巧断梯度：单个loss实现GAN模型

22 Feb By 苏剑林 | 2019-02-22 | 23193位读者

我们知道普通的模型都是搭好架构，然后定义好loss，直接扔给优化器训练就行了。但是GAN不一样，一般来说它涉及有两个不同的loss，这两个loss需要交替优化。现在主流的方案是判别器和生成器都按照1:1的次数交替训练（各训练一次，必要时可以给两者设置不同的学习率，即TTUR），交替优化就意味我们需要传入两次数据（从内存传到显存）、执行两次前向传播和反向传播。

如果我们能把这两步合并起来，作为一步去优化，那么肯定能节省时间的，这也就是GAN的同步训练。

（注：本文不是介绍新的GAN，而是介绍GAN的新写法，这只是一道编程题，不是一道算法题～）

如果在TF中

如果是在tensorflow中，实现同步训练并不困难，因为我们定义好了判别器和生成器的训练算子了（假设为`D_solver`和`G_solver`），那么直接执行

```
sess.run([D_solver, G_solver], feed_dict={x_in: x_train, z_in: z_train})
```

就行了。这建立在我们能分别获取判别器和生成器的参数、能直接操作`sess.run`的基础上。

更通用的方法

但是如果是Keras呢？Keras中已经把流程封装好了，一般来说我们没法去操作得如此精细。所以，下面我们介绍一个通用的技巧，只需要定义单一一个loss，然后扔给优化器，就能够实现GAN的训练。同时，从这个技巧中，我们还可以学习到如何更加灵活地操作loss来控制梯度。

判别器的优化

我们以GAN的hinge loss为例子，它的形式是：

$$\begin{aligned}D =& \arg\min_D \mathbb{E}_{x\sim p(x)}\big[\max\big(0, 1 + D(x)\big)\big] + \mathbb{E}_{z\sim q(z)}\big[\max\big(0, 1 - D(G(z))\big)\big]\\ G =& \arg\min_G \mathbb{E}_{z\sim q(z)}\big[D(G(z))\big]\end{aligned} \tag{1}$$

注意$\arg\min_D$意味着要固定G，因为G本身也是有优化参数的，不固定的话就应该是$\arg\min_{D,G}$。

为了固定G，除了“把G的参数从优化器中去掉”这个方法之外，我们也可以利用`stop_gradient`去手动固定：

$$D, G = \arg\min_{D,G} \mathbb{E}_{x\sim p(x)}\big[\max\big(0, 1 + D(x)\big)\big] + \mathbb{E}_{z\sim q(z)}\big[\max\big(0, 1 - D(G_{ng}(z))\big)\big] \tag{2}$$

这里

$$G_{ng}(z) = \text{stop\_gradient}(G(z)) \tag{3}$$

这样一来，在式(2)中，我们虽然同时放开了D, G的权重，但是不断地优化式(2)，会变的只有D，而G是不会

变的，因为我们用的是基于梯度下降的优化器，而G的梯度已经被停止了，换句话说，我们可以理解为G的梯度被强行设置为0，所以它的更新量一直都是0。

生成器的优化

现在解决了D的优化，那么G呢？`stop_gradient`可以很方便地放我们固定里边部分的梯度（比如$D(G(z))$的$G(z)$），但G的优化是要我们去固定外边的D，没有函数实现它。但不要灰心，我们可以用一个数学技巧进行转化。

首先，我们要清楚，我们想要$D(G(z))$里边的G的梯度，不想要D的梯度，如果直接对$D(G(z))$求梯度，那么同时会得到D, G的梯度。如果直接求$D(G_{ng}(z))$的梯度呢？只能得到D的梯度，因为G已经被停止了。那么，重点来了，将这两个相减，不就得到单纯的G的梯度了吗！

$$D, G = \arg\min_{D,G} \mathbb{E}_{z\sim q(z)}\big[D(G(z)) - D(G_{ng}(z))\big] \tag{4}$$

现在优化式(4)，那么D是不会变的，改变的是G。

> 注：不需要从链式法则来理解这种写法，而是要通过stop_gradient本身的意义来理解。对于$L(D,G)$，不管G, D的关系是什么，完整的梯度都是$(\nabla_D L, \nabla_G L)$，而把G的梯度停止后，相当于G的梯度强行设置为0的，也就是$L(D,G_{ng})$的梯度实际上为$(\nabla_D L, 0)$，所以$L(D,G) - L(D,G_{ng})$的梯度是$(\nabla_D L, \nabla_G L) - (\nabla_D L, 0) = (0, \nabla_G L)$。

值得一提的是，直接输出这个式子，结果是恒等于0，因为两部分都是一样的，直接相减自然是0，但它的梯度不是0。也就是说，这是一个恒等于0的loss，但是梯度却不恒等于0。

合成单一loss

好了，现在式(2)和式(4)都同时放开了D, G，大家都是$\arg\min$，所以可以将两步合成一个loss：

$$\begin{aligned} D, G = \arg\min_{D,G} &\mathbb{E}_{x\sim p(x)}\big[\max\big(0, 1 + D(x)\big)\big] + \mathbb{E}_{z\sim q(z)}\big[\max\big(0, 1 - D(G_{ng}(z))\big)\big] \\ &+ \lambda\, \mathbb{E}_{z\sim q(z)}\big[D(G(z)) - D(G_{ng}(z))\big] \end{aligned} \tag{5}$$

写出这个loss，就可以同时完成判别器和生成器的优化了，而不需要交替训练，但是效果基本上等效于1:1的交替训练。引入λ的作用，相当于让判别器和生成器的学习率之比为$1:\lambda$。

参考代码： https://github.com/bojone/gan/blob/master/gan_one_step_with_hinge_loss.py

文章小结

文章主要介绍了实现GAN的一个小技巧，允许我们只写单个模型、用单个loss就实现GAN的训练。它本质上就是用`stop_gradient`来手动控制梯度的技巧，在其他任务上也可能用得到它。

所以，以后我写GAN都用这种写法了，省力省时～当然，理论上这种写法需要多耗些显存，这也算是牺牲空间换时间吧。