

GAN & CGAN

目录

GAN	2
一、 理论学习.....	2
1. 理论分析.....	2
2. 创新性分析.....	3
二、 网络搭建.....	3
1. 原理应用.....	3
2. 代码实现.....	4
3. 结果展示.....	5
CGAN.....	7
一、 理论学习.....	7
1. 理论分析.....	7
2. 创新性分析.....	7
二、 网络构建.....	8
1. 代码实现.....	8
2. 结果展示.....	9
GAN&CGAN 实验总结.....	10
参考文献	11

GAN

一、 理论学习

1. 理论分析

对于 GAN, 看过了很多资料以后, 我将其理论核心理解为论文中提出的公式:

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

从该公式出发, 接下来将分成两个归纳出的方面, 来解读一下我理解的生成对抗网络。

1) 对抗博弈性:

公式本身体现的对抗博弈性在于 $\min_G \max_D V(D, G)$ 。也就是说这个公式要先优化 **Discriminator**, 也就是判别器, 然后优化 **Generator**, 也就是生成器。对于判别网络来说, 它需要尽可能区分生成网络产生的虚假样本和真实样本。所以它相当于一般的二分类问题, 它的目标函数就是和基本神经网络相似的损失函数, 所需要的是在训练过程中使损失函数最大 (因为 \log 这里是负数, 和一般神经网络的损失函数是相反的)。

对于生成网络来说, 已经固定判别网络, 前一项的期望值是固定值, 所以要优化的是后一项。所以要

$$\min_G E_{z \sim P_z(z)} [\log(1 - D(G(z)))]$$

这里 $D(x)$ 是属于真实类别的概率, 也就是说 $1 - D(G(z))$ 是假图片被判别器识别为假的概率。最小化该项相当于最大化 $E_{z \sim P_z(z)} [\log D(G(z))]$, 相当于原本的交叉熵函数的优化。

2) 隐式概率模型

这一点在阅读论文的过程中不是很理解, 是参考邱锡鹏老师的《神经网络与深度学习》理解。

隐式概率模型在公式中的体现为 $z \sim P_z(z)$, 这里的 $P_z(z)$ 是低维空间 \mathcal{Z} 中的一个

简单容易采样的分布，通常可以为标准多元正态分布 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ 。在 GAN 中，就可以使用生成网络构建一个映射函数 $G: Z \rightarrow X$ 。利用神经网络强大的拟合能力，使得 $G(\mathbf{z})$ 服从数据分布 $P_{data}(x)$ 。

3) 纳什均衡

论文中给出了最佳的优化结果，并且给出了理论证明。

理想情况下，生成网络可以生成“以假乱真”的样本 $G(\mathbf{z})$ ，最终的理论推导得到 $D(G(\mathbf{z})) = 0.5$ 。

在一开始的时候，我并不理解为什么当 $D(G(\mathbf{z})) = 0.5$ 时是最佳优化结果。按照一般神经网络的思路来理解，应该是 $D(G(\mathbf{z}))$ 越大越好。

在论文中证明了当 $D(G(\mathbf{z})) = 0.5$ 时， $P_{data}(x)$ 和 $P_z(z)$ 同分布，也就是生成分布等于真实分布。此时判别器判断真实样本和生成样本的效果是一样的，这才是“以假乱真”，也就是生成样本和真实样本完全无法区分。

2. 创新性分析

这里是对于 GAN 最初提出的论文做创新性分析。

《神经网络与深度学习》中提到，“GAN 突破了以往的概率模型必须通过最大似然估计来学习参数的限制”。再回头看 GAN 的理论分析，发现其实 GAN 对于概率的依赖不是很强。其用到的概率分布是最常见的正态分布。在论文中的数学推导也不像其他生成模型（比如变分编码器）一样繁琐。

在实现的过程中，我对 GAN 创新点的理解是，它比较好地结合了概率分析和深度神经网络，在两者之中取了折中，而不是全然偏向某一方。在 GAN 中，采用了判别网络代替了概率模型中的最大似然估计，这也是“对抗性”的来源。

二、网络搭建

1. 原理应用

在搭建网络之前，需要将理论落地。所以接下来会从理论的角度阐述网络框

架。

首先，从 GAN 的原理来看，搭建 GAN 最主要的是两种网络，判别网络和生成网络。

1) 判别网络

从上文中的公式可以看出，GAN 的判别网络完全可以采取一般二分类神经网络的结构。可以是卷积神经网络或者就是一般的全连接神经网络。输入为图片，输出为分类结果，也就是输入样本是否是真实样本，损失函数为交叉熵。

2) 生成网络

理论中的“隐式概率”就是在生成网络中使用。

其输入为符合正态分布的 n 维向量，输出为生成的图片。

对于生成网络，优化的目标函数需要通过判别网络生成。也就是说，生成网络这一个网络应该是生成+判别组合而成的网络。

2. 代码实现

具体的代码实现见源项目代码。接下来将记录在代码实现过程中网络的设计。

1) 初始化

在这个部分需要定义一些参数，比如输入判别网络图片的维度以及输入生成网络的随机数维度和优化器。在该实现中，就采用了最常用的 Adam 优化器。

在这个部分最重要的部分定义损失函数和编译网络。

判别网络的损失函数是很好确定的，就是最基础的交叉熵函数，其编译也是直接编译。

对于生成网络，按照上文的分析，需要判别网络的参与，但此时判别网络不需要训练优化。代码中的 `combined` 就是两个网络的结合。

2) 判别网络

这里的判别网络设计得非常简单。

输入层就是图片，首先用 `Flatten` 函数将多维数组或张量转换为一维向量的形式，然后设计了 3 个隐藏全连接层，激活函数为 `ReLU` 函数。最后一层到输出层的激活函数选择了 `sigmoid` 函数。

3) 生成网络

这里的生成网络也比较简单。全都采用全连接网络。

输入层为 n 维符合正态分布的向量，然后是 3 个隐藏全连接层，激活函数也是 `ReLU` 函数。然后是一个 784 维的全连接层，激活函数为 `tanh`。最后是 `reshape` 层，将 784 维的全连接层映射为 `28*28*1` 的图片。

在这两个网络实现的过程中都加了一些优化手段，比如 `dropout` 和 `batchnormalization`。

4) 训练

训练的过程首先要进行的就是图片的归一化处理。


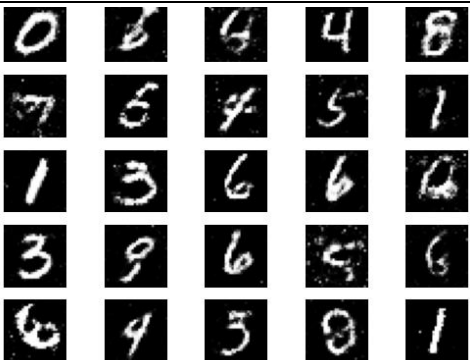
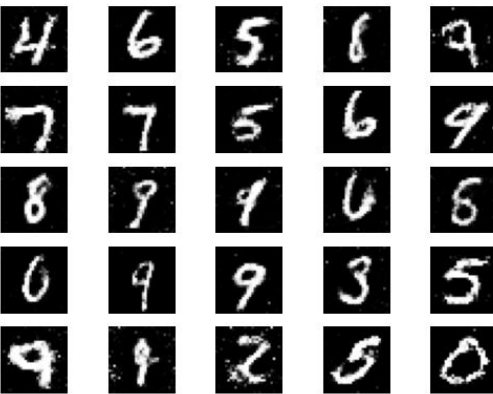
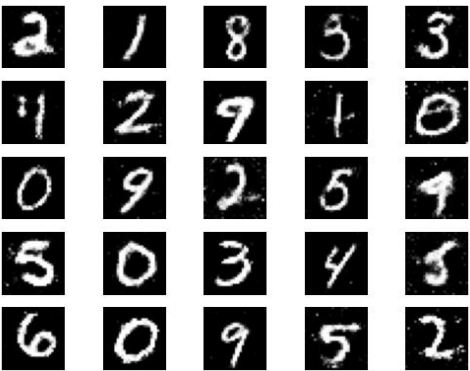
然后就是在每一个 `epoch` 中先后训练判别网络和生成网络，然后输出结果。

5) 结果可视化

每 200 个 `epoch` 保存一次结果，随机展示 25 张生成的图片。

3. 结果展示

相应的 `epoch` 和相应的结果（共 20000 个 `epoch`）

2000	10000
	
16000	20000
	

CGAN

一、 理论学习

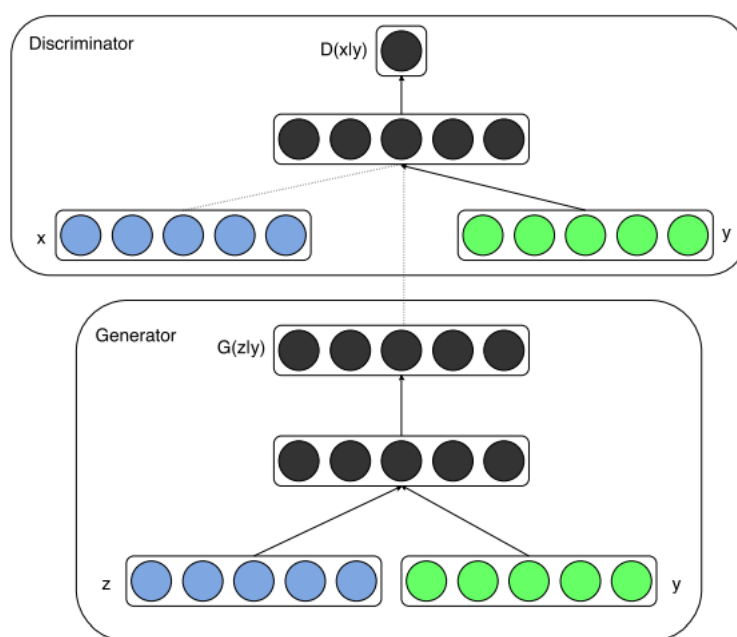
1. 理论分析

CGAN 的理论基础相较于 GAN 来说，从数学上来看，只添加了“条件概率”。其优化公式变成了

$$\min_G \max_D V(D, G) = E_{x \sim P_{data}(x)} [\log D(x|y)] + E_{z \sim P_z(z)} [\log(1 - D(G(z|y)))]$$

而公式中的 y 就是标签。也就是说，在网络搭建的过程中，要把标签考虑进去。对于判别网络，考虑标签也就是需要输入标签和图片，然后判断是否属于真实数据集。对于生成网络，不仅要输入随机向量，还要输入对应的标签，然后生成对应该标签的图片。

论文中也给了简要的网络构建图，非常直观：



2. 创新性分析

CGAN 的创新点主要在于引入了条件输入和条件对抗目标的概念，使得生成网络能够有针对性地生成符合条件的样本。而正由于 CGAN 支持条件输入，

它可以用于处理多模态生成任务。总体来说，CGAN 具有更大的灵活性和可控性。

二、网络构建

1. 代码实现

CGAN 和 GAN 最大的不同是条件概率的应用，体现在网络中就是多输入网络。所以 CGAN 的实现完全基于 GAN 的实现，对输入做了修改。在做输出层的修改的时候，大概有两种思路。第一种是在一个网络中分成两个小网络，然后输出结果并拼接结果。另一种是在输入的时候直接将两个输入处理成一个处理。这里选择了第二种方法，处理的方法是将两个输入变成同维向量并相乘。

1) 判别网络

先进行图片的 `flatten` 操作，再进行标签的转化。然后相乘得到带标签信息的图片输入

```
1. img = Input(shape=self.img_shape) # 输入层
2. flat_img=Flatten(input_shape=self.img_shape)(img)
3. # 输入: 标签
4. # 输出维度是 np.prod(self.img_shape)
5. label = Input(shape=(1,), dtype='int32')
6. # embedding, 将一个正整数转为 n 维的稠密向量
7. label_embedding = Flatten()(Embedding(self.num_classes, np.prod(self.img_shape))(label))
8. model_input = multiply([flat_img, label_embedding]) # 相乘, 获得带标签的图片输入
```

2) 生成网络

也是差不多的处理

```
1. # 输入一: 标签
2. # 输出维度是 self.latent_dim
3. label = Input(shape=(1,), dtype='int32')
4. # embedding
5. # 将一个正整数转为 n 维的稠密向量
```











```

6. label_embedding = Flatten()(Embedding(self.num_classes, self.latent_dim)(label))
7. # 输入二: n 维的随机数
8. # 将正态分布和索引对应的稠密向量相乘, 获得带标签的随机数
9. noise = Input(shape=(self.latent_dim,)) # 输入符合正态分布的 n 维向量
10. model_input = multiply([noise, label_embedding]) # 相乘, 获得带标签的随机数输入

```

2. 结果展示

2000	10000
<p>Digit: 0 Digit: 1 Digit: 2 Digit: 3 Digit: 4</p>  <p>Digit: 5 Digit: 6 Digit: 7 Digit: 8 Digit: 9</p> 	<p>Digit: 0 Digit: 1 Digit: 2 Digit: 3 Digit: 4</p>  <p>Digit: 5 Digit: 6 Digit: 7 Digit: 8 Digit: 9</p> 
16000	20000
<p>Digit: 0 Digit: 1 Digit: 2 Digit: 3 Digit: 4</p>  <p>Digit: 5 Digit: 6 Digit: 7 Digit: 8 Digit: 9</p> 	<p>Digit: 0 Digit: 1 Digit: 2 Digit: 3 Digit: 4</p>  <p>Digit: 5 Digit: 6 Digit: 7 Digit: 8 Digit: 9</p> 

GAN&CGAN 实验总结

首先关于理论学习，GAN 的思想非常简洁，公式理解起来也不太复杂。CGAN 更是和 GAN 一脉相承，很好理解。

其次关于实践，代码实现其实不算复杂。在训练的过程中，参数都是采用的一些参考资料中的经过试验的学习参数。可能是因为搭建的全连接网络，没有使用卷积神经网络一些比较复杂的网络，生成图片其实肉眼可见的有许多瑕疵。我也没有找到一些可以评价生成图片质量的方法，只能依靠训练时的准确率来粗略判断。总体上来说，CGAN 生成的图片在肉眼上要较 GAN 差一点，而且训练的时候非常不稳定。我认为有以下几点原因：

1. 输入层的处理不当，这里的输入是直接将 label 和数据相乘，这样可能会损失一些特征
2. 超参数的选择，CGAN 中的超参并没有修改，和 GAN 基本一样，但是考虑到 CGAN 有 label 这一条件限制，生成网络和判别网络之间的动态平衡变得更加复杂，还需要重新调整合适的超参数

参考文献

GAN:

Ian J. Goodfellow *Generative Adversarial Nets*

邱锡鹏《神经网络与深度学习》P318-P326

CGAN

Mehdi Mirza *Conditional Generative Adversarial Nets*