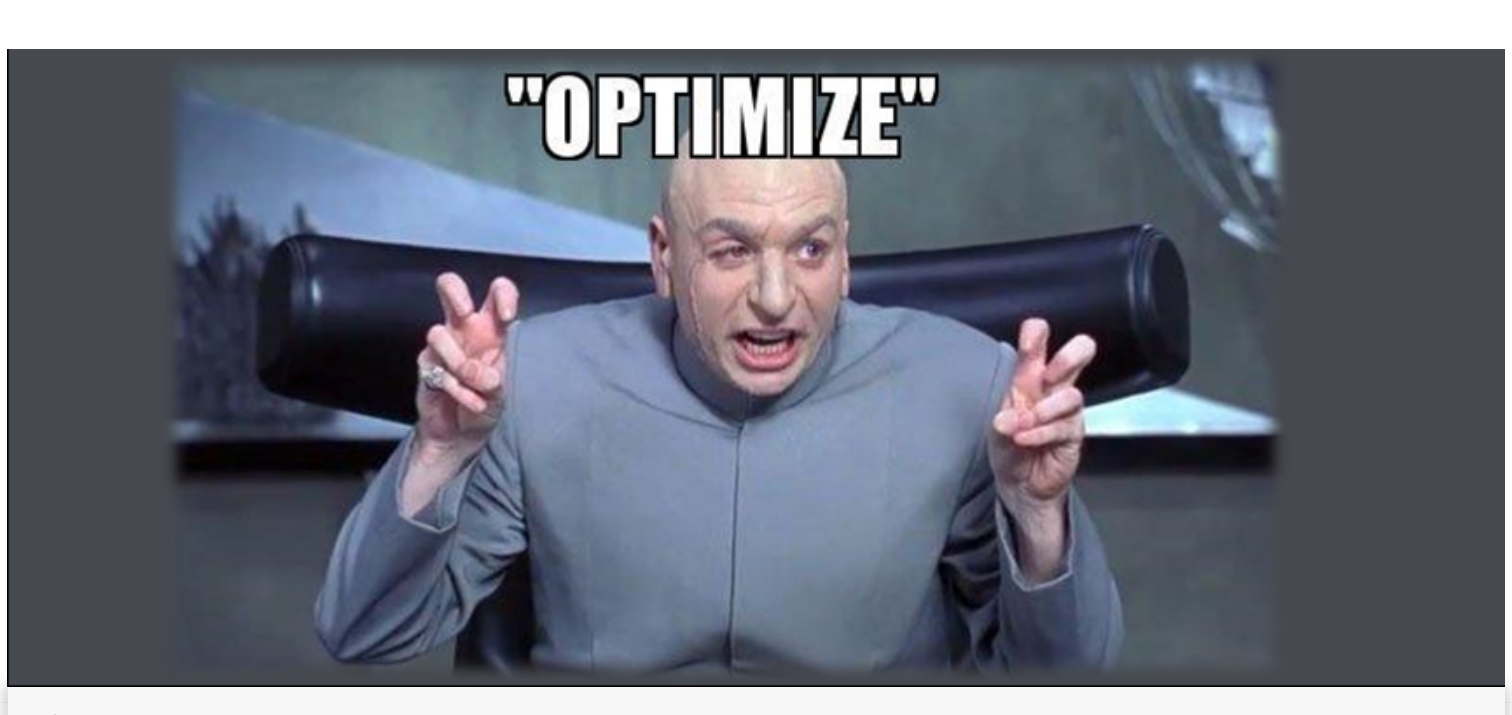
一训练就显存爆炸？Facebook 推出 8 比特优化器，两行代码拯救你的显存！

原创 jxyxiangyu 夕小瑶的卖萌屋 2021-11-11 12:05



文 | jxyxiangyu
编 | 小轶

> “小夕，小夕！又出来了个 SOTA 模型！赶紧 follow ！”
> 小夕看了看新模型的参数量， 然后看了看实验室服务器的几张小破卡。
> 小夕，陷入了沉默。

自从人们发现越大的模型性能越好后，神经网络模型的参数量就在越来越大的道路上一去不复返了。从XX-large到GPT3，再到5300亿参数的Megatron Turing-NLG，深度学习越来越像是只有财大气粗的大公司才能玩得起的玩具。如果，我们想要在实验室“简陋”的环境下，尝试更大的模型，有什么行之有效的方法呢？

最近，**Facebook 推出了支持 pytorch 的 8 位优化器，在减小内存占用的同时，竟然还能保持和32位优化器相当的准确性**。不得不说 facebook yyds。那么，下面就让我们一起来看看具体是怎么做的吧。

论文题目:
8-BIT OPTIMIZERS VIA BLOCK-WISE QUANTIZATION

论文链接:
https://arxiv-download.xixiaoyao.cn/pdf/2110.02861.pdf

开源链接:
https://github.com/facebookresearch/bitsandbytes

量化

在介绍论文作者的解决方法之前，先补充一点关于**量化**的基本概念。通常意义上来说，量化是指**将信号的连续取值近似为有限多个离散值的过程**。具体到计算机系统，指的是将**浮点数值**映射到**低bit数值**的操作[1]。

一般来说，我们可以通过以下手段应用量化

1. 量化模型**参数**w来压缩模型；
2. 量化模型某些层的**激活值**a来减少内存占用*；

(*注：参数和梯度也会占用一定的内存空间，但相对于激活值而言，占用比例不大，一般来说，量化参数和梯度带来的内存收益没有量化激活值的大*)

▲量化示意图

上图是 Song Han 在 ICLR'2016 上提出的量化方法。将模型参数分别聚类到几个质心，并将参数量化到对应的质心，在更新参数时，是将同一质心对应的梯度累加用于更新该质心对应的参数

可以看到，量化通过将参数（浮点值）映射到二值、三值或线性量化到一个区间（一般是低比特数值）的方式，减小了模型大小，在某些硬件上面，低比特数值运算速度高于浮点数值，一定程度上可以加速模型的训练和预测；除此之外，模型在训练和预测的时候，模型参数本身只占用了内存的一小部分，大部分存储了模型的激活值，如果将量化应用到激活值上，一定程度也减小了内存占用，这样，我们就可以尝试更大的模型和设置更大的mini-batch了。

当然，量化这么好，也不是没有缺点的，量化后的模型或多或少会引入精度损失；并且目前学术界多采用 pytorch 框架，好巧不巧的是 pytorch 框架对量化的支持没有 tensorflow 好，这总不能为了体验大模型的快感再转到 tensorflow 上面去吧，想想 tensorflow 混乱的 api 就头疼(╯ °Д°)╯ ┻━┻

最近，Facebook 推出了支持 pytorch 的 8 位优化器，在减小内存占用的同时，竟然还能保持和32位优化器相当的准确性。

状态优化器

再简单介绍下带有状态的优化器（stateful optimizer）。和普通的随机梯度下降（SGD）相比，为了加速优化而提出的带有梯度统计信息的优化器，就是**状态优化器**。常见的例如带动量的 SGD 和 Adam 。计算公式如下：

其中，β_1 和 β_2 是平滑因子，ϵ 是非常小的常量，α 是学习率。

作者认为，**状态优化器会维护历史梯度数据，一定程度上占用了内存**。通过**量化这些梯度，可以有效地降低内存占用**。

非线性量化

前述已经介绍了量化就是将信号的连续取值近似为有限多个离散值的过程，在降低模型参数量的同时，也会带来一定的精度损失，为减小精度损失，多采用非线性的量化方式，大致可以归纳为三个步骤：

1. 对于输入张量T，计算归一化因子N；
2. 将张量T通过T/N归一化后，找到在量化空间D中距离最近的值q_i；
3. 将量化后的张量T^Q的每个元素q_i的索引i存储下来

那么，我们就可以遍历存储的索引并通过下式得到反量化张量T^D：$T_i^D = Q^{map}(T_i^Q) \cdot N$，其中，$Q^{map}$是反量化映射

为了使不同元素值量级一致，一般会将张量归一化到$[-1, 1]$的区间范围，这时，N取的是输入张量中绝对值的最大值，即$N = max(|T|)$，然后通过二分查找的方式找到量化空间中距离该值最近的量化值$T_i^Q = \underset{j=0}{\overset{2^n}{\arg\min}} |Q_j^{map} - \frac{T_i}{N}|$

动态树量化

上一节看到，非线性量化在归一化时会严重依赖输入张量中的最值，像某些特别大或特别小的异常值，对量化会产生较大的精度影响。动态树量化（dynamic tree quantization）就是一种以较低的量化精度损失处理这种情况的方法。

与浮点数的存储方式类似，动态树量化以这类方式解释存储在内存中的数值，以此实现量化，具体由四部分组成：

1. 首位是符号位
2. 符号位后连续的0的数量表示指数大小
3. 再之后的第一位是指示位，如果指示位为1表示后续剩余的位为线性量化区域
4. 线性量化区域

其中，指示位是可以动态移动的。通过移动指示位，可以表示指数为10^{-7}或者精度为$1/63$的数值，表示范围为$[-1, 1]$

8位优化器

有了前面的知识铺垫，下面就可以详细地说明作者提出的8位优化器了。该8位优化器由三部分构成：

1. 逐块量化(block-wise quantization)
2. 动态量化(dynamic quantization)
3. 稳定的词嵌入层(stable embedding layer)

应用上述组件，将8位优化器的状态反量化为32位并更新状态和参数，然后将这些状态量化回8位进行存储。由于是在寄存器中进行8位和32位的转换，一定程度上可以减小内存占用并加速训练。

逐块量化

常见的量化需要将原始的张量在张量级别归一化，这样可能会引入核之间的多次信息通信和同步，造成额外的时间开销，而逐块量化则是将张量分成多个小块并在块级别归一化，减小了核之间的通信开销，除此之外，还可以将张量元素中的异常值的影响限制在单个块中。假设T为有n个元素的张量，分成每个大小为B的块，那么，可以分成n/B个块，对每个块分别做归一化，归一化因子为$N_b = max(|T_b|), b = 0, 1, \dots, n/B$，每个块分别通过下式进行量化操作：$T_{bi}^Q = \underset{j=0}{\overset{2^n}{\arg\min}} |Q_j^{map} - \frac{T_{bi}}{N_b}|$其中，$b$为块索引，$i$为块中元素的索引

动态量化

8位优化器的动态量化部分是对前面提到的动态树量化的扩展，对于像$Adam$的第二个状态r_t这种严格为正的数值，符号位就显得有些多余，而在语言模型的训练过程中，作者发现r_t的变化范围在3~5个数量级，小于动态树量化的7个数量级，因此，可以用固定的位将只会用到的位划分开，进一步减小内存占用。对于其他带符号的状态张量，则继续使用动态树量化。

稳定的词嵌入层

为了确保nlp任务中模型的稳定训练，作者还添加了稳定的词嵌入层。作者使用Xavier uniform对词嵌入层进行初始化，并且在与位置向量合并前进行层归一化操作，这样可以使参数在初始化和训练期间保持1左右的方差。词嵌入层的优化器状态用32位存储，权重和梯度用16位存储。

8位优化器 vs 32位优化器

作者在多个任务（包括机器翻译、大规模语言模型的预训练以及微调、图像分类和图像预训练以及微调）上比较了8位优化器和32位优化器的性能，比较的优化器包括$Adam$、$AdamW$或$Momentum$，实验中，除了将32位优化器替换为8位优化器外，**没有**改动超参和权重、梯度以及激活值的精度。

除了GLUE任务之外，其余的NLP任务均使用了作者提出的稳定词嵌入层。为确保实验结果的可信度，还在不同随机数种子下多次实验，选取了实验结果的中位数作为最终性能。实验结果如下所示：

可以看到，8位优化器在多个任务上均**达到**甚至是**超过**了32位优化器的性能，与此同时，还能**大幅减小内存开销**并**加速训练**。此外，作者还列出了在同等显存大小的条件下，使用8位优化器和32位优化器可以支持训练的模型。可以说，非常贴心了ヾ(●゜▽゜)ﾉ

消融研究

作者基于$RoBERTa$语料库训练了多个模型，用于研究8位优化器中各个组件的影响。实验结果如下：

其中，32位优化器(baseline)采用的是线性量化。

为测试优化器的稳定性，对于小规模的模型，作者分别训练了不同的超参数下的模型，超参数为ϵ {1e-8, 1e-7, 1e-6}，β_1 {0.90, 0.87, 0.93}，β_2 {0.999, 0.99, 0.98}以及学习率方面的改动，而对于超过1B的大规模模型，则是在相同超参下采用不同的随机数种子多次运行。所有的结果均是选择可以成功训练完（没有因梯度爆炸或弥散而无法训练）的模型性能的中值。

可以看出，逐块量化、动态量化和稳定的词嵌入层对结果都有**正向影响**。

此外，作者还对比了32位优化器和8位优化器对超参的敏感程度，比较了32位和8位优化器在β_1、β_2、ϵ和lr的变化下的走势

可以看到，8位优化器和32位相比，困惑度走势基本一致，表明对超参**不敏感**，在将32位优化器替换为8位优化器后，超参**不需要**进一步的调整

局限性

从实验结果可以看出，8位优化器完全可以作为32位优化器的替代品。当然，8位优化器也存在一些局限性：

1. 8位优化器需要稳定的词嵌入层来达到32位优化器的性能；
2. 8位优化器减小内存的大小与**模型参数量成正比**，对于像cnn这种激活值比参数占内存多得多的模型，8位优化器并没有特别明显的内存减小，反而更适合**transformer**这种架构的**大规模模型**

总结

不得不说，Facebook的8位优化器简直是我等“穷困”炼丹党的福音。现在，8位优化器已经开源，开源地址已经在文章开头提到。目前，8位优化器已经支持Adam, AdamW, RMSProp, LARS, LAMB优化器。使用时，需要安装并导入包bitsandbytes-cudaXXX，其中，XXX是本地环境的cuda工具包版本号，注释掉原有的优化器，调用8位优化器就可以了。

```
import bitsandbytes as bnb


# adam = torch.optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.995)) #
adam = bnb.optim.Adam8bit(model.parameters(), lr=0.001, betas=(0.9, 0.995)) #
adam = bnb.optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.995), optim_



torch.nn.Embedding(...) ->  bnb.nn.StableEmbedding(...) # recommended for NLP
```

据官网描述，仅仅需要改动两行代码，就可以节省75%的内存！小伙伴们，还不想抓紧时间上车体验一下嘛？

▲没时间解释了，快上车

[1] 商汤科技SenseTime，模型量化了解一下？(https://zhuanlan.zhihu.com/p/132561405)

[2] Song, H. , H. Mao , and W. J. Dally . "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." ICLR 2016. (https://arxiv-download.xixiaoyao.cn/pdf/1510.00149.pdf)