利用生成对抗网络（GAN）生成各种类别的数据集

信息与通信工程　蒋凌岳
2016010903022

1. 背景

深度学习(DL, Deep Learning)是机器学习(ML, Machine Learning)领域中一个新的研究方向，它被引入机器学习使其更接近于最初的目标——人工智能(AI, Artificial Intelligence)。深度学习是学习样本数据的内在规律和表示层次，这些学习过程中获得的信息对诸如文字，图像和声音等数据的解释有很大的帮助。它的最终目标是让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。 深度学习是一个复杂的机器学习算法，在语音和图像识别方面取得的效果，远远超过先前相关技术。

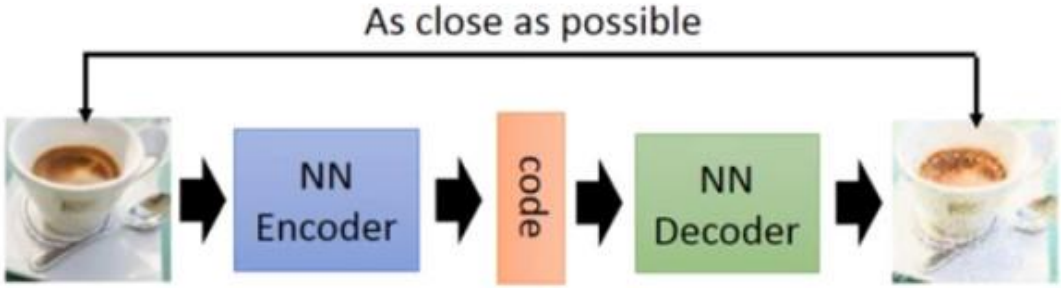
然而，深度学习需要大量的数据集，当前面临的问题一是数据集的数量不够，二是数据集的质量不够，或是图片模糊，或是标签错误。

因此获得大量高质量的数据集变得尤为重要。而 GAN 作为无监督学习的新技术，在训练或生成样本期间，不需要任何马尔科夫链或展开的近似推理网络。实验通过对生成的样品的定性和定量评估证明了本框架的潜力。

本文期望通过少量的图片数据，比如猫，狗，建筑，头像，生成大量的、高质量的、在原始数据中从未出现的猫、狗，建筑，头像。

2. Gan 详述

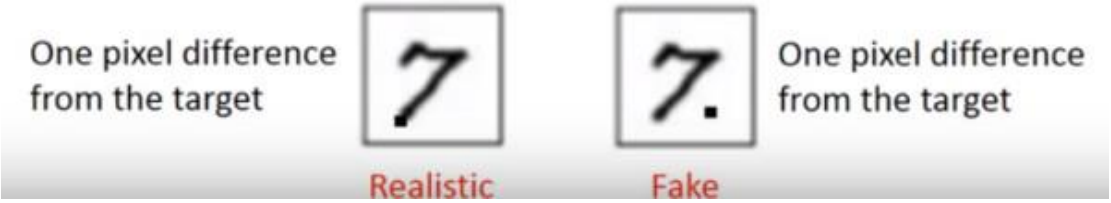
什么是生成（generation）？就是模型通过学习一些数据，然后生成类似的数据。让机器看一些动物图片，然后自己来产生动物的图片，这就是生成。以前就有很多可以用来生成的技术了，比如 auto-encoder（自编码器），结构如下图：



你训练一个 encoder，把 input 转换成 code，然后训练一个 decoder，把 code 转换成一个 image，然后计算得到的 image 和 input 之间的 MSE（mean square error），训练完这个 model 之后，取出后半部分 NN Decoder，输入一个随机的 code，就能 generate 一个 image。

但是 auto-encoder 生成 image 的效果，当然看着很别扭啦，一眼就能看出真假。所以后来还提出了比如 VAE 这样的生成模型，我对此也不是很了解，在这就不细说。

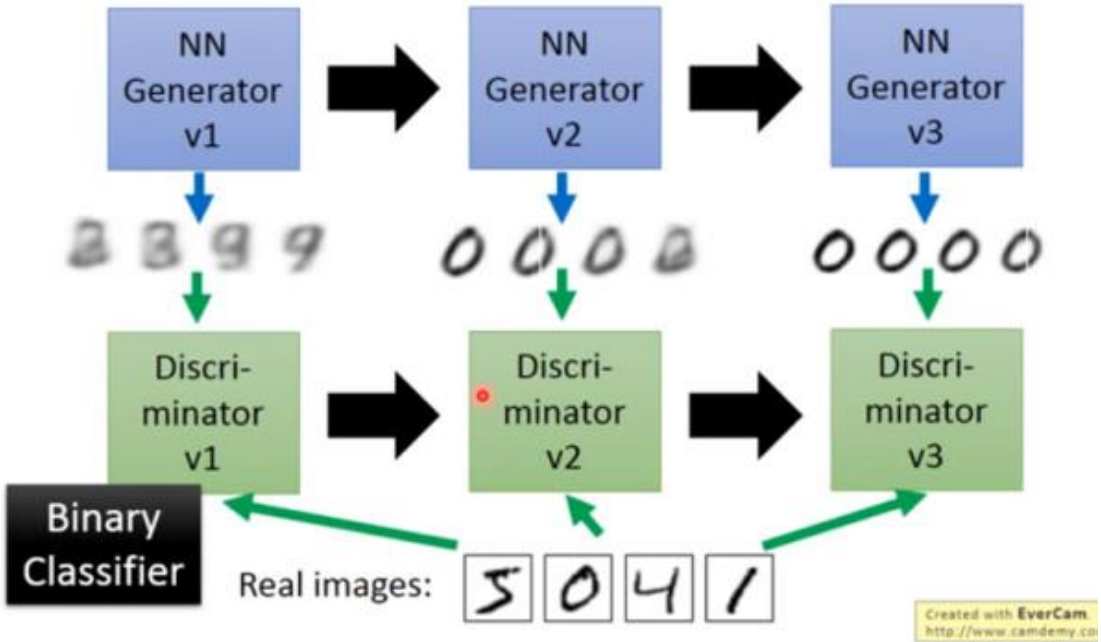
上述的这些生成模型，其实有一个非常严重的弊端。比如 VAE，它生成的 image 是希望和 input 越相似越好，但是 model 是如何来衡量这个相似呢？model 会计算一个 loss，采用的大多是 MSE，即每一个像素上的均方差。loss 小真的表示相似嘛？



比如这两张图，第一张，我们认为是好的生成图片，第二张是差的生成图片，但是对于上述的 model 来说，这两张图片计算出来的 loss 是一样大的，所以会认为是一样好的图片。

这就是上述生成模型的弊端，用来衡量生成图片好坏的标准并不能很好的完成想要实现的目的。于是就有了下面要讲的 GAN。

大名鼎鼎的 GAN 是如何生成图片的呢？首先大家都知道 GAN 有两个网络，一个是 generator，一个是 discriminator，从二人零和博弈中受启发，通过两个网络互相对抗来达到最好的生成效果。流程如下：



开发者自述：我是这样学习 GAN 的

主要流程类似上面这个图。首先，有一个一代的 generator，它能生成一些很差的图片，然后有一个一代的 discriminator，它能准确的把生成的图片，和真实的图片分类，简而言之，这个 discriminator 就是一个二分类器，对生成的图片输出 0，对真实的图片输出 1。
接着，开始训练出二代的 generator，它能生成稍好一点的图片，能够让一代的 discriminator 认为这些生成的图片是真实的图片。然后会训练出一个二代的 discriminator，它能准确的识别出真实的图片，和二代 generator 生成的图片。以此类推，会有三代，四代。。。n 代的 generator 和 discriminator，最后 discriminator 无法分辨生成的图片和真实图片，这个网络就拟合了。
这就是 GAN，运行过程就是这么的简单。这就结束了嘛？显然没有，下面还要介绍一下 GAN 的原理。
首先我们知道真实图片集的分布 Pdata(x)，x 是一个真实图片，可以想象成一个向量，这个向量集合的分布就是 Pdata。我们需要生成一些也在这个分布内的图片，如果直接就是这个分布的话，怕是做不到的。
我们现在有的 generator 生成的分布可以假设为 PG(x;θ)，这是一个由 θ 控制的分布，θ 是这个分布的参数（如果是高斯混合模型，那么 θ 就是每个高斯分布的平均值和方差）
假设我们在真实分布中取出一些数据，{x1, x2, … , xm}，我们想要计算一个似然 PG(xi; θ)。
对于这些数据，在生成模型中的似然就是

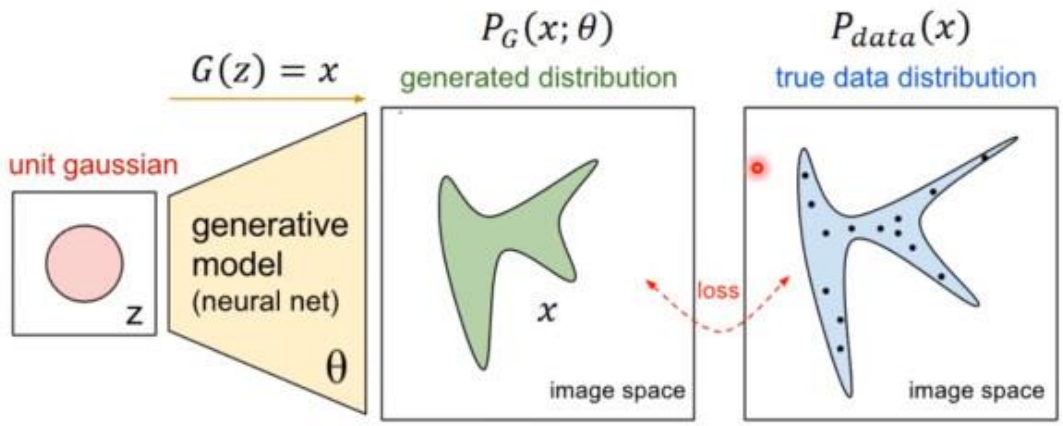
$$L = \prod_{i=1}^{m} P_G(x^i;\theta)$$

我们想要最大化这个似然，等价于让 generator 生成那些真实图片的概率最大。这就变成了一个最大似然估计的问题了，我们需要找到一个 θ* 来最大化这个似然。

$$\begin{aligned}
\theta^* &= arg \ \max_{\theta} \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= arg \ \max_{\theta} \ log \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= arg \ \max_{\theta} \sum_{i=1}^{m} log P_G(x^i;\theta) \\
&\approx arg \ \max_{\theta} \ E_{x\sim P_{data}}[log P_G(x;\theta)] \\
&= arg \ \max_{\theta} \int_x P_{data}(x) log P_G(x;\theta) dx - \int_x P_{data}(x) log P_{data}(x) dx \\
&= arg \ \max_{\theta} \int_x P_{data}(x)(log P_G(x;\theta) - log P_{data}(x)) dx \\
&= arg \ \min_{\theta} \int_x P_{data}(x) log \frac{P_{data}(x)}{P_G(x;\theta)} dx \\
&= arg \ \min_{\theta} \ KL(P_{data}(x)||P_G(x;\theta))
\end{aligned}$$

寻找一个 θ* 来最大化这个似然，等价于最大化 log 似然。因为此时这 m 个数据，是从真实分布中取的，所以也就约等于，真实分布中的所有 x 在 PG 分布中的 log 似然的期望。真实分布中的所有 x 的期望，等价于求概率积分，所以可以转化成积分运算，因为减号后

面的项和 θ 无关，所以添上之后还是等价的。然后提出共有的项，括号内的反转，max 变 min，就可以转化为 KL divergence 的形式了，KL divergence 描述的是两个概率分布之间的差异。
所以最大化似然，让 generator 最大概率的生成真实图片，也就是要找一个 θ 让 PG 更接近于 Pdata。
那如何来找这个最合理的 θ 呢？我们可以假设 PG(x; θ) 是一个神经网络。
首先随机一个向量 z，通过 G(z)=x 这个网络，生成图片 x，那么我们如何比较两个分布是否相似呢？只要我们取一组 sample z，这组 z 符合一个分布，那么通过网络就可以生成另一个分布 PG，然后来比较与真实分布 Pdata。
大家都知道，神经网络只要有非线性激活函数，就可以去拟合任意的函数，那么分布也是一样，所以可以用一直正态分布，或者高斯分布，取样去训练一个神经网络，学习到一个很复杂的分布。



如何来找到更接近的分布，这就是 GAN 的贡献了。先给出 GAN 的公式：

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

这个式子的好处在于，固定 G，max V(G,D) 就表示 PG 和 Pdata 之间的差异，然后要找一个最好的 G，让这个最大值最小，也就是两个分布之间的差异最小。

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

表面上看这个的意思是，D 要让这个式子尽可能的大，也就是对于 x 是真实分布中，D(x) 要接近与 1，对于 x 来自于生成的分布，D(x) 要接近于 0，然后 G 要让式子尽可能的小，让来自于生成分布中的 x，D(x) 尽可能的接近 1。
现在我们先固定 G，来求解最优的 D：

- Given x, the optimal D* maximizing

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

a D b D

- Find D* maximizing: $f(D) = alog(D) + blog(1-D)$

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

$$a \times \frac{1}{D^*} = b \times \frac{1}{1-D^*} \qquad a \times (1 - D^*) = b \times D^*$$

$$a - aD^* = bD^*$$

$$D^* = \frac{a}{a+b} \Rightarrow \qquad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

对于一个给定的 x，得到最优的 D 如上图，范围在 (0,1) 内，把最优的 D 带入

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

可以得到：

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

$$= E_{x \sim P_{data}}\left[log\frac{P_{data}(x)}{P_{data}(x) + P_G(x)}\right] + E_{x \sim P_G}\left[log\frac{P_G(x)}{P_{data}(x) + P_G(x)}\right]$$

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

$$-2log2$$

JS divergence 是 KL divergence 的对称平滑版本，表示了两个分布之间的差异，这个推导就表明了上面所说的，固定 G。

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

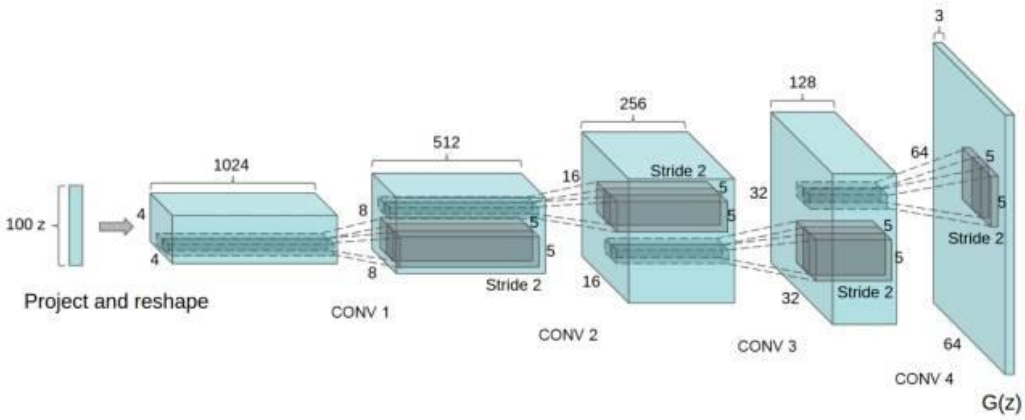
表示两个分布之间的差异，最小值是 -2log2，最大值为 0。
现在我们需要找个 G，来最小化

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

观察上式，当 PG(x)=Pdata(x) 时，G 是最优的。

3. DCGAN

我在实验中采用了较为有稳定有效的 DCGAN。
DCGAN 的全称是 Deep Convolutional Generative Adversarial Networks ,意即深度卷积对抗生成网络，它是由 Alec Radford 在论文 Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks 中提出的。
结构如下：



DCGAN 的生成器网络结构如上图所示，相较原始的 GAN，DCGAN 几乎完全使用了卷积层代替全链接层，判别器几乎是和生成器对称的，从上图中我们可以看到，整个网络没有 pooling 层和上采样层的存在，实际上是使用了带步长（fractional-strided）的卷积代替了上采样，以增加训练的稳定性。

DCGAN 能改进 GAN 训练稳定的原因主要有：

◆ 使用步长卷积代替上采样层，卷积在提取图像特征上具有很好的作用，并且使用卷积代替全连接层。

◆ 生成器 G 和判别器 D 中几乎每一层都使用 batchnorm 层，将特征层的输出归一化到一起，加速了训练，提升了训练的稳定性。（生成器的最后一层和判别器的第一层不加 batchnorm）

◆ 在判别器中使用 leakrelu 激活函数，而不是 RELU，防止梯度稀疏，生成器中仍然采用 relu，但是输出层采用 tanh

◆ 使用 adam 优化器训练，并且学习率最好是 0.0002，（我也试过其他学习率，不得不说 0.0002 是表现最好的了）

```python
def discriminator(inpt,gn_stddev, training=True):
    '''
        input_arguments:
            inpt: the inpt image tensor, [batch, width, length, channels]
            gn_stddev: a scalar, the stddev for the gaussian noise added to the input image,
    '''
    with tf.variable_scope('dis',reuse=tf.AUTO_REUSE):
        channels = 128
        inpt = inpt+tf.random_normal(shape=tf.shape(inpt), mean=0.0, stddev=gn_stddev, dtype=tf.float32)
        out = tf.layers.conv2d(inpt, filters=channels, kernel_size=5, strides=1, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)

        out = tf.layers.conv2d(out, filters=channels*2, kernel_size=5, strides=2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)

        out = tf.layers.conv2d(out, filters=channels*4, kernel_size=5, strides=2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)

        out = tf.layers.conv2d(out, filters=channels*8, kernel_size=5, strides=2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)

        out = tf.layers.flatten(out)
        out = tf.layers.dense(out, 1)
        # out = tf.nn.sigmoid(out)
    return out
```

```python
def generator(inpt,training=True):
    channels = 128
    with tf.variable_scope('gen',reuse=tf.AUTO_REUSE):
        out = tf.layers.dense(inpt, 8*channels*8*8)
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.relu(out)
        out = tf.reshape(out, [-1,8,8,channels*8])# (8,8,1024)

        # out = tf.layers.conv2d(out, channels*8, 5, padding='SAME')
        # out = tf.nn.leaky_relu(out)
        # out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)

        out = tf.layers.conv2d_transpose(out, channels*4, 4, 2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)# (16,16,512)

        out = tf.layers.conv2d_transpose(out, channels*2, 4, 2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)# (32,32,256)

        out = tf.layers.conv2d_transpose(out, channels, 4, 2, padding='SAME')
        out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        out = tf.nn.leaky_relu(out)# (64,64,128)

        # out = tf.layers.conv2d(out, channels, 4, padding='SAME')
        # out = tf.layers.batch_normalization(out,epsilon=1e-5,training=training)
        # out = tf.nn.leaky_relu(out)
        out = tf.layers.conv2d(out, 3, 4,padding='SAME')
        out = tf.nn.tanh(out)
    return out
```

4. 实验结果

（1） 猫

DCGAN 生成器在两到三个小时后就可以收敛出非常逼真的图像，只用了 209 个 epoch，但为了让收敛效果更佳，我们需要做适当的调整。你必须为鉴别器与生成器选择单独的学习速率，以避免它们有一方表现过于出色——这需要非常小心的平衡，不过一旦成功，你就会得到收敛。在 64×64 像素的图片中，鉴别器的最佳学习率是 0.00005，生成器的则是 0.0002。这样就不会出现模型崩溃，得到真正可爱的图片了！

(2)　二次元头像

5. 总结

GAN 在图像生成上取得了巨大的成功，这无疑取决于 GAN 在博弈下不断提高建模能力，最终实现以假乱真的图像生成。

GAN 自 2014 年诞生至今也有 4 个多年头了，大量围绕 GAN 展开的文章被发表在各大期刊和会议，以改进和分析 GAN 的数学研究、提高 GAN 的生成质量研究、GAN 在图像生成上的应用（指定图像合成、文本到图像，图像到图像、视频）以及 GAN 在 NLP 和其它领域的应用。图像生成是研究最多的，并且该领域的研究已经证明了在图像合成中使用 GAN 的巨大潜力。