One Day One GAN_keras版

Hi ,my name is Chen Yang ,I am a sophomore in ocean university of China .I do some scientific research in my spare time. Based on the current hot direction of artificial intelligence, I hope to share my research progress with **Generative adversarial network**

嗨，我的名字是陈扬，我是中国海洋大学的二年级学生。我在业余时间做了一些科学研究。基于当前人工智能的热点方向，我希望与**生成对抗网络分享我的研究进展**

前言

ODOG,顾名思义就我我希望能每天抽出一个小时的时间来讲讲到目前为止,GAN的前沿发展和研究,笔者观察了很多深度学习的应用,特别是在图像这一方面,GAN已经在扮演着越来越重要的角色,我们经常可以看到老黄的NVIDIA做了各种各样的application,而且其中涉及到了大量GAN的理论及其实现,再者笔者个人也觉得目前国内缺少GAN在pytorch,keras,tensorflow等主流的框架下的实现教学.

我的老师曾经对我说过:"**深度学习是一块未知的新大陆,它是一个大的黑箱系统,而GAN则是黑箱中的黑箱,谁要是能打开这个盒子,将会引领一个新的时代**"

DAY1-GAN的前身今世

我们先来看一段wikipedia上对GAN的定义:

> **生成对抗网络**（英语：**G**enerative **A**dversarial **N**etwork，简称GAN）是非监督式学习的一种方法，通过让两个神经网络相互博弈的方式进行学习。该方法由伊恩·古德费洛等人于2014年提出。[1]

> 生成对抗网络由一个生成网络与一个判别网络组成。生成网络从潜在空间（latent space）中随机采样作为输入，其输出结果需要尽量模仿训练集中的真实样本。判别网络的输入则为真实样本或生成网络的输出，其目的是将生成网络的输出从真实样本中尽可能分辨出来。而生成网络则要尽可能地欺骗判别网络。两个网络相互对抗、不断调整参数，最终目的是使判别网络无法判断生成网络的输出结果是否真实。[2][1][3]

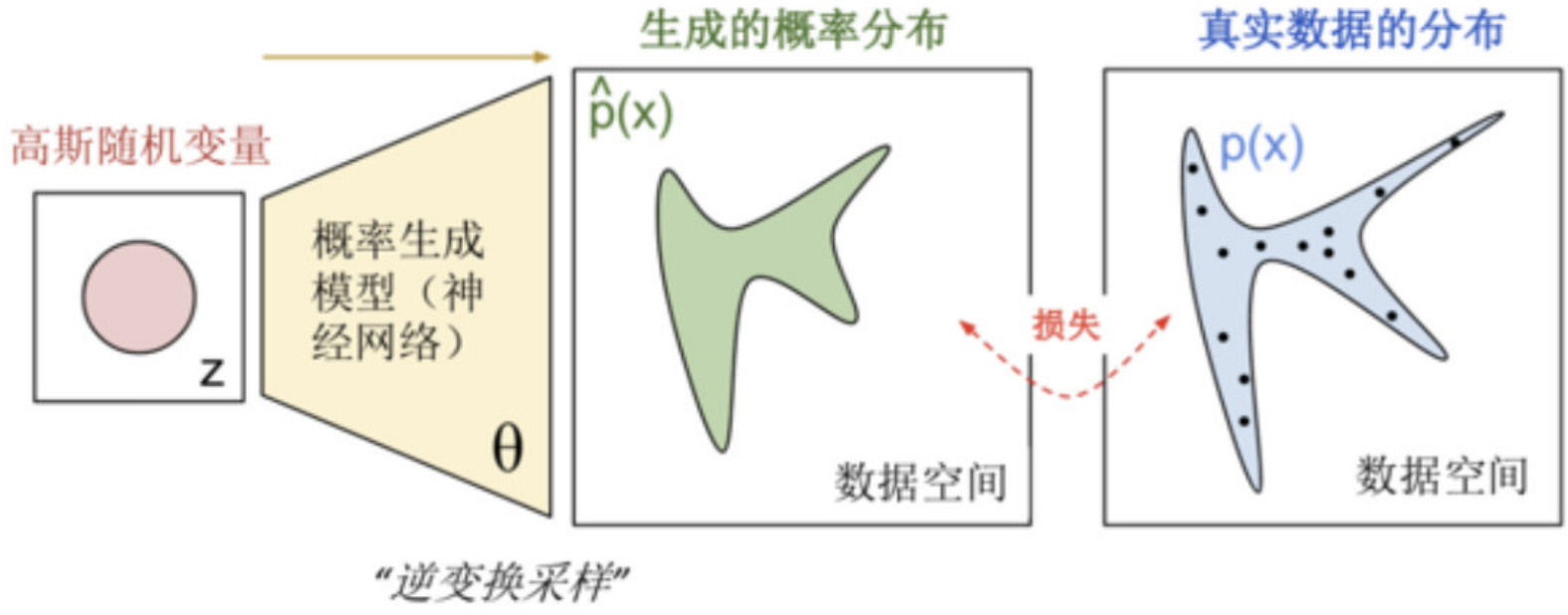
> 生成对抗网络常用于生成以假乱真的图片。[4]此外，该方法还被用于生成视频[5]、三维物体模型[6]等。

简单点说GAN做了一件什么事情呢?

我觉得学过基础的概率的同学肯定都多少能理解这句话:**GAN就是把一个随机变量的分布映射到我们给定的数据集的分布**

是不是听起来还是有些拗口,哎呦你可能觉得哎,这个小陈啊,我这个出来工作有点时间了,对你说的这些偏学术的"鬼话"不太听得懂了,那要不然就放弃吧　.....

那怎么行呢,我这个ODOG虽然说我是假定你已经懂了DL和ML的基本概念,掌握一点点数学分析和概率统计的基础,但至少我觉得是有义务在前几天说的ODOG中,做到足够的照顾广大初学者的,毕竟我也不是啥阳春白雪,那么我们慢慢来看看刚才那句话究竟是什么意思?

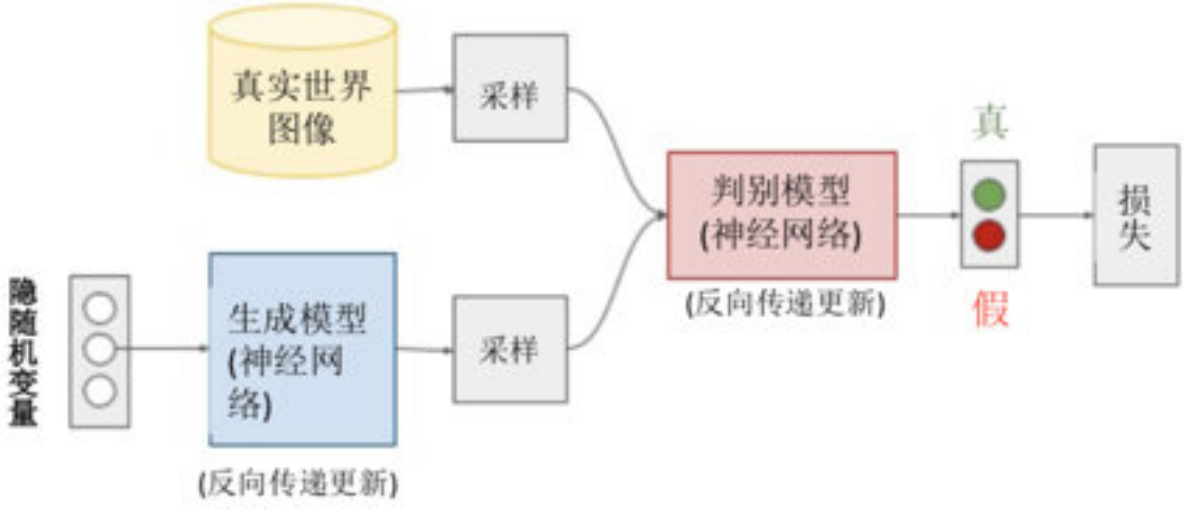


生成模型: $\mathbf{x} = G(\mathbf{z}; \theta^{(G)})$

- 需可导
- 不要求可逆
- 给定z, x可条件高斯分布

简单的来说,就给定一个噪声z的输入,通过生成器的变换把噪声的概率分布空间尽可能的去拟合真实数据的分布空间.

基本框架:



(在最初的解释中:笔者注1)在这里,我们把生成器看的目标看成是要"以假乱真",判别器的目标是要"明辨真假".

核心公式:

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

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

这个公式我们要分成两个部分来看:
先看前半部分:

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

这个公式的意思是,先看加号前面\Epsilon_{x\sim p_{data}(x)}[\log D(x)]+\Epsilon_{z\sim p_z(z)}[log(1-D(G(z)))]

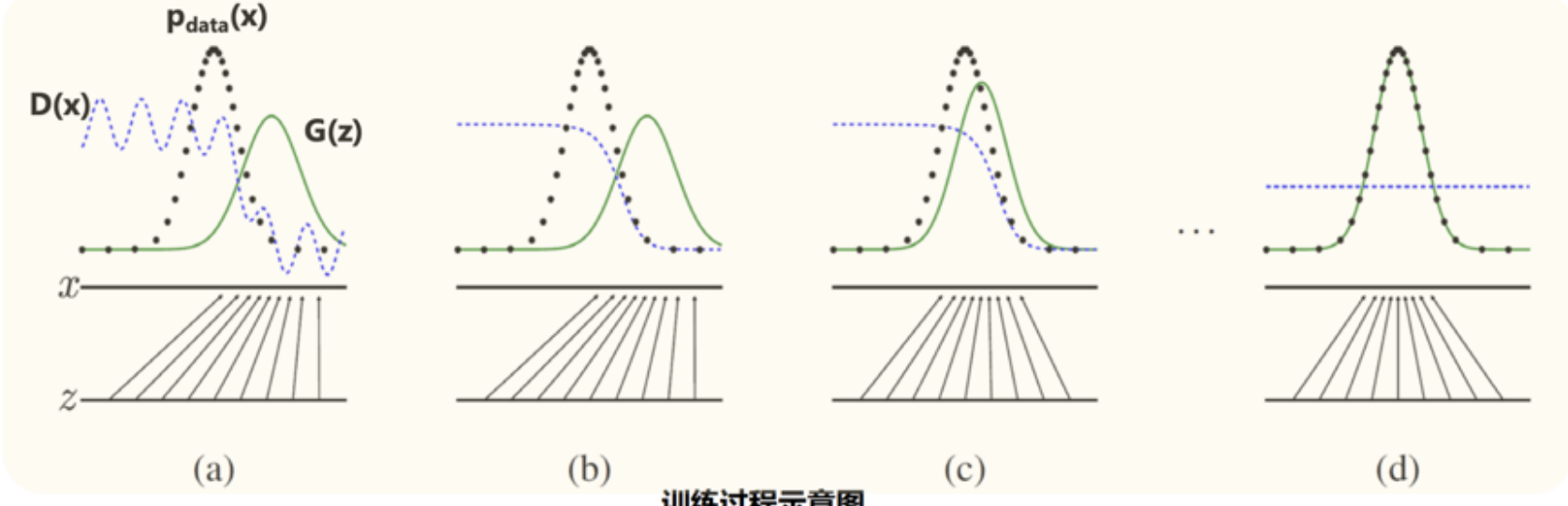
,我们希望D最大,所以log(D(x))应该最大,意味着我的判别器可以很好的识别出,真实世界图像是"true",在看加号后面\Epsilon_{z\sim p_z(z)}[\log(1-D(G(z)))],要让log尽可能的大,需要的是D(G(z))尽可能的小,意味着我们生成模型的图片应该尽可能的被判别模型视为"FALSE".

再看后半部分部分

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

,

我们应该让G尽可能的小,加号前面的式子并没有G,所以无关,在看加号后面的式子\Epsilon_{z\sim p_z(z)}[\log(1-D(G(z)))],要让G尽可能地小,就要D(G(Z))尽可能的大,也就是说本来就一张→噪声生成的图片,判别器却被迷惑了,以为是一张真实世界图片.这就是所谓的以假乱真.



训练过程示意图

代码实现

生成器:

```
    def build_generator(self):
        model = Sequential()
        model.add(Dense(256, input_dim=self.latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(512))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(1024))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(np.prod(self.img_shape), activati
        model.add(Reshape(self.img_shape))

        model.summary()

        noise = Input(shape=(self.latent_dim,))
        img = model(noise)

        return Model(noise, img)
```

生成器的输入是一个100维服从高斯分布的向量,输出是一张 $28 * 28 * 1$ 的图片

```
_________________________________________________________________
Layer (type)                 Output Shape              Pa
=================================================================
input_2 (InputLayer)         (None, 100)               0
_________________________________________________________________
sequential_2 (Sequential)    (None, 28, 28, 1)         14
=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
```

判別器

```
    def build_discriminator(self):

        model = Sequential()

        model.add(Flatten(input_shape=self.img_shape))
        model.add(Dense(512))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dense(256))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dense(1, activation='sigmoid'))
        model.summary()

        img = Input(shape=self.img_shape)
        validity = model(img)

        return Model(img, validity)
```

判别器的输入是一张$28*28*1$的图片和一个一维的真假标签,1代表是真实世界图片,0代表的的生成模型生成的图片.

```
_________________________________________________________________
Layer (type)                 Output Shape              Pa
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
sequential_1 (Sequential)    (None, 1)                 53
=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
```

注意了!在SGAN(2104)中,作者并没有用卷积池化等操作,他只是用了最简单的full connection全连接层.

训练

定义模型

```
def __init__(self):
        self.img_rows = 28
        self.img_cols = 28
        self.channels = 1
        self.img_shape = (self.img_rows, self.img_cols, s
        self.latent_dim = 100

        optimizer = Adam(0.0002, 0.5)

        # Build and compile the discriminator
        self.discriminator = self.build_discriminator()
        self.discriminator.summary()
        self.discriminator.compile(loss='binary_crossentr
            optimizer=optimizer,
            metrics=['accuracy'])
```

```
        # Build the generator
        self.generator = self.build_generator()
        self.generator.summary ()
        # The generator takes noise as input and generates
        z = Input(shape=(self.latent_dim,))
        img = self.generator(z)

        # For the combined model we will only train the g
        self.discriminator.trainable = False

        # The discriminator takes generated images as inp
        validity = self.discriminator(img)
        # The combined model  (stacked generator and disc
        # Trains the generator to fool the discriminator
        self.combined = Model(z, validity)
        self.combined.summary()
        self.combined.compile(loss='binary_crossentropy',
```

判别器discriminator只训练判别器的参数;生成器的训练是把生成器和判别器两个网络连在一起,但是冻结判别器的学习率,一起组成combined.用的都是binary_crossentropy二分类的交叉熵作为损失函数.

```
    # Adversarial ground truths
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for epoch in range(epochs):

        # ---------------------
        #  Train Discriminator
        # ---------------------

        # Select a random batch of images
        idx = np.random.randint(0, X_train.shape[0], batch
        imgs = X_train[idx]

        noise = np.random.normal(0, 1, (batch_size, self.

        # Generate a batch of new images
        gen_imgs = self.generator.predict(noise)

        # Train the discriminator
        d_loss_real = self.discriminator.train_on_batch(im
        d_loss_fake = self.discriminator.train_on_batch(ge
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # ---------------------
        #  Train Generator
        # ---------------------
```

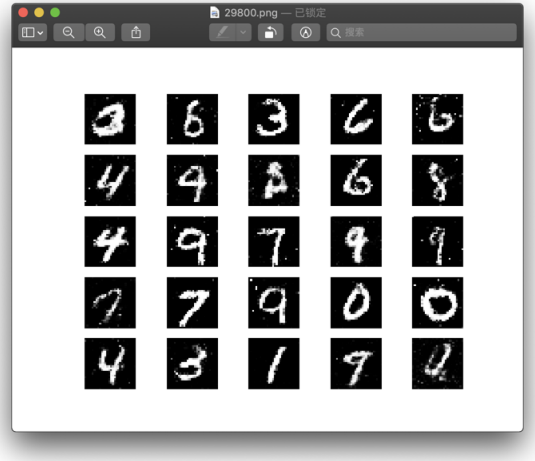
先加载数据集,然后每一次训练从数据集里面随机选取batchsize张图片进行训练,训练的时候,真实图片对应的标签是valid=1,生成器生成的图片对应的标签是fake=0;

训练的时候,先训练dloss,dloss由真实世界图片和生成图片以及其标签进行训练.在训练判别器的时候,真实世界图片对应真实的标签valid,生成的图片对应fake标签,也就是让判别器"明辨真假"的过程.在训练生成器的时候,我们输入高斯噪声和ground truths(中文翻译叫标注),等于是告诉生成对抗网络,我给你一个"假的"图片,但是是"真的"标签,也就是我们让生成器以假乱真的过程.不断的在"明辨真假"和"以假乱真"的两个过程不断迭代训练,最终,生成器可以很好的"以假乱真",判别器可以很好的"明辨真假".当我们把生成器的图片给"人"看的时候,人就会被"以假乱真"了.

在服务器上训练

在训练了30000epoch后

```
30000 [D loss: 0.693933, acc.: 57.81%] [G loss: 0.853226]
```



我们看得到,判别器分辨真假的能力接近1/2,相当于已经被生成器以假乱真了.