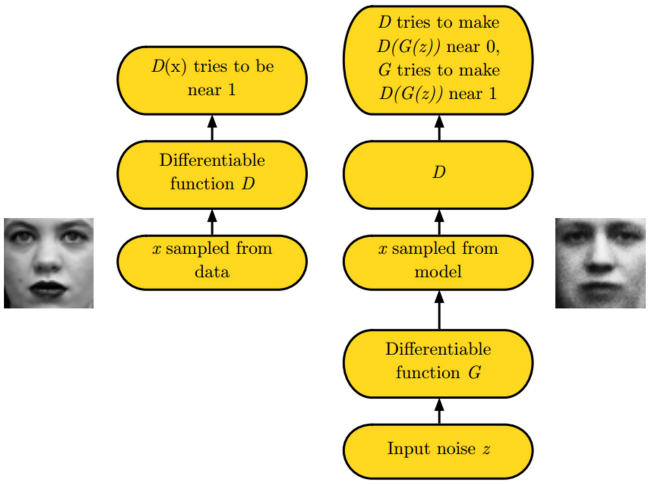
生成模型读书笔记七

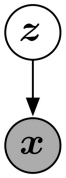
2021年1月11日 21:52

1. **Generative Adversarial Network(GAN) I**
   a. 对抗训练
      i. 模型框架
      GAN提出一种训练方法，称为对抗训练Adversarial Training。在对抗训练中，有两个模型：一个是生成模型，它的作用是生成一些样本，希望这些样本与训练数据来自同一个分布；一个是判别器，它的作用是分辨一个样本是否来自训练数据所在的真实分布。
      ii. 对抗过程
      生成器和判别器的对抗关系在于，生成器想要生成能够骗过判别器的样本，让判别器以为它生成的样本是来自真实分布的；而判别器想要不被生成器骗，要正确分辨出来自真实分布的样本和来自生成器的样本。对抗训练本质上就是minimax game，对抗的双方轮流操作。一开始生成器和判别器都很差，我们会首先固定生成器，训练判别器，给它一些真实数据样本，打上标签real，以及从生成器生成出来的样本，打上标签fake，用监督学习的方法教会它怎么辨别真和假，当判别器学会了之后，我们再固定判别器，训练生成器，这时候我们让生成器生成一些样本，将这些样本输入到判别器，判别器给出真或假的判断，生成器根据判别器的判断来更新参数，使得所生成的样本更像真实的数据。（先固定一个，再去优化另一个，是不是跟前面EM算法的思想很像）
      生成器与判别器的关系就好比制造伪钞的人和警察。一开始，造假者的造假技术很差，警察也没见过伪钞，后来警察见到了伪钞，学会了怎么辨别真钞和伪钞，把造假者给抓了。这时候造假者就知道自己造的伪钞还不够真，出来后苦心钻研技术，造出来更真的钞，骗过了一部分警察，但警察也不是吃素的，经过训练辨别出新技术的伪钞，造假者又被抓了，继续钻研新技术，如此循环往复。。。。最终造假者技术日益精湛，造出警察分辨不出来的伪钞。
   b. 正式定义
   GAN有两个部分组成，一个是生成器generator，一个是判别器discriminator。
      i. 生成器
      生成器以随机噪音z为输入，目的是生成样本x，我们将生成器记为$x=G(z;\theta_g)$其中z是来自预设的分布p(z)的随机输入，一般是均匀分布或者标准高斯分布，函数G是可导的，通常就是用一个神经网络来表示这个函数。G要学习的就是一种分布变换，从简单的输入分布(均匀分布或者高斯分布甚至伯努利分布都可以)到复杂的数据分布的变换。
      ii. 判别器
      判别器是一个二分类器，以样本x为输入，我们将判别器记为$D(x;\theta_d)$，输出表示x来自真实数据分布的概率
      iii. 计算流程图
      每一步训练中，我们需要两类数据: 从p(z)采样的随机输入z和从数据样本中采样的真实数据x
      首先从p(z)中采样z，输入生成器G，G生成样本x，x输入到判别器D，得到判别结果；判别器的输入有两类，采样自真实数据的样本和来自生成器的样本，判别器的最后一层用sigmoid实现，输出[0，1]区间的值，接近0，说明D认为x是生成的假样本，接近1，说明D认为x是来自真实数据的真样本。



iv. 概率图

原始的GAN可以看作是下图所示的结构化概率模型，变量x依赖于隐变量z，这跟VAE的概率图很像啊，只是VAE多了一条从x到z的边来表示它的编码功能。(那VAE的decoder也可以看作是学习从z到x的分布变换？)



v. 应用要求

GAN框架是可以应用在很多模型上的，只要满足生成器和判别器都是可导的条件即可，没有特别的要求。

GAN中对生成器G的设计非常灵活，应用时z并不一定是G的第一层输入，你可以根据你任务的需要在任何地方加入z，也可以将z分成好几个分量，多次输入，输入的形式也有很多种，假设你想在某一hidden layer的输入中加入z，可以将z add，concatenate，或者multiply到该层的输入中。

在看过的很多论文中，GAN框架不是他们模型的主体，甚至没有z noise的输入，而是将对抗训练用于辅助提高模型生成样本的质量，相当于一个refiner，就是在原有的已经work的模型上再加一个判别器，对模型的输出进行判别，相当于对原模型生成样本的质量评分，把adversarial loss加到原loss一同训练，以达到refine目的。

GAN论文原文也提到这点，This framework can yield specific training algorithms for many kinds of model and optimization algorithm. 只要有生成和判别两个对抗角色的存在，就可以叫做对抗训练。而论文原文讨论的，也是我们学习GAN时最常见的一种情况，就是生成器的输入只有随机noise的情况，它只是对抗训练中最简单的一种情况，不要误以为对抗训练的生成器只能是以noise为输入。原文将前面所述的对抗训练框架称为adversarial framework, 而这种输入是noise的情况称为adversarial nets，也就是generative adversarial net的来源。这里讨论的以及上面概率图所表示的也是这种最简单的情况。

c. 损失函数

首先来分析判别器。判别器是一个二分类的分类器，输出是样本x属于真实分布的概率。对于二分类的损失，自然就想到了用分类问题的标配--交叉熵损失。真样本(x~p_{data})的标签为1，假样本(x~p_g)的标签为0，于是我们可以写出判别器D的损失函数(期望形式)

$$J(D) = min - E_{x\sim p_{data}}[\log D(x)] - E_{z\sim p(z)}[\log(1 - D(G(z))]$$

对于生成器，它的目标是要生成的样本G(z)骗过判别器，也即希望判别器分类错误，所以生成器与判别器的目标相反

$$J(G) = \max - E_{z\sim p(z)}[\log(1 - D(G(z))]$$

通常我们会把两个目标写到一起，体现出来生成器和对抗器之间是minmax game的关系，并且为了简洁，把负号去掉，最终形式为

$$V(G, D) = \min_{\theta_g} max_{\theta_d} E_{x\sim p_{data}}[\log(D(x))] + E_{z\sim p(z)}[\log(1 - D(G(z))]$$

现在来分析一下V(G, D)的最优解会是什么。G和D的损失函数中都包含了对方，但是在优化的时候，他们又只能调整自己的参数而没法控制另一个网络的参数，且两者对于损

失函数的优化目标相反，因此在其中一个调整参数的过程中，会导致另一方损失函数值的变化，这就是minimax game。一般优化问题的最优解是一个局部极大值或者极小值，而对于minimax game，最优解既不是极大值也不是极小值，而是损失函数V(G, D)的一个saddle point鞍点，它是θ_g方向上的极小值而同时又是θ_d方向上的极大值。(薯片就是一个具有鞍点的形状)。

d. 判别器与生成器的最优解

上面分析了模型整体的最优解是空间中的一个鞍点，现在来分析生成器和判别器各自的最优解，这将有助于我们了解G和D到底在学什么。

根据训练的先后次序，先来看判别器的最优解。训练判别器时，固定G，优化D

$$\begin{aligned}\mathrm{D} &= \arg max_D\, E_{x\sim p_{data}}[\log(D(x))] + E_{z\sim p(z)}[\log(1-\mathrm{D}(\mathrm{z}))] \\ &= argmax_D\, E_{x\sim p_{data}}[\log(D(x))] + E_{x\sim p_g}[\log(1-\mathrm{D}(\mathrm{x}))] \\ &= argmax_D \int p_{data}(x) log\, \mathrm{D}(\mathrm{x})\, dx + \int p_g(x) \log(1-D(x))dx\end{aligned}$$

要求使得上式取最大值的D，那就对上式求导，使导数等于0，求D，于是有

$$\frac{\partial}{\partial D}\int p_{data}(x) log\, \mathrm{D}(\mathrm{x})\, dx + \int p_g(x) \log(1-D(x))dx = 0$$

$$\frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1-D(x)} = 0$$

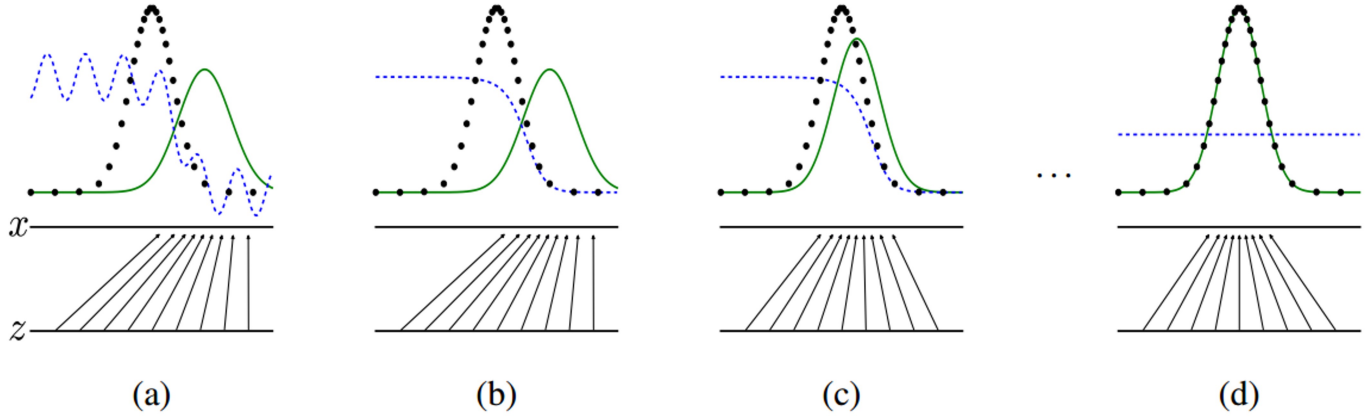
于是D(x)的最优解是

$$D^*(\mathrm{x}) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$$

因此D(x)实际上估计的是数据真实分布和生成分布之间的比率$\frac{p_{data}}{p_g}$。

说明：这里可能会有疑问，不是说GAN不对分布建模吗，怎么这里又出来了个生成分布呢？GAN是不显式地对分布建模，不求解概率密度函数，但是它在隐式地学习分布，它所生成的样本形成了一个分布，所以它相当于隐式地定义了一个分布，这种"定义"不是函数形式上的，而是由样本定义的，这个分布就是这里说的生成分布。

下图展示了GAN训练过程中判别器和生成器的一个理想变化状态



上图中，下半部分的x，z横线表示G学习从z空间到x空间的映射，上半部分的绿色实线表示生成器的分布p_g，黑色虚线表示真实数据分布$\mathrm{p_{data}}$，蓝色虚线表示判别器D(x)，它在$p_{data} > p_g$的地方概率大，在$p_g > p_{data}$的地方概率小。考虑模型将要收敛的某个时刻，如图a，生成器分布与真实分布之间还有一定距离，判别器也不是一个最佳的判别器，经过交替训练的第一步: 固定G，训练D，我们把D训练好了，就变成了图b，然后轮到固定D，训练G，G根据损失函数往真实分布靠，如图c，理想状态下，经过训练，G与真实分布完美重合，模型收敛，这时候D得到了它的最优解$\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$，而$p_{data} = p_g$，所以成了一条直线，处处为1/2。

接着，我们来分析生成器G。在minimax game的每次训练中，我们会先训练D，假设我们每次都把D训练得很好，取得了基于当前G的最优解，那么接下来训练G的过程中，G要学习的是什么呢？将上面求出的$\mathrm{D}^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$代入到损失函数中，可以得到

$$\begin{aligned}C(G) &= \max_D V(G,D) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1-D_G^*(G(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1-D_G^*(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}}\left[\log\frac{p_{\mathrm{data}}(\boldsymbol{x})}{P_{\mathrm{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\frac{p_g(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}\right]\end{aligned}$$

将上式稍作变形，给两个分母都乘以1/2除以1/2，把log展开成加减形式再把分母除的1/2拿到外面，就可以得到下式

$$= \mathbb{E}_{X\sim p_{data}}\left[log\frac{p_{data}(x)}{\frac{1}{2}\left(p_{data}(x)+p_g(x)\right)}\right]+$$
$$\mathbb{E}_{x\sim p_g}\left[log\frac{p_g(x)}{\frac{1}{2}\left(p_{data}(x)+p_g(x)\right)}\right]-2\log 2$$

这里要介绍一个东西叫JS散度，是KL散度的对称版本

$$JS(p\|q)=\frac{1}{2}KL(p\|\frac{p+q}{2})+\frac{1}{2}KL(q\|\frac{p+q}{2}).$$

因此，在判别器D训练到当前最优的情况下，生成器G要优化的损失函数就变成了

$$C(G)=KL(p_{data}\|\frac{p_{data}+p_g}{2})+KL(p_g\|\frac{p_{data}+p_g}{2})$$
$$-2\log 2$$
$$=2JS(p_{data}\|p_g)-2\log 2.$$

所以，生成器G实际上在做的事情是最小化真实数据分布和生成分布的JS距离。JS距离与KL距离一样，都是≥0的，所以说G的最优解就是当$p_{data}=p_g$的时候，此时JS=0。所以G训练的目标就是要让生成器G生成的样本所形成的分布等于真实分布，是真实分布的一个估计。

GAN整个算法过程如下所示

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

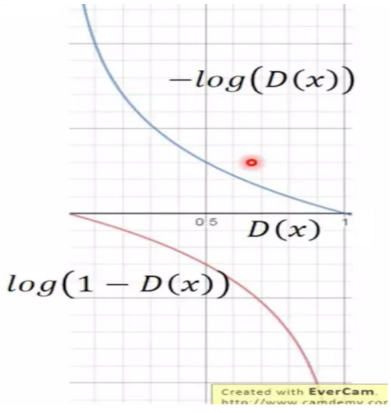
$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x}^{(i)}\right)+\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

e. 损失函数的另外两个版本
   i. Heuristic, non-saturating game
   由Minimax game得到的损失函数存在一个问题，设想在每一步训练中，我们对D训练k steps，把D训练得很好再去训练G，这时候会发生什么事情呢？把G的损失函数J(G) = $\min E_{z\sim p(z)}[\log(1-D(z))]$画出来，看看J(G)随着D(G(z))的变化如何变化



如上图所示，log(1-D(x))在D(x)接近0时比较平缓，梯度较小，在D(x)接近1的时候比较陡峭，梯度较大。在训练初期，G是比较差的，它生成的样本可以很容易地与真实样本区分开来，D也就很容易地被训练得很好，于是它从D得到的判别概率D(x)就会接近0，从而G损失函数提供的梯度很小甚至梯度消失，就会导致训练进

程非常缓慢甚至停滞。

为了解决这个问题，论文提出了改进版的G损失函数

$$J(G) = \min -E_{z\sim p(z)}[\log(D(G(z))]$$

看上图的蓝线，与原损失函数的单调性是一样的，所以最小化它与最小化原损失函数的结果是一样的，都是会让D(G(z))往1的方向发展，不同之处在于训练初期，也即D(x)接近0的时候，它能够提供较大的梯度，使得G更新较快，避免原损失函数有可能发生的梯度消失问题，且在D(x)接近1的区域梯度较小，即较真实的生成样本对梯度贡献小，较假的生成样本对梯度贡献大，接近收敛时放慢更新速度，是一个比较直观合理的训练过程。

使用这个新的损失函数的话，D和G之间的关系就不是minimax game了，他们的损失也不是零和的，需要分开写

$$J(D) = maxE_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z))]$$

$$J(G) = \min -E_{z\sim p(z)}[\log(D(G(z))]$$

我们将这个从梯度的角度得到的， 解决了梯度饱和问题的损失函数叫做Heuristic, non-saturating启发式损失函数。

但是！这个损失函数虽然解决了梯度问题，却带来了新的问题。

与前面一样，我们来看一下D和G的最优解是什么，各自在解决什么问题。

D的损失没变，和之前一样，所以还是$D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$

将D*代入G的损失函数，有以下推导

$$\begin{aligned}
C(G) &= -E_{x\sim p_g}[\log(D(x))] \\
&= -E_{x\sim P_g}\left[log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] \\
&= E_{x\sim p_g}[\log\frac{p_{data}(x)+p_g(x)}{p_{data}(x)}] \\
&= E_{x\sim P_g}\left[log\frac{p_g(x)*\left(\frac{p_{data}(x)}{p_g(x)}+1\right)}{p_{data}(x)}\right] \\
&= E_{x\sim p_g}\left[log\frac{p_g(x)}{p_{data}(x)} + \log\frac{p_{data}(x)+p_g(x)}{p_g(x)}\right] \\
&= E_{x\sim p_g}\left[log\frac{p_g(x)}{p_{data}(x)}\right] - E_{x\sim p_g}\left[1 - \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] \\
&= KL(p_g(x) \parallel p_{data}(x)) - E_{x\sim p_g}[1 - D^*(x)]
\end{aligned}$$

由之前minimax game G损失函数的最优解推导中，我们得到

$$\begin{aligned}
&E_{x\sim p_{data}}[\log D^*(x)] + E_{x\sim p_g}[1 - D^*(x)] \\
&= 2JS\left(p_{data}(x) \parallel p_g(x)\right) - 2log2 \\
&E_{x\sim p_g}[1 - D^*(x)] \\
&= 2JS\left(p_{data}(x) \parallel p_g(x)\right) - 2log2 - E_{x\sim p_{data}}[\log D^*(x)]
\end{aligned}$$

代入C(G)的式子就有

$$C(G) = KL(p_g(x) \parallel p_{data}(x)) - 2JS\left(p_{data}(x) \parallel p_g(x)\right) + 2log2 + E_{x\sim p_{data}}[\log D^*(x)]$$

最后一项与G无关，2log2是个常数，所以

$$C(G) = KL(p_g(x) \parallel p_{data}(x)) - 2JS\left(p_{data}(x) \parallel p_g(x)\right)$$

有趣的事情发生了，上式是G要最小化的式子，第一项是KL距离，第二项是负的JS距离，也就是说使用启发式损失函数意味着优化目标要求G在最小化两个分布之间的KL距离的同时，最大化他们之间JS距离，这不就是一个矛盾的目标吗？于是在实际应用中，这会导致损失函数的数值不稳定，训练不稳定。

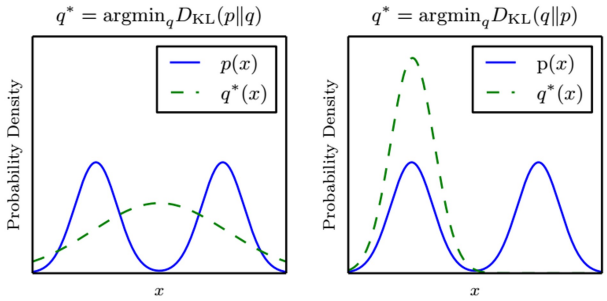
除去数值不稳定的问题，启发式损失函数还存在另一个问题：KL距离是不对称的，$\min_{p_g} KL(p_g(x) \parallel p_{data}(x))$ 和 $min_{p_g} KL(p_{data}(x) \parallel p_g(x))$的结果不一样。在这个损失函数中，计算的是前者，我们来看看它有什么问题。

当$p_{data}(x)\to 0$，而$p_g(x)\to 1$时，$KL(p_g(x) \parallel p_{data}(x))\to\infty$；

当$p_{data}(x)\to 1$，而$p_g(x)\to 0$时，$KL(p_g(x) \parallel p_{data}(x))\to 0$

这两种情况下JS距离的值都是log2，所以损失函数的值一个趋向正无穷，一个是比较小的值。这两种错误对G的惩罚不一样，第一种错误是生成器生成了一个不真实的样本，惩罚巨大，第二种错误是生成器生成不出来真实的样本，惩罚轻微，所以为了安全，不被惩罚，生成器就会倾向于生成单一但是安全的样本，即使数据分布中还有一部分的样本没被生成，它也不会去轻易尝试，因为生成不出来真实分布中的样本的惩罚很小，但是一旦尝试生成出来了一个糟糕的错误的样本，那就要接受巨大的惩罚。这个情况就导致了mode collapse的问题，mode collapse就是说，生成器学到的分布多样化比真实的数据多样化少很多，只能生成部分mode。

从两种KL距离的图像上来看更直观：



上图所示，两种KL距离的优化区别在当生成器所能表示的分布族复杂度小于真实分布时尤为明显。假设真实的分布p是两个高斯的混合，而我们生成器能表示的分布q是一个高斯。左边的情况，$min_q D_{KL}(p \parallel q) = E_{x\sim p}(logp - logq) \sim -plogq$，也即等价于$\max_q plogq$，p的值是给定的，q的值越接近1时logq值就越大，因此q(x)会希望在p(x)>0的地方都有比较大的值，即把概率放在p>0的地方，真实分布有两个高斯，q(x)希望两边的值都大，但是假设的分布只有一个高斯， 因此它选择了对两个高斯取平均，因此得到了左图。这种情况有点类似于使用MSE损失，也是导致一些算法生成的图比较模糊的原因。右边的情况，$min_q D_{KL}(q \parallel p) = E_{x\sim q}(logq - logp) \sim qlogq - qlogp$，第一项是q本身的熵，这个与p无关，第二项$min_q$-$qlogp \sim \max qlogp$, p的值是给定的，p接近1时logp大，趋向0，p接近0时logp小，且趋向无穷，所以q会希望在p比较小的地方尽量不放置概率，这样才能避免无穷小，而q只有一个高斯，所以它会选择p的其中一个高斯，而忽略另一个。p的两个高斯就是两个mode，右边情况完全忽略了其中一个mode，也即发生了mode collapse。

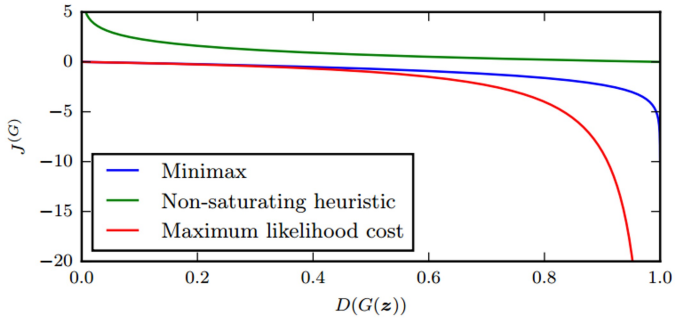
ii. Maximum likelihood game

在生成模型图谱中，本文讨论的所有生成模型都是源于最大似然的，因此GAN的损失也可以表示成最大似然的形式。如果将G的损失函数写成

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \exp\left(\sigma^{-1}\left(D(G(\boldsymbol{z}))\right)\right)$$

其中σ^{-1}是sigmoid函数的逆函数。同样地，与上述类似的推导过程，将$D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$代入上式并化简，可以证明最小化上式，就等价于$min_{p_g}$KL($p_{\text{data}}(x) \parallel p_{\text{g}}(x)$)，证明过程与前两种情况类似，略。

iii. 三个版本的损失函数对比



从上图可以看出: (1) minimax和maximum likelihood都存在训练初期梯度消失的问题 (2) maximum likelihood的损失函数变化较大，且靠近D(x)=1地方十分陡峭，即大部分的梯度都来自看起来比较真实的样本，这样的训练不太合理。(3) 启发式的损失函数解决了上述两个问题，但是如前所述，他同时也带来了mode collapse，数值不稳定等问题。虽然如此，在原始GAN的框架下，使用启发式损失函数在实际应用中的效果还是比较好的。

补充：

（1）在GAN tutorial2016中提到，有人认为mode collapse的产生原因不是因为KL($q \parallel p$)，GAN能够生成较清晰图片的原因也有待研究。

（2）在VAE或者其他生成模型中，也有最小化KL距离呀，所以他们会有mode collapse问题吗？不会，因为这些方法是生成器可以直接接触真实的数据样本，如VAE中有重构误差等，强行使得他们生成训练数据中所有出现过的mode，而GAN的生成器根本见不到真实样本，只能通过D的反馈调整自己的分布。