = \arg \max_{x \in X} D(x)$$

首先也需要随机生成负样本，然后与真实样本一起送入鉴别器进行训练，在循环迭代中，通过最大概率选出最好的负样本，再与真样本一起送入鉴别器进行训练，然而，看起来和GAN训练差不多一致，没啥问题，其实这里面还有存在着问题的。我们来看下面这张图：

Discriminator - Training



In the end



鉴别器的问题：鉴别器的训练是对真样本进行奖励，对假样本进行压低，也就是图中的绿色抬高，蓝色压低，这就造成了问题，我们要训练出好的鉴别器，训练过程需要随机采样出除绿色图像外所有的假样本，这样鉴别器就只会对真实样本的分布取高分，对其他分布取低分，这样才能训练的好，然后再高维空间中，这样的负样本采样过程其实是很难进行的，而且还有一个问题，生成样本的过程要枚举大量样本，才有可能出现一个与真样本分布相符的样本，通过求那个最大化概率问题求出最好的样本，这实在是过于繁琐。

3.3 生成器、鉴别器和GAN的优缺点

通过上面的阐述，我们初步知道了它们的优缺点，下面这张ppt直观地给出了每个的优缺点，如图所示：

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

可以看出生成器和鉴别器的优缺点是可以互补的，这也就是GAN的优势。（**生成器+鉴别器**），下图介绍了GAN的优点，从两个角度出发。

- 从鉴别器的角度出发，利用生成器去生成样本，去求解最大化问题
- 从生成器角度出发，生成的样本依旧是逐个元素，但通过鉴别器可以得到全局性。

Benefit of GAN

- From Discriminator's point of view
 - Using generator to generate negative samples

$$\boxed{\begin{array}{c} \text{G} \rightarrow \tilde{x} \end{array}} = \boxed{\tilde{x} = \arg \max_{x \in X} D(x)}$$

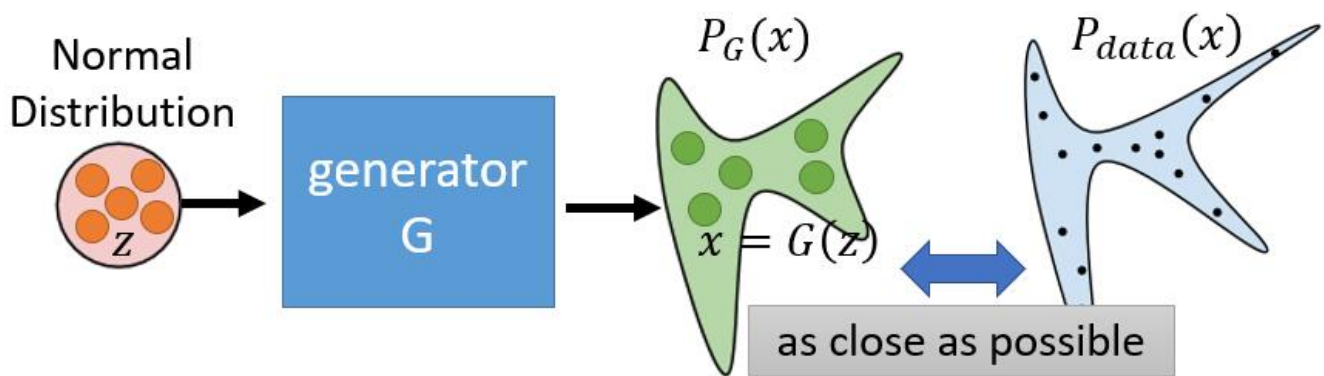
efficient

- From Generator's point of view
 - Still generate the object component-by-component
 - But it is learned from the discriminator with global view.

当然，GAN也是有缺点的，它是一种隐变量模型，可解释没有生成器和鉴别器强，另外GAN是不好进行训练。我在训练DAGAN的时候就成功造成了鉴别器的误差为0，无法进行反向传播更新梯度。

3.4 GAN背后的理论

对于生成器而言，它的目标是希望能够学习到真实样本的分布，这样就可以随机生成以假乱真的样本。如下图所示



如何去学习真实样本分布呢，这就需要用到极大似然估计 (Maximum Likelihood Estimation)，先来看看下面这张图

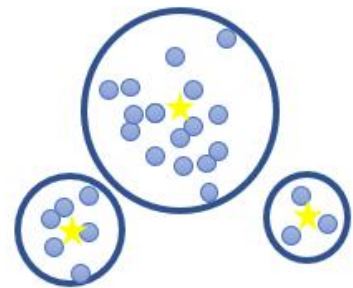
- Given a data distribution $P_{data}(x)$ (We can sample from it.)
- We have a distribution $P_G(x; \theta)$ parameterized by θ
 - We want to find θ such that $P_G(x; \theta)$ close to $P_{data}(x)$
 - E.g. $P_G(x; \theta)$ is a Gaussian Mixture Model, θ are means and variances of the Gaussians

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

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



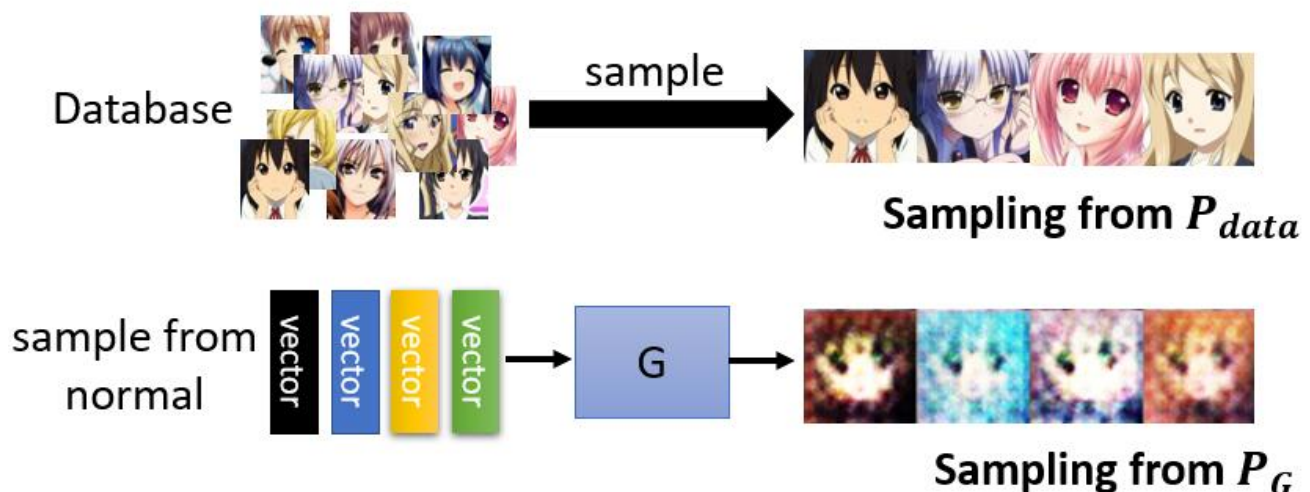
Find θ^* maximizing the likelihood

我们需要随机采样真实分布中的数据，通过学习 $P(x; \theta)$ 中的 θ ，希望 $P(x; \theta)$ 越接近 $P_{data}(x)$ ，其中每一个 x 对应的 $P_{data}(x)$ 的概率是很大的，为了使 $P(x; \theta)$ 越接近 $P_{data}(x)$ ，原问题等价于最大化每一个 $P(x_i; \theta)$ ，合起来就是最大化 $\prod_{i=1}^m P_G(x^i; \theta)$ 。而实际上极大似然估计是等价于最小化 $KL - divergence$ ，具体推导看下图，先取 \log (\log 是单调递增，不会改变原问题) 将相乘化为相加，最后变成了 P_{data} 下 $\log P_G(x; \theta)$ 的期望，然后转化成积分的形式，后面加了一项 $\int_x P_{data}(x) \log P_{data}(x) dx$ ，这一项是一个常数，没有变量 θ ，加了也不会影响原问题的解，加了这一项之后原问题就等于最小化 P_{data} 和 P_G 的 $KL - divergence$ 。

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

我们已经知道生成器要做的是 $\arg \min_G \text{Div}(P_{data}, P_G)$ ，这里 P_G 是我们要去最优化的，虽然我们有真实样本，但 P_G 的分布我们还是不知道，而且如何去定量计算 P_{data} 和 P_G 的 *divergence*，也就是 $\text{Div}(P_{data}, P_G)$ ，我们也是不知道的。所以接下来就需要引入鉴别器了。



虽然我们不知道 P_G 和 P_{data} 的分布，但我们可以随机采样它们分布的样本，如下图所示：



而我们知道鉴别器的目标是给真样本奖励，假样本惩罚，如下图所示，最后得到要鉴别器要优化的目标函数，鉴别器希望能够最大化这个目标函数，也就是 $\arg \max_D V(D, G)$ 。注意，这里是将 G 是 *fixed*，是不变的。

Example Objective Function for D

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

(G is fixed)

我们再来解这个问题，解出最优 D^* ，接下来的步骤就比较数学了，给一个目标函数，求出极大值解。具体如图下

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

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

Given x , the optimal D^* maximizing

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

Given x , the optimal D^* maximizing

$$\underbrace{P_{data}(x)}_a \log \underbrace{D(x)}_D + \underbrace{P_G(x)}_b \log \underbrace{(1-D(x))}_D$$

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

$$\frac{df(D)}{dD} = a \times \frac{1}{D} + b \times \frac{1}{1-D} \times (-1) = 0$$

$$a \times \frac{1}{D^*} = b \times \frac{1}{1-D^*} \quad a \times (1-D^*) = b \times D^* \quad a - aD^* = bD^* \quad a = (a+b)D^*$$

$$D^* = \frac{a}{a+b} \quad \Rightarrow \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \quad 0 < \quad < 1$$

$$\max_D V(G, D) = V(G, D^*) \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= -2\log 2 + \int_x P_{data}(x) \log \frac{P_{data}(x)}{(P_{data}(x) + P_G(x))/2} dx + \int_x P_G(x) \log \frac{P_G(x)}{(P_{data}(x) + P_G(x))/2} dx$$

$$= -2\log 2 + \text{KL}\left(P_{data} \parallel \frac{P_{data} + P_G}{2}\right) + \text{KL}\left(P_G \parallel \frac{P_{data} + P_G}{2}\right)$$

$$= -2\log 2 + 2\text{JSD}(P_{data} \parallel P_G) \quad \text{Jensen-Shannon divergence}$$

这个求解过程还是蛮详细的，最后我们竟然得到最大化 $V(D, G)$ 竟然等于一个常数加上 P_G 和 P_{data} 的 *JS-divergence* (*JS-divergence* 与 *KL-divergence* 类似，不会改变解)，这正是我们在生成器一直想求，可不会求得东西，鉴别器帮我们做到了。

于是，原始生成器的最优化问题 $\arg \min_G \text{Div}(P_G, P_{data})$ 就可以转化成

$\arg \min_G \max_D V(G, D)$ 。那如何来求解 $\arg \min_G \max_D V(G, D)$ 这个最小最大问题呢？

其实上面图上已经给出答案了，通过固定其中一个，求另一个，然后固定另一个，求之前固定住的这个。具体做法如图下：

- 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

更加详细的实践过程（也就是GAN的训练过程）如下所示，相信看了上面的一系列解释，会对GAN如此训练有了比较深的理解了吧。

Algorithm Initialize θ_d for D and θ_g for G

- In each training iteration:

Learning
D

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

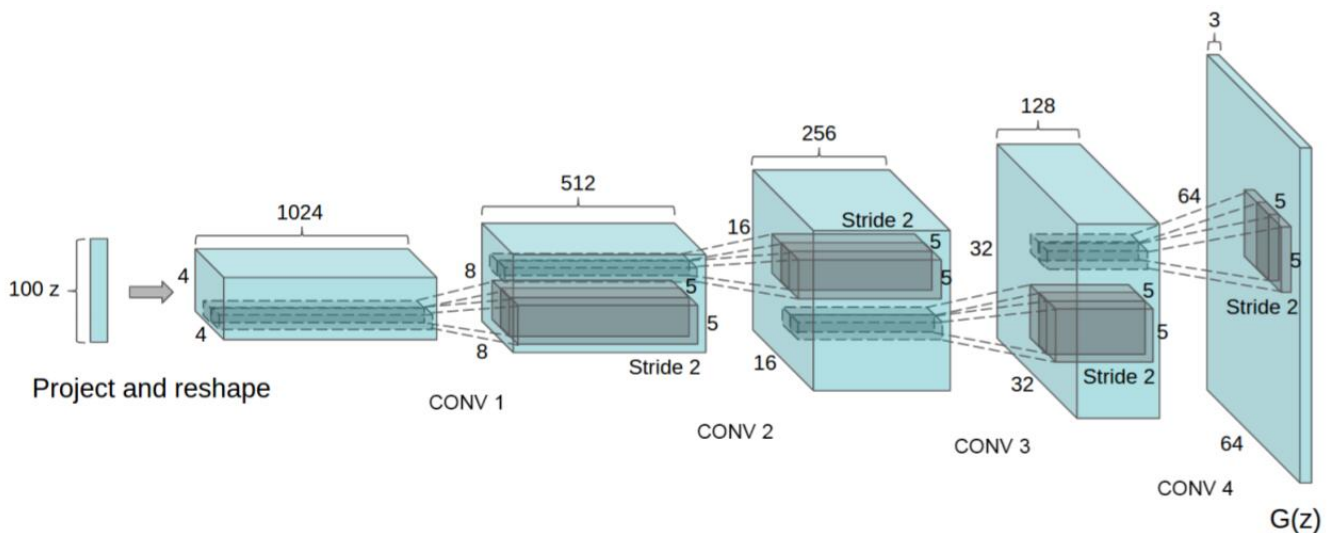
Learning
G

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

GAN的理论就到此结束。

4. 实现DCGAN

这里使用数据集是Anime——台大李宏毅老师的GAN课程的数据集，点击[链接](#)下载，首先我们来看一下DCGAN的框架，如图所示



这个是生成器的结构图，鉴别器的结构与生成器大致相反，DCGAN与普通的GAN有一些区别，具体分为下面几点

- DCGAN的网络都是全卷积的
- 生成器除最后一层外都加batchnorm，鉴别器则是第一层没加batchnorm
- 鉴别器中的激活函数使用的是leaky_relu，负斜率是0.2
- 生成器中的激活函数使用relu，输出层采用tanh
- 采用Adam优化算法，学习率是0.0002，beta1=0.5

4.1 model.py

```
1. import torch
2. import torch.nn as nn
3. import torch.functional as F
4.
5. class Generate(nn.Module):
6.     def __init__(self, input_dim=100):
7.         super(Generate, self).__init__()
8.         channel = [512, 256, 128, 64, 3]
9.         kernel_size = 4
10.        stride = 2
```



```

11.         padding = 1
12.         self.convtrans1_block = self.__convtrans_bolck(input_dim, channel
13.         el[0], 6, padding=0, stride=stride)
14.         self.convtrans2_block = self.__convtrans_bolck(channel[0], chan
15.         nel[1], kernel_size, padding, stride)
16.         self.convtrans3_block = self.__convtrans_bolck(channel[1], chan
17.         nel[2], kernel_size, padding, stride)
18.         self.convtrans4_block = self.__convtrans_bolck(channel[2], chan
19.         nel[3], kernel_size, padding, stride)
20.         self.convtrans5_block = self.__convtrans_bolck(channel[3], chan
21.         nel[4], kernel_size, padding, stride, layer="last_layer")
22.
23.     def __convtrans_bolck(self, in_channel, out_channel, kernel_size,
24.     padding, stride, layer=None):
25.         if layer == "last_layer":
26.             convtrans = nn.ConvTranspose2d(in_channel, out_channel, ker
27.             nel_size, stride, padding, bias=False)
28.             tanh = nn.Tanh()
29.             return nn.Sequential(convtrans, tanh)
30.         else:
31.             convtrans = nn.ConvTranspose2d(in_channel, out_channel, ker
32.             nel_size, stride, padding, bias=False)
33.             batch_norm = nn.BatchNorm2d(out_channel)
34.             relu = nn.ReLU(True)
35.             return nn.Sequential(convtrans, batch_norm, relu)
36.
37.     def forward(self, inp):
38.         x = self.convtrans1_block(inp)
39.         x = self.convtrans2_block(x)
40.         x = self.convtrans3_block(x)
41.         x = self.convtrans4_block(x)
42.         x = self.convtrans5_block(x)
43.         return x
44.
45. class Discriminator(nn.Module):
46.     def __init__(self):
47.         super(Discriminator, self).__init__()
48.         channels = [3, 64, 128, 256, 512]
49.         kernel_size = 4
50.         stride = 2
51.         padding = 1
52.         self.conv_bolck1 = self.__conv_block(channels[0], channels[1],
53.         kernel_size, stride, padding, "first_layer")
54.         self.conv_bolok2 = self.__conv_block(channels[1], channels[2],
55.         kernel_size, stride, padding)

```

```

46.         self.conv_bolok3 = self.__conv_block(channels[2], channels[3],
kernel_size, stride, padding)
47.         self.conv_bolok4 = self.__conv_block(channels[3], channels[4],
kernel_size, stride, padding)
48.         self.conv_bolok5 = self.__conv_block(channels[4], 1, kernel_size+1, stride, 0, "last_layer")
49.
50.         def __conv_block(self, inchannel, outchannel, kernel_size, stride,
padding, layer=None):
51.             if layer == "first_layer":
52.                 conv = nn.Conv2d(inchannel, outchannel, kernel_size, stride
, padding, bias=False)
53.                 leakrelu = nn.LeakyReLU(0.2, inplace=True)
54.                 return nn.Sequential(conv, leakrelu)
55.             elif layer == "last_layer":
56.                 conv = nn.Conv2d(inchannel, outchannel, kernel_size, stride
, padding, bias=False)
57.                 sigmoid = nn.Sigmoid()
58.                 return nn.Sequential(conv, sigmoid)
59.             else:
60.                 conv = nn.Conv2d(inchannel, outchannel, kernel_size, stride
, padding, bias=False)
61.                 batchnorm = nn.BatchNorm2d(outchannel)
62.                 leakrelu = nn.LeakyReLU(0.2, inplace=True)
63.                 return nn.Sequential(conv, batchnorm, leakrelu)
64.
65.         def forward(self, inp):
66.             x = self.conv_bolck1(inp)
67.             x = self.conv_bolok2(x)
68.             x = self.conv_bolok3(x)
69.             x = self.conv_bolok4(x)
70.             x = self.conv_bolok5(x)
71.             return x
72.
73.
74.         def weight_init(m):
75.             classname = m.__class__.__name__
76.             if classname.find('Conv') != -1:
77.                 m.weight.data.normal_(0,0.01)
78.             elif classname.find('BatchNorm') != -1:
79.                 m.weight.data.normal_(1.0,0.01)
80.                 m.bias.data.fill_(0)
81.
82.
83.

```

```

84.     if __name__ == "__main__":
85.         model1 = Generate()
86.         x = torch.randn(10,100,1,1)
87.         y = model1.forward(x)
88.         print(y.size())
89.         model2 = Discriminator()
90.         a = torch.randn(10,3,96,96)
91.         b = model2.forward(a)
92.         print(b.size())
93.

```

4.2 AnimeDataset.py

```

1.     import torch,torch.utils.data
2.     import numpy as np
3.     import scipy.misc, os
4.
5.     class AnimeDataset(torch.utils.data.Dataset):
6.         def __init__(self, directory, dataset, size_per_dataset):
7.             self.directory = directory
8.             self.dataset = dataset
9.             self.size_per_dataset = size_per_dataset
10.            self.data_files = []
11.            data_path = os.path.join(directory, dataset)
12.            for i in range(size_per_dataset):
13.                self.data_files.append(os.path.join(data_path,"{}.jpg".format(i)))
14.
15.            def __getitem__(self, ind):
16.                path = self.data_files[ind]
17.                img = scipy.misc.imread(path)
18.                img = img.transpose(2,0,1)-127.5/127.5
19.                return img
20.
21.            def __len__(self):
22.                return len(self.data_files)
23.
24.     if __name__ == "__main__":
25.         dataset = AnimeDataset(os.getcwd(),"anime",100)
26.         loader = torch.utils.data.DataLoader(dataset, batch_size=10, shuffle=True,num_workers=4)
27.         for i, inp in enumerate(loader):
28.             print(i,inp.size())

```

4.3 utils.py

```
1.  import os, imageio, scipy.misc
2.  import matplotlib.pyplot as plt
3.
4.
5.  def creat_gif(gif_name, img_path, duration=0.3):
6.      frames = []
7.      img_names = os.listdir(img_path)
8.      img_list = [os.path.join(img_path, img_name) for img_name in
img_names]
9.      for img_name in img_list:
10.         frames.append(imageio.imread(img_name))
11.         imageio.mimsave(gif_name, frames, 'GIF', duration=duration)
12.
13. def visualize_loss(generate_txt_path, discriminator_txt_path):
14.
15.     with open(generate_txt_path, 'r') as f:
16.         G_list_str = f.readlines()
17.
18.     with open(discriminator_txt_path, 'r') as f:
19.         D_list_str = f.readlines()
20.
21.     D_list_float, G_list_float = [], []
22.
23.     for D_item, G_item in zip(D_list_str, G_list_str):
24.         D_list_float.append(float(D_item.strip().split(':')[1]))
25.         G_list_float.append(float(G_item.strip().split(':')[1]))
26.
27.     list_epoch = list(range(len(D_list_float)))
28.
29.     full_path = os.path.join(os.getcwd(), "saved/logging.png")
30.     plt.figure()
31.     plt.plot(list_epoch, G_list_float, label="generate", color='g')
32.     plt.plot(list_epoch, D_list_float, label="discriminator", color='b'
)
33.     plt.legend()
34.     plt.title("DCGAN_Anime")
35.     plt.xlabel("epoch")
36.     plt.ylabel("loss")
37.     plt.savefig(full_path)
```

4.4 main.py

```
1.  import torch
2.  import torch.nn as nn
3.  from torch.optim import Adam
4.  from torchvision.utils import make_grid
5.  from model import Generate, Discriminator, weight_init
6.  from AnimeDataset import AnimeDataset
7.  import matplotlib.pyplot as plt
8.  import numpy as np
9.  import scipy.misc
10. import os, argparse
11. from tqdm import tqdm
12. from utils import creat_gif, visualize_loss
13.
14. def main():
15.
16.     parse = argparse.ArgumentParser()
17.
18.     parse.add_argument("--lr", type=float, default=0.0001,
19.                        help="learning rate of generate and
discriminator")
20.     parse.add_argument("--beta1", type=float, default=0.5,
21.                        help="adam optimizer parameter")
22.     parse.add_argument("--batch_size", type=int, default=64,
23.                        help="number of dataset in every train or test
iteration")
24.     parse.add_argument("--dataset", type=str, default="anime",
25.                        help="base path for dataset")
26.     parse.add_argument("--epochs", type=int, default=500,
27.                        help="number of training epochs")
28.     parse.add_argument("--loaders", type=int, default=4,
29.                        help="number of parallel data loading
processing")
30.     parse.add_argument("--size_per_dataset", type=int, default=30000,
31.                        help="number of training data")
32.     parse.add_argument("--pre_train", type=bool, default=False,
33.                        help="whether load pre_train model")
34.
35.     args = parse.parse_args()
36.
37.     if torch.cuda.is_available():
38.         device = torch.device("cuda")
39.     else:
```



```

40.         device = torch.device("cpu")
41.
42.         if not os.path.exists("saved"):
43.             os.mkdir("saved")
44.         if not os.path.exists("saved/img"):
45.             os.mkdir("saved/img")
46.
47.         if os.path.exists("faces"):
48.             pass
49.         else:
50.             print("Don't find the dataset directory, please copy the link
51. in website ,download and extract faces.tar.gz .\n \
52. https://drive.google.com/drive/folders/lmCsY5LEsgCnc0Txv0rpAUhKVPWVkbw5I
53. \n ")
54.             exit()
55.         if args.pre_train:
56.             generate = torch.load("saved/generate.t7").to(device)
57.             discriminator = torch.load("saved/discriminator.t7").to(device)
58.         else:
59.             generate = Generate().to(device)
60.             discriminator = Discriminator().to(device)
61.
62.         generate.apply(weight_init)
63.         discriminator.apply(weight_init)
64.
65.         dataset = AnimeDataset(os.getcwd(), args.dataset, args.size_per_dat
66. aset)
67.         dataload = torch.utils.data.DataLoader(dataset, batch_size=args.bat
68. ch_size, shuffle=True, num_workers=4)
69.
70.         criterion = nn.BCELoss().to(device)
71.
72.         optimizer_G = Adam(generate.parameters(), lr=args.lr, betas=(args.b
73. etal, 0.999))
74.         optimizer_D = Adam(discriminator.parameters(), lr=args.lr, betas=(a
75. rgs.betal, 0.999))
76.
77.         fixed_noise = torch.randn(64, 100, 1, 1).to(device)
78.
79.         for epoch in range(args.epochs):
80.
81.             print("Main epoch{:}".format(epoch))
82.             progress = tqdm(total=len(dataload.dataset))
83.             loss_d, loss_g = 0, 0

```

```

78.
79.     for i, inp in enumerate(dataload):
80.         # train discriminator
81.         real_data = inp.float().to(device)
82.         real_label = torch.ones(inp.size()[0]).to(device)
83.         noise = torch.randn(inp.size()[0], 100, 1, 1).to(device)
84.         fake_data = generate(noise)
85.         fake_label = torch.zeros(fake_data.size()[0]).to(device)
86.         optimizer_D.zero_grad()
87.         real_output = discriminator(real_data)
88.         real_loss = criterion(real_output.squeeze(), real_label)
89.         real_loss.backward()
90.         fake_output = discriminator(fake_data)
91.         fake_loss = criterion(fake_output.squeeze(), fake_label)
92.         fake_loss.backward()
93.         loss_D = real_loss + fake_loss
94.         optimizer_D.step()
95.
96.         #train generate
97.         optimizer_G.zero_grad()
98.         fake_data = generate(noise)
99.         fake_label = torch.ones(fake_data.size()[0]).to(device)
100.        fake_output = discriminator(fake_data)
101.        loss_G = criterion(fake_output.squeeze(), fake_label)
102.        loss_G.backward()
103.        optimizer_G.step()
104.
105.        progress.update(dataload.batch_size)
106.        progress.set_description("D:{}", G:{}".format(loss_D.item(),
loss_G.item()))
107.
108.        loss_g += loss_G.item()
109.        loss_d += loss_D.item()
110.
111.    loss_g /= (i+1)
112.    loss_d /= (i+1)
113.
114.    with open("generate_loss.txt", 'a+') as f:
115.        f.write("loss_G:{} \n".format(loss_G.item()))
116.
117.    with open("discriminator_loss.txt", 'a+') as f:
118.        f.write("loss_D:{} \n".format(loss_D.item()))
119.
120.    if epoch % 20 == 0:
121.

```

```
122.         torch.save(generate, os.path.join(os.getcwd(),
123.         "saved/generate.t7"))
124.         torch.save(discriminator, os.path.join(os.getcwd(), "saved/
125.         discriminator.t7"))
126.
127.         img = generate(fixed_noise).to("cpu").detach().numpy()
128.
129.         display_grid = np.zeros((8*96,8*96,3))
130.
131.         for j in range(int(64/8)):
132.             for k in range(int(64/8)):
133.                 display_grid[j*96:(j+1)*96,k*96:(k+1)*96,:] = (img[k
134.                 +8*j].transpose(1, 2, 0)+1)/2
135.
136.                 img_save_path = os.path.join(os.getcwd(),"saved/img/{}.png"
137.                 .format(epoch))
138.                 scipy.misc.imsave(img_save_path, display_grid)
139.
140.         creat_gif("evolution.gif", os.path.join(os.getcwd(),"saved/img"))
141.
142.         visualize_loss("generate_loss.txt", "discriminator_loss.txt")
143.
144. if __name__ == "__main__":
145.     main()
```

代码运行请参考github的[readme](#) , 最后500个epoch的结果图如下



5. 实现WGAN

WGAN pytorch版本一直都有bug，目前还没找到原因，实现了一个keras版本的，代码如下(运行前记得看readme)：

```
1. import os, scipy.misc
2. import keras.backend as K
3. from keras.models import Sequential, Model
```

```

4.  from keras.layers import Conv2D, ZeroPadding2D, BatchNormalization, Input
5.  from keras.layers import Conv2DTranspose, Reshape, Activation, Cropping2D, Flatten
6.  from keras.layers.advanced_activations import LeakyReLU
7.  from keras.optimizers import RMSprop
8.  from keras.activations import relu
9.  from keras.initializers import RandomNormal
10. from keras.preprocessing.image import ImageDataGenerator
11. import numpy as np
12.
13.
14. conv_init = RandomNormal(0, 0.02)
15. gamma_init = RandomNormal(1., 0.02)
16.
17. os.environ['KERAS_BACKEND']='tensorflow'
18. os.environ['TF_FORCE_GPU_ALLOW_GROWTH']='true'
19.
20. def DCGAN_D(isize, nc, ndf):
21.     inputs = Input(shape=(isize, isize, nc))
22.     x = ZeroPadding2D()(inputs)
23.     x = Conv2D(ndf, kernel_size=4, strides=2, use_bias=False, kernel_initializer=conv_init)(x)
24.     x = LeakyReLU(alpha=0.2)(x)
25.     for _ in range(4):
26.         x = ZeroPadding2D()(x)
27.         x = Conv2D(ndf*2, kernel_size=4, strides=2, use_bias=False, kernel_initializer=conv_init)(x)
28.         x = BatchNormalization(epsilon=1.01e-5, gamma_init=gamma_init)(x, training=1)
29.         x = LeakyReLU(alpha=0.2)(x)
30.         ndf *= 2
31.     x = Conv2D(1, kernel_size=3, strides=1, use_bias=False, kernel_initializer=conv_init)(x)
32.     outputs = Flatten()(x)
33.     return Model(inputs=inputs, outputs=outputs)
34.
35. def DCGAN_G(isize, nz, ngf):
36.     inputs = Input(shape=(nz,))
37.     x = Reshape((1, 1, nz))(inputs)
38.     x = Conv2DTranspose(filters=ngf, kernel_size=3, strides=2, use_bias=False,
39.                         kernel_initializer = conv_init)(x)
40.     for _ in range(4):
41.         x = Conv2DTranspose(filters=int(ngf/2), kernel_size=4, strides=

```



```

42.         2, use_bias=False,
43.             kernel_initializer = conv_init)(x)
44.         x = Cropping2D(cropping=1)(x)
45.         x = BatchNormalization(epsilon=1.01e-5, gamma_init=gamma_init)(
46. x, training=1)
47.         x = Activation("relu")(x)
48.         ngf = int(ngf/2)
49.         x = Conv2DTranspose(filters=3, kernel_size=4, strides=2, use_bias=F
50. else,
51.             kernel_initializer = conv_init)(x)
52.         x = Cropping2D(cropping=1)(x)
53.         outputs = Activation("tanh")(x)
54.
55.         return Model(inputs=inputs, outputs=outputs)
56.
57. nc = 3
58. nz = 100
59. ngf = 1024
60. ndf = 64
61. imageSize = 96
62. batchSize = 64
63. lrD = 0.00005
64. lrG = 0.00005
65. clamp_lower, clamp_upper = -0.01, 0.01
66.
67. netD = DCGAN_D(imageSize, nc, ndf)
68. netD.summary()
69.
70. netG = DCGAN_G(imageSize, nz, ngf)
71. netG.summary()
72.
73. clamp_updates = [K.update(v, K.clip(v, clamp_lower, clamp_upper))
74.                   for v in netD.trainable_weights]
75. netD_clamp = K.function([], [], clamp_updates)
76.
77. netD_real_input = Input(shape=(imageSize, imageSize, nc))
78. noisev = Input(shape=(nz,))
79.
80. loss_real = K.mean(netD(netD_real_input))
81. loss_fake = K.mean(netD(netG(noisev)))
82. loss = loss_fake - loss_real
83. training_updates = RMSprop(lr=lrD).get_updates(netD.trainable_weights,
84. [], loss)
85. netD_train = K.function([netD_real_input, noisev],
86. [loss_real, loss_fake],

```

```

83.         training_updates)
84.
85.     loss = -loss_fake
86.     training_updates = RMSprop(lr=lrG).get_updates(netG.trainable_weights,
87.     [], loss)
88.
89.     netG_train = K.function([noisev], [loss], training_updates)
90.
91.     fixed_noise = np.random.normal(size=(batchSize, nz)).astype('float32')
92.
93.     datagen = ImageDataGenerator(
94.         # featurewise_center=True,
95.         # featurewise_std_normalization=True,
96.         rotation_range=20,
97.         rescale=1./255
98.     )
99.
100.     train_generate = datagen.flow_from_directory("faces/", target_size=(96,
101.     96), batch_size=64,
102.
103.     shuffle=True, class_mode
104.     =None, save_format='jpg')
105.
106.     step = 0
107.     print(dir(train_generate))
108.     for step in range(100000):
109.
110.         for _ in range(5):
111.             real_data = (np.array(train_generate.next())*2-1)
112.             noise = np.random.normal(size=(batchSize, nz))
113.             errD_real, errD_fake = netD_train([real_data, noise])
114.             errD = errD_real - errD_fake
115.             netD_clamp([])
116.
117.             noise = np.random.normal(size=(batchSize, nz))
118.             errG, = netG_train([noise])
119.             print('[%d] Loss_D: %f Loss_G: %f Loss_D_real: %f Loss_D_fake %f'
120.             % (step, errD, errG, errD_real, errD_fake))
121.
122.             if step%1000==0:
123.                 netD.save("discriminator.h5")
124.                 netG.save("generate.h5")
125.                 fake = netG.predict(fixed_noise)
126.                 display_grid = np.zeros((8*96,8*96,3))
127.
128.                 for j in range(int(64/8)):
129.                     for k in range(int(64/8)):

```

```

124.         display_grid[j*96:(j+1)*96,k*96:(k+1)*96,:] = fake[k+8*j
    ]
125.         img_save_path = os.path.join(os.getcwd(),"saved/img/{}.png".for
    mat(step))
126.         scipy.misc.imsave(img_save_path, display_grid)

```

代码运行请参考github的[readme](#) , 100000step的结果 :



6. 实现ConditionalGAN

详细运行请看github中的[readme](#)。

6.1 CGAN.py

```
1. import torch,os,scipy.misc,random
2. import torch.nn as nn
3. import numpy as np
4. import torch.nn.functional as F
5. from torch.optim import Adam
6. from utils import load_Anime,test_Anime
7.
8.
9.
10. class Reshape(nn.Module):
11.     def __init__(self, *args):
12.         super(Reshape, self).__init__()
13.         self.shape = args
14.     def forward(self, x):
15.         return x.view(self.shape)
16.
17. class Generate(nn.Module):
18.     def __init__(self, z_dim, y_dim, image_height, image_width):
19.         super(Generate, self).__init__()
20.         self.conv_trans = nn.Sequential(
21.             nn.Linear(z_dim+y_dim, (image_height//16)*(image_width//
22. /16)*384),
23.             nn.BatchNorm1d((image_height//16)*(image_width//16)*384
24. ,
25.                 eps=1e-5, momentum=0.9, affine=True),
26.             Reshape(-1, 384, image_height//16, image_width//16),
27.             nn.ConvTranspose2d(384, 256, kernel_size=4, stride=2, p
28. adding=1, bias=False),
29.             nn.BatchNorm2d(256, eps=1e-5, momentum=0.9, affine=True
30. ),
31.             nn.ReLU(inplace=True),
32.             nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, p
33. adding=1, bias=False),
34.             nn.BatchNorm2d(128, eps=1e-5, momentum=0.9, affine=True
35. ),
36.             nn.ReLU(inplace=True),
37.             nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, pa
38. dding=1, bias=False),
39.             nn.BatchNorm2d(64, eps=1e-5, momentum=0.9, affine=True)
```



```

33.         nn.ReLU(inplace=True),
34.         nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2,
padding=1, bias=False),
35.         nn.Tanh()
36.     )
37.
38.     def forward(self, z, y):
39.         z = torch.cat((z,y), dim=-1)
40.         z = self.conv_trans(z)
41.         return z
42.
43. class Discriminator(nn.Module):
44.     def __init__(self):
45.         super(Discriminator, self).__init__()
46.         self.conv = nn.Sequential(
47.             nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1,
bias=False),
48.             nn.LeakyReLU(0.2, inplace=True),
49.             nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1,
bias=False),
50.             nn.LeakyReLU(0.2, inplace=True),
51.             nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1,
bias=False),
52.             nn.LeakyReLU(0.2, inplace=True),
53.             nn.Conv2d(256, 384, kernel_size=4, stride=2, padding=1,
bias=False),
54.             nn.LeakyReLU(0.2, inplace=True),
55.         )
56.         self.conv1 = nn.Sequential(
57.             nn.Conv2d(407, 384, kernel_size=1, stride=1, padding=0,
bias=False),
58.             nn.LeakyReLU(0.2, inplace=True)
59.         )
60.         self.linear = nn.Linear(4*4*384, 1)
61.
62.     def forward(self, x, y):
63.         x = self.conv(x)
64.         y = torch.unsqueeze(y, 2)
65.         y = torch.unsqueeze(y, 3)
66.         y = y.expand(y.size()[0], y.size()[1], x.size()[2], x.size()[3]
)
67.
68.         x = torch.cat((x,y), dim=1)
69.         x = self.conv1(x)
70.         x = x.view(x.size()[0], -1)

```



```

70.         x = self.linear(x)
71.         x = x.squeeze()
72.         x = F.sigmoid(x)
73.         return x
74.
75.     def weight_init(m):
76.         classname = m.__class__.__name__
77.         if classname.find('Conv') != -1:
78.             m.weight.data.normal_(0,0.01)
79.         elif classname.find('BatchNorm') != -1:
80.             m.weight.data.normal_(1.0,0.01)
81.             m.bias.data.fill_(0)
82.
83.     class CGAN(object):
84.
85.         def __init__(self, dataset_path, save_path, epochs, batchsize,
86. z_dim, device, mode):
87.             self.dataset_path = dataset_path
88.             self.save_path = save_path
89.             self.epochs = epochs
90.             self.batch_size = batchsize
91.             self.mode = mode
92.             self.image_height = 64
93.             self.image_width = 64
94.             self.learning_rate = 0.0001
95.             self.z_dim = z_dim
96.             self.y_dim = 23
97.             self.iters_d = 2
98.             self.iters_g = 1
99.             self.device = device
100.            self.criterion = nn.BCELoss().to(device)
101.            if mode == "train":
102.                self.X, self.Y = load_Anime(self.dataset_path)
103.                self.batch_nums = len(self.X)//self.batch_size
104.
105.            def train(self):
106.                generate = Generate(self.z_dim, self.y_dim, self.image_height,
107. self.image_width).to(self.device)
108.                discriminator = Discriminator().to(self.device)
109.                generate.apply(weight_init)
110.                discriminator.apply(weight_init)
111.                optimizer_G = Adam(generate.parameters(), lr=self.learning_rate
112. )
113.                optimizer_D = Adam(discriminator.parameters(), lr=self.learning
114. _rate)

```

```

111.         step = 0
112.         for epoch in range(self.epochs):
113.             print("Main epoch:{}".format(epoch))
114.             for i in range(self.batch_nums):
115.                 step += 1
116.                 batch_images = torch.from_numpy(np.asarray(self.X[i*self
f.batch_size:(i+1)*self.batch_size])).astype(np.float32)).to(self.device
)
117.                 batch_labels = torch.from_numpy(np.asarray(self.Y[i*self
f.batch_size:(i+1)*self.batch_size])).astype(np.float32)).to(self.device
)
118.                 batch_images_wrong = torch.from_numpy(np.asarray(self.X
[random.sample(range(len(self.X)), len(batch_images))]).astype(np.float
32)).to(self.device)
119.                 batch_labels_wrong = torch.from_numpy(np.asarray(self.Y
[random.sample(range(len(self.Y)), len(batch_images))]).astype(np.float
32)).to(self.device)
120.                 batch_z = torch.from_numpy(np.random.normal(0, np.exp(-
1 / np.pi), [self.batch_size, self.z_dim])).astype(np.float32)).to(self.
device)
121.                 # discriminator twice, generate once
122.                 for _ in range(self.iters_d):
123.                     optimizer_D.zero_grad()
124.                     d_loss_real =
self.criterion(discriminator(batch_images, batch_labels), torch.ones(s
elf.batch_size).to(self.device))
125.                     d_loss_fake =
(self.criterion(discriminator(batch_images, batch_labels_wrong), torch
.zeros(self.batch_size).to(self.device)) \
126.                      + self.criterion(discriminator(batch_i
mages_wrong, batch_labels),
torch.zeros(self.batch_size).to(self.device)) \
127.                      + self.criterion(discriminator(generat
e(batch_z, batch_labels), batch_labels), torch.zeros(self.batch_size).
to(self.device)))/3
128.                     d_loss = d_loss_real + d_loss_fake
129.                     d_loss.backward()
130.                     optimizer_D.step()
131.
132.                 for _ in range(self.iters_g):
133.                     optimizer_G.zero_grad()
134.                     g_loss = self.criterion(discriminator(generate(batc
h_z, batch_labels), batch_labels), torch.ones(self.batch_size).to(self
.device))
135.                     g_loss.backward()

```

```

136.         optimizer_G.step()
137.
138.         print("epoch:{}, step:{}, d_loss:{}, g_loss:{}".format(
epoch, step, d_loss.item(), g_loss.item()))
139.         #show result and save model
140.         if (step)%5000 == 0:
141.             z, y = test_Anime()
142.             image = generate(torch.from_numpy(z).float().to(self
self.device), torch.from_numpy(y).float().to(self.device)).to("cpu").detach
().numpy()
143.             display_grid = np.zeros((5*64,5*64,3))
144.             for j in range(5):
145.                 for k in range(5):
146.                     display_grid[j*64:(j+1)*64,k*64:(k+1)*64,:]
= image[k+5*j].transpose(1, 2, 0)
147.             img_save_path =
os.path.join(self.save_path,"training_img/{}.png".format(step))
148.             scipy.misc.imsave(img_save_path, display_grid)
149.             torch.save(generate, os.path.join(self.save_path, "
generate.t7"))
150.             torch.save(discriminator,
os.path.join(self.save_path, "discriminator.t7"))
151.
152.         def infer(self):
153.             z, y = test_Anime()
154.             generate = torch.load(os.path.join(self.save_path, "generate.t7
")).to(self.device)
155.             image = generate(torch.from_numpy(z).float().to(self.device), to
rch.from_numpy(y).float().to(self.device)).to("cpu").detach().numpy()
156.             display_grid = np.zeros((5*64,5*64,3))
157.             for j in range(5):
158.                 for k in range(5):
159.                     display_grid[j*64:(j+1)*64,k*64:(k+1)*64,:] = image[k+5*
j].transpose(1, 2, 0)
160.             img_save_path =
os.path.join(self.save_path,"testing_img/test.png")
161.             scipy.misc.imsave(img_save_path, display_grid)
162.             print("infer ended, look the result in the save/testing_img/")

```

6.2 utils.py

```

1.     # most code from
https://github.com/JasonYao81000/MLDS2018SPRING/blob/master/hw3/hw3_2/

```

```

2. import numpy as np
3. import cv2
4. import os
5.
6. def test_Anime():
7.     np.random.seed(999)
8.     z = np.random.normal(0, np.exp(-1 / np.pi), [25, 62])
9.     tag_dict = ['orange hair', 'white hair', 'aqua hair', 'gray hair',
10.                'green hair', 'red hair', 'purple hair',
11.                'pink hair', 'blue hair', 'black hair', 'brown hair',
12.                'blonde hair',
13.                'gray eyes', 'black eyes', 'orange eyes', 'pink eyes', 'yellow eyes',
14.                'aqua eyes', 'purple eyes', 'green eyes', 'brown eyes', 'red eyes', 'blue eyes']
15.     tag_txt = open("test.txt", 'r').readlines()
16.     labels = []
17.     for line in tag_txt:
18.         label = np.zeros(len(tag_dict))
19.         for i in range(len(tag_dict)):
20.             if tag_dict[i] in line:
21.                 label[i] = 1
22.         labels.append(label)
23.     for i in range(len(tag_txt)):
24.         for j in range(4):
25.             labels.insert(5*i+j, labels[5*i])
26.
27.     return z, np.array(labels)
28.
29.
30. def load_Anime(dataset_filepath):
31.     tag_csv_filename = dataset_filepath.replace('images/', 'tags.csv')
32.     tag_dict = ['orange hair', 'white hair', 'aqua hair', 'gray hair',
33.                'green hair', 'red hair', 'purple hair',
34.                'pink hair', 'blue hair', 'black hair', 'brown hair',
35.                'blonde hair',
36.                'gray eyes', 'black eyes', 'orange eyes', 'pink eyes', 'yellow eyes',
37.                'aqua eyes', 'purple eyes', 'green eyes', 'brown eyes', 'red eyes', 'blue eyes']
38.
39.     tag_csv = open(tag_csv_filename, 'r').readlines()

```

```

39.     id_label = []
40.     for line in tag_csv:
41.         id, tags = line.split(',')
42.         label = np.zeros(len(tag_dict))
43.
44.         for i in range(len(tag_dict)):
45.             if tag_dict[i] in tags:
46.                 label[i] = 1
47.
48.         # Keep images with hair or eyes.
49.         if np.sum(label) == 2 or np.sum(label) == 1:
50.             id_label.append((id, label))
51.
52.
53.     # Load file name of images.
54.     image_file_list = []
55.     for image_id, _ in id_label:
56.         image_file_list.append(image_id + '.jpg')
57.
58.     # Resize image to 64x64.
59.     image_height = 64
60.     image_width = 64
61.     image_channel = 3
62.
63.     # Allocate memory space of images and labels.
64.     images = np.zeros((len(image_file_list), image_channel,
image_width, image_height))
65.     labels = np.zeros((len(image_file_list), len(tag_dict)))
66.     print ('images.shape: ', images.shape)
67.     print ('labels.shape: ', labels.shape)
68.
69.     print ('Loading images to numpy array...')
70.     data_dir = dataset_filepath
71.     for index, filename in enumerate(image_file_list):
72.         images[index] = cv2.cvtColor(
73.             cv2.resize(
74.                 cv2.imread(os.path.join(data_dir, filename), cv2.IMREAD
_COLOR),
75.                     (image_width, image_height)),
76.                 cv2.COLOR_BGR2RGB).transpose(2,0,1)
77.         labels[index] = id_label[index][34]
78.
79.     print ('Random shuffling images and labels...')
80.     np.random.seed(9487)
81.     indice = np.array(range(len(image_file_list)))

```



```

82.     np.random.shuffle(indice)
83.     images = images[indice]
84.     labels = labels[indice]
85.
86.     print ('[Tip 1] Normalize the images between -1 and 1.')
87.     # Tip 1. Normalize the inputs
88.     #     Normalize the images between -1 and 1.
89.     #     Tanh as the last layer of the generator output.
90.     return (images / 127.5) - 1, labels
91.
92. def check_folder(log_dir):
93.     if not os.path.exists(log_dir):
94.         os.makedirs(log_dir)

```

6.3 main.py

```

1.  import argparse
2.  from CGAN import CGAN
3.  from utils import check_folder
4.
5.  def parse_args():
6.      parser = argparse.ArgumentParser()
7.      parser.add_argument('--epochs', type=int, default=100, help='The nu
8. mber of epochs to run')
9.      parser.add_argument('--batch_size', type=int, default=64, help='The
10. size of batch')
11.      parser.add_argument('--z_dim', type=int, default=62, help='Dimensio
12. n of noise vector')
13.      parser.add_argument('--dataset_path', type=str, default='./images/'
14. ,
15.                           help='Directory name to save the checkpoints')
16.      parser.add_argument('--save_path', type=str, default='./save/',
17.                           help='Directory name to save the generated imag
18. es')
19.      parser.add_argument('--mode', type=str, default='train',
20.                           help='train or infer')
21.      parser.add_argument('--device', type=str, default='cuda',
22.                           help='train on GPU or CPU')
23.      parser.add_argument('--save_training_img_path', type=str, default='./
24. save/training_img/',
25.                           help='Directory name to save the training image
26. s')
27.      parser.add_argument('--save_testing_img_path', type=str,

```

```

21.     default='./save/testing_img/',
        help='Directory name to save the training image
s')
22.     return parser.parse_args()
23.
24.
25. def main():
26.     args = parse_args()
27.     check_folder(args.dataset_path)
28.     check_folder(args.save_path)
29.     gan = CGAN(args.dataset_path,
30.                args.save_path,
31.                args.epochs,
32.                args.batch_size,
33.                args.z_dim,
34.                args.device,
35.                args.mode)
36.     if args.mode == "train":
37.         check_folder(args.save_training_img_path)
38.         gan.train()
39.     else:
40.         check_folder(args.save_testing_img_path)
41.         gan.infer()
42.
43. if __name__ == "__main__":
44.     main()

```

55000step的结果：

green hair red eyes



white hair black eyes



black hair orange eyes



blonde hair pink eyes



gray hair aqua eyes



6.4 GUI.py

```
1.  import torch
2.  import tkinter as tk
3.  import os
4.  import numpy as np
5.  from tkinter import ttk
6.  import scipy.misc
7.  from PIL import Image, ImageTk
8.
9.
10. win = tk.Tk()
11. win.title('Conditional-GAN-GUI')
12. win.geometry('200x200')
13.
14. device = 'cuda' if torch.cuda.is_available() else 'cpu'
15.
16. generate = torch.load("save/generate.t7").to(device)
17. generate.eval()
18.
19. def create():
20.     z = np.random.normal(0, np.exp(-1 / np.pi), [1, 62])
21.     line = combolist1.get() + ' ' + combolist2.get()
22.     tag_dict = ['orange hair', 'white hair', 'aqua hair', 'gray hair',
```

```

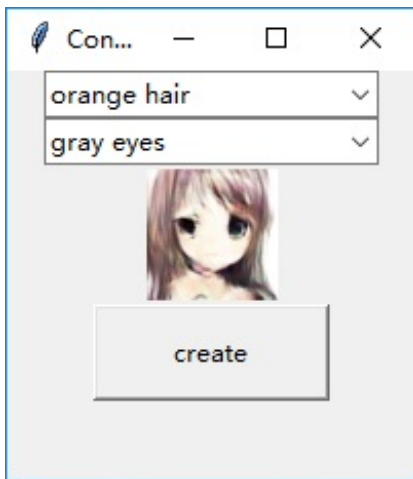
23.         'green hair', 'red hair', 'purple hair',
24.         'pink hair', 'blue hair', 'black hair', 'brown hair',
25.         'blonde hair',
26.         'gray eyes', 'black eyes', 'orange eyes', 'pink eyes', 'yel
low eyes',
27.         'aqua eyes', 'purple eyes', 'green eyes', 'brown eyes', 're
d eyes', 'blue eyes']
28.     y = np.zeros((1, len(tag_dict)))
29.
30.     for i in range(len(tag_dict)):
31.         if tag_dict[i] in line:
32.             y[0][i] = 1
33.
34.     image = generate(torch.from_numpy(z).float().to(device), torch.from
_numpy(y).float().to(device)).to("cpu").detach().numpy()
35.     image = np.squeeze(image)
36.     image = image.transpose(1, 2, 0)
37.     scipy.misc.imsave('anime.png', image)
38.     img_open = Image.open('anime.png')
39.     img = ImageTk.PhotoImage(img_open)
40.     label.configure(image=img)
41.     label.image=img
42.
43. # def go(*args):    #处理事件, *args表示可变参数
44. #     print(comboxlist.get()) #打印选中的值
45.
46. comvalue1=tk.StringVar() #窗体自带的文本, 新建一个值
47. combolist1=ttk.Combobox(win,textvariable=comvalue1) #初始化
48. combolist1["values"]=('orange hair', 'white hair', 'aqua hair', 'gray
hair', 'green hair', 'red hair',
49.                       'purple hair', 'pink hair', 'blue hair', 'black ha
ir', 'brown hair', 'blonde hair')
50. combolist1.current(0)    #选择第一个
51. # combolist.bind("<<ComboboxSelected>>",go)    #绑定事件, (下拉列表框被选中时
, 绑定go()函数)
52. combolist1.pack()
53.
54. comvalue2=tk.StringVar()
55. combolist2=ttk.Combobox(win,textvariable=comvalue2)
56. combolist2["values"]=('gray eyes', 'black eyes', 'orange eyes', 'pink
eyes', 'yellow eyes',
57.                       'aqua eyes', 'purple eyes', 'green eyes', 'brown e
yes', 'red eyes', 'blue eyes')
58. combolist2.current(0)

```

```

58.     # combolist.bind("<<ComboboxSelected>>",go)
59.     combolist2.pack()
60.
61.     bm = tk.PhotoImage(file='anime.png')
62.     label = tk.Label(win, image = bm)
63.     label.pack()
64.
65.     b = tk.Button(win,
66.                   text='create',          # 显示在按钮上的文字
67.                   width=15, height=2,
68.                   command=create)        # 点击按钮式执行的命令
69.     b.pack()    # 按钮位置
70.
71.     win.mainloop()

```



7. GAN小技巧

- 1.对真实图片进行归一化，与生成图片分布一样，也就是 $[-1,1]$.
- 2.随机噪声使用高斯分布，不要使用均匀分布，也就是在代码中使用`torch.randn`，而不是`torch.rand`
- 3.初始化权重很有必要，详细见`model.py`中的`weight_init`函数
- 4.在训练时，在鉴别器中产生的noise，生成器也要用这个noise进行参数，这点很重要。我最开始的时候就是鉴别器随机产生noise，生成器也随机产生noise，训练得很不好。
- 5.在训练过程中，很有可能鉴别器的loss等于0（鉴别器太强了，起初我试过减小鉴别器的学习率，但还是会有这个情况，我猜想原因是在某一个batch中，鉴别器恰好将随机噪声产生的图片和真实图片完全区分开，loss为0），导致生成器崩溃（梯度弥散），所以

最好按多少个epoch保存模型，然后在导入模型再训练。个人觉得数据增强和增大batchsize会减弱这种情况的可能性，这个还未实践。

8. 参考

- 1 [李宏毅GAN课程及PPT](#)
- 2 [DCGAN paper](#)
- 3 [chenyuntc](#)