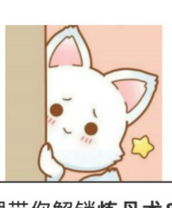
硬核推导Google AdaFactor：一个省显存的宝藏优化器

原创 苏剑林 夕小瑶的卖萌屋 2020-04-28 21:45:10 手机阅读



一只小狐狸带你解锁**炼丹术&NLP**秘籍

作者：苏剑林（来自追一科技，人称“苏神”）

前言

自从GPT、BERT等预训练模型流行起来后，其中一个明显的趋势是模型越做越大，因为更大的模型配合更充分的预训练通常能更有效地刷榜。不过，理想可以无限远，现实通常很局促，有时候模型太大了，大到哪怕你拥有了大显存的GPU甚至TPU，依然会感到很绝望。比如GPT2最大的版本有**15亿参数**，最大版本的T5模型参数量甚至去到了**110亿**，这等规模的模型，哪怕在TPU集群上也没法跑到多大的batch size。

这时候通常要往优化过程着手，比如使用**混合精度训练**（tensorflow下还可以使用一种叫做bfloat16的新型浮点格式），即省显存又加速训练；又或者使用更省显存的优化器，比如RMSProp就比Adam更省显存。本文则介绍**AdaFactor**，一个由Google提出来的新型优化器，首发论文为《Adafactor: Adaptive Learning Rates with Sublinear Memory Cost》。

AdaFactor具有自适应学习率的特性，但比RMSProp还要省显存，并且还针对性地解决了Adam的一些缺陷。

Adam

首先我们来回顾一下常用的Adam优化器的更新过程。设t为迭代步数，α_t为当前学习率，$L(\theta)$是损失函数，θ是待优化参数，ϵ则是防止溢出的小正数，那么Adam的更新过程为

$$\left\{\begin{aligned}
&g_t = \nabla_\theta L(\theta_t)\\
&m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t\\
&v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2\\
&\hat{m}_t = m_t\left/\left(1-\beta_1^t\right)\right.\\
&\hat{v}_t = v_t\left/\left(1-\beta_2^t\right)\right.\\
&\theta_t = \theta_{t-1} - \alpha_t \hat{m}_t\left/\sqrt{\hat{v}_t+\epsilon}\right.
\end{aligned}\right.$$

要省显存，就首先得知道显存花在哪里的。首先，计算量和显存的大头肯定都是$\nabla_\theta L(\theta_t)$，也就是说，计算梯度是很费资源的，这也是为啥“ALBERT相比BERT参数量虽然少了那么多，但训练速度也没见快多少”的原因了；除此之外，显存的消耗主要是m, v了，我们要维护两组缓存变量，来滑动计算梯度的前两阶矩（也就是m和v），用以计算参数的更新量。这两组变量每一组都跟训练参数本身一样大，因此对于参数比较多的模型，两组缓存变量所消耗的显存也不少。

AdaFactor

在这一节中，我们会相对详细地介绍一些AdaFactor优化器，介绍中会设计比较多的公式和推导。如果只求一个大致了解的读者，可以自行跳过部分数学内容～

抛弃动量

我们知道，CV模型很多时候要靠“SGD+动量”来炼出最优效果来，自适应学习率优化器通常训练不出最好的效果。但对于NLP模型来说，情况有点相反，自适应学习率显得更重要一些，很少听到由纯靠SGD调NLP模型的案例。因此，作为省显存的第一步，我们可以抛弃Adam里边的动量，这样就少一组缓存参数了，自然也就省了显存：

$$\left\{\begin{aligned}
&g_t = \nabla_\theta L(\theta_t)\\
&v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2\\
&\hat{v}_t = v_t\left/\left(1-\beta_2^t\right)\right.\\
&\theta_t = \theta_{t-1} - \alpha_t g_t\left/\sqrt{\hat{v}_t+\epsilon}\right.
\end{aligned}\right.$$

这其实就是RMSProp的变种，比RMSProp多了$\hat{v}_t = v_t\left/\left(1-\beta_2^t\right)\right.$这一步。

低秩分解

去掉m之后，缓存变量直接减少了一半，但AdaFactor还不满意，它希望保留自适应学习率功能，但把缓存变量v的参数量再压一压。这一次，它用到了矩阵的低秩分解。

广义KL散度

在SGD中，所有参数都是共用一个标量学习率；在Adam中，则是每一个参数都有自己的学习率$\alpha_t\left/\sqrt{\hat{v}_t+\epsilon}\right.$。我们知道通过精调学习率，SGD其实也能有不错的效果，这表明“每一个参数都有自己的学习率”这件事情都不是特别重要，或者换一种说法，就是“精调每一个参数自己的学习率”并不是特别重要。

这启发我们，将$\hat{v}_t$换一种参数更少的近似可能也就足够了。而“参数更少的近似”，我们就不难想到低秩分解了。对于$m\times n$的矩阵C，我们希望找到$m\times k$的矩阵A和$k\times n$的矩阵B，使得

$$AB \approx C$$

当k足够小时，A、B的参数总量就小于C的参数量。为了“省”到极致，AdaFactor直接让$k=1$，即寻找$\{a_i\}_{i=1}^m$和$\{b_j\}_{j=1}^n$，使得

$$a_i b_j \approx c_{i,j}$$

既然要近似，就要有一个度量的标准。很容易想到的标准是欧氏距离，即

$$\sum_{i,j}(a_i b_j - c_{i,j})^2$$

但在这个距离之下，a_i, b_j并没有解析解；此外，在优化过程中$c_{i,j}$（即$\hat{v}_t$）是非负的，而通过上述目标优化出来的$a_i b_j$无法保证非负，因此很可能扰乱优化过程。原论文的作者们很机智地换了一个度量标准，使得a_i, b_j有解析解。具体来说，它使用了“广义KL散度”，又称“I散度”，其形式为：

$$l = \sum_{i,j} c_{i,j}\log\frac{c_{i,j}}{a_i b_j} - c_{i,j} + a_i b_j$$

这个度量源自不等式$x\log x \geq x - 1 (\forall x > 0)$，当且仅当$x=1$时等号成立。所以代入$x = p/q\,(p,q>0)$，然后两端乘以$q$，我们有

$$p\log\frac{p}{q} - p + q \geq 0$$

当且仅当$p=q$成立，如果p,q有多个分量，那么对多个分量的结果求和即可，这就得到了度量。显然，广义KL散度是概率的KL散度的自然推广，但它不要求$c_{i,j}$和$a_i b_j$满足归一化，只要求它们非负，这正好对应了AdaFactor的场景。而且巧妙的是，这种情形配上这个目标，刚好有解析解：

$$a_i = \sum_j c_{i,j},\quad b_j = \frac{\sum\limits_i c_{i,j}}{\sum\limits_{i,j} c_{i,j}} \cdots (5)$$

其实这个解析解也很形象，就是行、列分别求和，然后相乘，再除以全体的和。

推导过程

直接对求偏导数并让偏导数等于0，得

$$\left\{\begin{aligned}
&\frac{\partial l}{\partial a_i} = \sum_j -\frac{c_{i,j}}{a_i} + b_j = 0\\
&\frac{\partial l}{\partial b_j} = \sum_i -\frac{c_{i,j}}{b_j} + a_i = 0
\end{aligned}\right.$$

整理得

$$\left\{\begin{aligned}
&a_i\sum_j b_j = \sum_j c_{i,j}\\
&b_j\sum_i a_i = \sum_i c_{i,j}
\end{aligned}\right.$$

注意到如果(a_i, b_j)是一组最优解，那么$(\lambda a_i, b_j/\lambda)$也是，说白了，所有的$a_i$乘以一个常数，所有的$b_j$也除以这个常数，$a_i b_j$是不变的。那么我们就可以随意指定$\sum\limits_i a_i$或$\sum\limits_j b_j$，因为它们就只是一个缩放标量而已。不失一般性，我们指定$\sum\limits_j b_j = 1$，那么就解得(5)。

直观理解

我们也可以从另一个角度理解结果。由于$c_{i,j}$是非负的，我们可以将它归一化，变成具有概率分布的特性，即$\hat{c}_{i,j} = \frac{c_{i,j}}{\sum\limits_{i,j} c_{i,j}}$，然后我们试图完成分解$\hat{c}_{i,j} \approx \hat{a}_i \hat{b}_j$，由于$\hat{c}_{i,j}$现在相当于一个二元联合概率分布，那么$\hat{a}_i, \hat{b}_j$就相当于它们的边缘分布，即

$$\hat{a}_i = \sum_j \hat{c}_{i,j} = \frac{\sum\limits_j c_{i,j}}{\sum\limits_{i,j} c_{i,j}},\quad \hat{b}_j = \sum_i \hat{c}_{i,j} = \frac{\sum\limits_i c_{i,j}}{\sum\limits_{i,j} c_{i,j}}$$

现在$\hat{c}_{i,j}$到$c_{i,j}$还需要乘上一个$\sum\limits_{i,j} c_{i,j}$，我们可以把它乘到$\hat{a}_i$或$\hat{b}_j$中，不失一般性，我们假设乘到$\hat{a}_i$上，那么就得到$(5)$。

AdaFactor雏形

有了结果(5)后，我们就可以用它来构建更省内存的优化器了，这就是AdaFactor的雏形。简单来说，当参数θ是普通一维向量时，优化过程保持不变；但θ是$m\times n$的矩阵时，算出来的梯度g_t也是矩阵，从而g_t^2也是矩阵，这时候我们对g_t^2做低秩分解，然后维护两组缓存变量$v_t^{(r)}\in\mathbb{R}^m, v_t^{(c)}\in\mathbb{R}^n$，分别滑动平均低秩分解后的结果，最后用$v_t^{(r)}, v_t^{(c)}$共同调整学习率：

$$\left\{\begin{aligned}
&g_{i,j;t} = \nabla_\theta L(\theta_{i,j;t})\\
&v_{i;t}^{(r)} = \beta_2 v_{t-1;i}^{(r)} + (1-\beta_2)\sum_j\left(g_{i,j;t}^2+\epsilon\right)\\
&v_{j;t}^{(c)} = \beta_2 v_{t-1;j}^{(c)} + (1-\beta_2)\sum_i\left(g_{i,j;t}^2+\epsilon\right)\\
&v_{i,j;t} = v_{i;t}^{(r)} v_{j;t}^{(c)}\left/\sum_j v_{j;t}^{(c)}\right.\\
&\hat{v}_t = v_t\left/\left(1-\beta_2^t\right)\right.\\
&\theta_t = \theta_{t-1} - \alpha_t g_t\left/\sqrt{\hat{v}_t}\right.
\end{aligned}\right.$$

（把ϵ加到g_t^2上去而不是$\hat{v}_t$上去，这是AdaFactor整出来的形式，不是笔者的锅～）.

滑动权重

在Adam以及上述AdaFactor雏形中，滑动权重β_2都是恒为常数，AdaFactor指出这是不科学的，并提出新的策略。

等价形式

为了认识到这一点，我们重写一下Adam的$\hat{v}_t$的更新过程：

$$\begin{aligned}
\hat{v}_t =& v_t\left/\left(1-\beta_2^t\right)\right.\\
=& \frac{\beta_2 v_{t-1} + (1-\beta_2) g_t^2}{1-\beta_2^t}\\
=& \frac{\beta_2 \hat{v}_{t-1}\left(1-\beta_2^{t-1}\right) + (1-\beta_2) g_t^2}{1-\beta_2^t}\\
=& \beta_2\frac{1-\beta_2^{t-1}}{1-\beta_2^t}\hat{v}_{t-1} + \left(1-\beta_2\frac{1-\beta_2^{t-1}}{1-\beta_2^t}\right)g_t^2
\end{aligned}$$

所以如果设$\hat{\beta}_{2,t} = \beta_2\frac{1-\beta_2^{t-1}}{1-\beta_2^t}$，那么更新公式就是

$$\hat{v}_t = \hat{\beta}_{2,t}\hat{v}_{t-1} + \left(1-\hat{\beta}_{2,t}\right)g_t^2$$

问题是这个$\hat{\beta}_{2,t}$够不够合理呢？答案是可能不大够。当$t=0$时$\hat{\beta}_{2,t}=0$，这时候$\hat{v}_t$就是g_t^2，也就是用实时梯度来校正学习率，这时候校正力度最大；当$t\to\infty$时，$\hat{\beta}_{2,t}\to\beta_2$，这时候$v_t$是累积梯度平方与当前梯度平方的加权平均，由于$\beta_2<1$，所以意味着当前梯度的权重$1-\beta_2$不为0，这可能导致训练不稳定，因为训练后期梯度变小，训练本身趋于稳定，校正学习率的意义就不大了，因此学习率的校正力度应该变小，并且$t\to\infty$，学习率最好恒定为常数（这时候相当于退化为SGD），这就要求$t\to\infty$时，$\hat{\beta}_{2,t}\to 1$。

新的衰减策略

为了达到这个目的，AdaFactor采用如下的衰减策略

$$\hat{\beta}_{2,t} = 1 - \frac{1}{t^c}$$

它满足$\hat{\beta}_{2,0}=0, \lim\limits_{t\to\infty}\hat{\beta}_{2,t}=1$。但即便如此，也不是任何$c$都适合，必须有$0<c<1$。$c>0$好理解，那为什么要$c<1$呢？原论文包含了对它的分析，大家可以去读读，但笔者觉得原论文的推导过于晦涩，所以这里给出自己的理解。

首先，对于$\hat{v}_t$来说，一个很容易想到的方案是所有梯度平方的平均，即：

$$\hat{v}_t = \frac{1}{t}\sum_{i=1}^t g_i^2 = \frac{t-1}{t}\hat{v}_{t-1} + \frac{1}{t}g_t^2$$

所以这等价于让$\hat{\beta}_{2,t} = 1-\frac{1}{t}$。这个方案美中不足的一点是，每一步梯度都是平权的，这不符合直觉，因为正常来说**越久远的梯度应该越不重要**才对，所以应该适当降低历史部分权重，而当$c<1$时，$1-\frac{1}{t^c} < 1-\frac{1}{t}$，因此一个简洁的方案是在式中取$c<1$，AdaFactor默认的$c$是$0.8$。

层自适应

最后，我们还可以进一步根据参数的模长来校正更新量，这个思路来自LAMB优化器，在之前的文章《6个派生优化器的简单介绍及其实现》中也介绍过。简单来说，它就是将最后的更新量标准化，然后乘以参数的模长，说白了，就是不管你怎么折腾，最后的更新量我只要你的方向，而大小由参数本身的模长和预先设置学习率共同决定，使得所有层所有参数的相对变化程度保持一致。

AdaFactor完整版

至此，我们终于可以写出完整版AdaFactor的更新过程了：

$$\left\{\begin{aligned}
&g_{i,j;t} = \nabla_\theta L(\theta_{i,j;t})\\
&\hat{\beta}_{2,t} = 1 - t^{-c}\\
&v_{i;t}^{(r)} = \hat{\beta}_{2,t} v_{t-1;i}^{(r)} + \left(1-\hat{\beta}_{2,t}\right)\sum_j\left(g_{i,j;t}^2+\epsilon_1\right)\\
&v_{j;t}^{(c)} = \hat{\beta}_{2,t} v_{t-1;j}^{(c)} + \left(1-\hat{\beta}_{2,t}\right)\sum_i\left(g_{i,j;t}^2+\epsilon_1\right)\\
&\hat{v}_{i,j;t} = v_{i;t}^{(r)} v_{j;t}^{(c)}\left/\sum_j v_{j;t}^{(c)}\right.\\
&u_t = g_t\left/\sqrt{\hat{v}_t}\right.\\
&\hat{u}_t = u_t/\max(1, RMS(u_t)/d)\times\max(\epsilon_2, RMS(\theta_{t-1}))\\
&\theta_t = \theta_{t-1} - \alpha_t\hat{u}_t
\end{aligned}\right.$$

其中$RMS(x) = \sqrt{\frac{1}{n}\sum\limits_{i=1}^n x_i^2}$是模长的变种，$\max(1, RMS(u_t)/d)$这一步相当于做了个截断，即$RMS(u_t)>d$时才执行归一化。原论文中的默认参数为

ϵ_1	10^{-30}
ϵ_2	10^{-3}
d	1
$\hat{\beta}_{2,t}$	$1-t^{-0.8}$

如果参数是一维向量而不是矩阵，那么$\hat{v}_t$使用普通的更新公式$\hat{v}_t = \hat{\beta}_{2,t}v_{t-1} + \left(1-\hat{\beta}_{2,t}\right)\left(g_t^2+\epsilon_1\right)$就行了。此外，论文还提出如果没有传入学习率，那么可以使用$a_t = \min\left(10^{-2}, \frac{1}{\sqrt{t}}\right)$为默认学习率，但笔者看源码的时候发现这个默认学习率很少使用，基本上还是需要自己传入学习率的。

开源实现

为了方便大家使用，笔者开源了自己实现的AdaFactor：

> https://github.com/bojone/adafactor

开源包括纯keras版和tf.keras版，使用方法跟普通keras优化器一样，tf.keras版也可以当做一个普通的tensorflow优化器使用。开源实现参考了mesh_tensorflow版的源码，在此表示感谢。优化器也已经内置在bert4keras中，方便大家调用。

需要提醒的是，用AdaFactor的时候，batch_size最好大一些，因为本身低秩分解会带来误差，而如果batch_size过小，那么梯度估算本身也带来较大的误差，两者叠加优化过程可能还不收敛。对于预训练模型来说，batch_size通常还是很大的，所以现在不少预训练模型开始用AdaFactor优化器了；对于普通的下游任务来说，AdaFactor也可以尝试，但可能需要多炼炼丹，才能搞出由于无脑Adam的效果。

文章小结

本文介绍了Google提出来的AdaFactor优化器，一个旨在减少显存占用的优化器，并且针对性地分析并解决了Adam的一些缺陷。笔者认为，AdaFactor针对Adam所做的分析相当经典，值得我们认真琢磨体味，对有兴趣研究优化问题的读者来说，更是一个不可多得的分析案例。

当然，没有什么绝对能有效的方法，有的只是方法虽好，要想实际有效，依然要用心炼丹。