GAN原理再解析之MMGAN与NSGAN

【参考资料】

[训练GAN，你应该知道的二三事]

GAN通常被定义为一个minimax的过程：

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim P_r(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P_z(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]$$

其中P_r是真实数据分布，P_z是用于生成的随机噪声分布。

在原始的GAN中，判别器要尽可能的将真实样本分类为正例，将生成样本分类为负例，所以判别器需要最小化如下损失函数：

$$-\mathbb{E}_{x\sim P_r}[\log D(x)] - \mathbb{E}_{x\sim P_g}[\log(1-D(x))]$$

其中P_g表示生成数据分布。

作为对抗训练，生成器需要不断将生成数据分布拉到真实数据分布，Ian Goodfellow 首先提出了如下式的生成器损失函数：

$$\mathbb{E}_{x\sim P_g}[\log(1-D(x))] \tag{1}$$

由于在训练初期阶段，生成器的能力比较弱，判别器这时候也比较弱，但仍然可以足够精准的区分生成样本和真实样本，这样$D(x)$就非常接近1，导致$log(1-D(x))$达到饱和，后续网络就很难再调整过来。为了解决训练初期阶段饱和问题，作者提出了另外一个损失函数，即：

$$\mathbb{E}_{x\sim P_g}[-\log D(x)] \tag{2}$$

Ian Goodfellow论文里面已经给出，固定G的参数，我们得到最优的D^*：

$$D^*_G(\mathbf{x}) = \frac{P_r(\mathbf{x})}{P_r(\mathbf{x}) + P_g(\mathbf{x})}$$

也就是说，只有当$P_r = P_g$时，不管是真实样本和生成样本，判别器给出的概率都是 0.5，这个时候就无法区分样本到底是来自于真实样本还是来自于生成样本，这是最理想的情况。

1. MMGAN

使用第一种生成器损失函数，即式（1）的GAN，又称为 Minimax GAN (MMGAN)。在最优判别器D^*下，我们给损失函数加上一个与G无关的项，（1）式变成：

$$\mathbb{E}_{x\sim P_r}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))]$$

注意，该式子其实就是判别器的损失函数的相反数。

把最优判别器D^*带入，可以得到：

$$2JS(P_r||P_g) - 2\log 2 \tag{3}$$

推导过程如下：

$$
\begin{aligned}
&\mathbb{E}_{x\sim P_r}\left[\log D_G^*(x)\right]+\mathbb{E}_{z\sim P_z}\left[\log\left(1-D_G^*(G(z))\right)\right]\\
&=\mathbb{E}_{x\sim P_x}\left[\log D_G^*(x)\right]+\mathbb{E}_{x\sim P_g}\left[\log\left(1-D_G^*(x)\right)\right]\\
&=\mathbb{E}_{x\sim P_r}\left[\log\frac{P_r(x)}{P_r(x)+P_g(x)}\right]+\mathbb{E}_{x\sim P_g}\left[\log\frac{P_g(x)}{P_r(x)+P_g(x)}\right]\\
&=\mathbb{E}_{x\sim P_g}\left[\log\frac{\frac{1}{2}*P_r(x)}{\frac{1}{2}*(P_r(x)+P_g(x))}\right]+\mathbb{E}_{x\sim P_g}\left[\log\frac{\frac{1}{2}*P_g(x)}{\frac{1}{2}*(P_r(x)+P_g(x))}\right]\\
&=\mathbb{E}_{x\sim P_r}\left[\log\frac{P_r(x)}{\frac{1}{2}*(P_r(x)+P_g(x))}-\log 2\right]+\mathbb{E}_{x\sim P_g}\left[\log\frac{P_g(x)}{\frac{1}{2}*(P_r(x)+P_g(x))}-\log 2\right]\\
&=2JS\left(P_r\|P_g\right)-2\log 2
\end{aligned}
$$

因此，在最优判别器的情况下，其实我们在优化两个分布的 JS 散度。当然在训练过程中，判别器一开始不是最优的，但是随着训练的进行，我们优化的目标也逐渐接近JS散度，而问题恰恰就出现在这个 JS 散度上面。一个直观的解释就是**只要两个分布之间的没有重叠或者重叠部分可以忽略不计，那么大概率上我们优化的目标就变成了一个常数，这种情况通过判别器传递给生成器的梯度就是零**，也就是说，生成器不可能从判别器那里学到任何有用的东西，这也就导致了无法继续学习。

2. NSGAN

使用第二种生成器损失函数，即式（2）的GAN，又称为 Non-saturating GAN (NSGAN)。同样在最优判别器下，优化（2）式等价于优化如下：

$$
KL\left(P_g\|P_r\right)-2JS\left(P_r\|P_g\right) \tag{4}
$$

推导：

$$
\begin{aligned}
KL\left(P_g\|P_r\right)&=\mathbb{E}_{x\sim P_g}\left[\log\frac{P_g(x)}{P_r(x)}\right]\\
&=\mathbb{E}_{x\sim P_g}\left[\log\frac{P_g(x)/\left(P_r(x)+P_g(x)\right)}{P_r(x)/\left(P_r(x)+P_g(x)\right)}\right]\\
&=\mathbb{E}_{x\sim P_g}\left[\log\frac{1-D^*(x)}{D^*(x)}\right]\\
&=\mathbb{E}_{x\sim P_g}\log[1-D^*(x)]-\mathbb{E}_{x\sim P_g}\log D^*(x)
\end{aligned}
$$

由上式可得：

$$
\begin{aligned}
\mathbb{E}_{x\sim P_g}\left[-\log D^*(x)\right]&=KL\left(P_g\|P_r\right)-\mathbb{E}_{x\sim P_g}\log[1-D^*(x)]\\
&=KL\left(P_g\|P_r\right)-2JS\left(P_r\|P_g\right)+2\log 2+\mathbb{E}_{x\sim P_r}\left[\log D^*(x)\right]
\end{aligned}
$$

由于上式第三项和第四项与优化G无关，所以舍去不管。

从式（4）中不难发现，我们优化的目标是相互悖论的，因为 KL 散度和 JS 散度的符号相反，优化 KL 是把两个分布拉近，但是优化 -JS 是把两个分布推远，这「一推一拉」就会导致梯度更新非常不稳定。

此外，我们知道 KL 不是对称的，对于生成器无法生成真实样本的情况，KL 对 loss 的贡献非常大，而对于生成器生成的样本多样性不足的时候，KL 对 loss 的贡献非常小。

- 当$P_g(x)\to 0$而$P_r(x)\to 1$时，$P_g(x)\log\frac{P_g(x)}{P_r(x)}\to 0$，对$KL\left(P_g\|P_r\right)$贡献趋近于0；
- 当$P_g(x)\to 1$而$P_r(x)\to 0$时，$P_g(x)\log\frac{P_g(x)}{P_r(x)}\to+\infty$，对$KL\left(P_g\|P_r\right)$贡献趋近正无穷。

这会导致模型不愿意去“尝试”生成具有多样性的样本，因为这样可能会出现上面的第二种情形。

而 JS 是对称的，不会改变 KL 的这种不公平的行为。这就解释了我们经常在训练阶段经常看见两种情况，一个是训练 loss 抖动非常大，训练不稳定；另外一个是即使达到了稳定训练，生成器也大概率上只生成一些安全保险的样本，这样就会导致模型缺乏多样性。