用 TensorFlow 压缩神经网络

2016-07-07 19:46:39

这是一篇翻译文章，原文来自 Pete Warden 的博客。

还可参考前文为什么八个二进制位对深度神经网络足够了，主要讲的是良好训练的神经网络必须能应对训练数据中的无关信息，这成就了神经网络对输入噪声和计算误差的强壮性。

图片来自 Jaebum Joo

很高兴我们发布了 TensorFlow 的八位量化支持的第一个版本。注意这里的“位”是指二进制位 (bit)。我努力促成在嵌入式可视化峰会之前完成，因为这对低功耗以及移动设备很重要。

神经网络发展过程中最大的挑战是：能让它工作起来就了不起了！这意味着训练阶段的精度和速度拥有最高优先级。采用浮点数是保证精度的最简单方法，而 GPU 正适合用来加速这些计算，因此自然而然没太多人关注别的数值格式。

如今，有许多模型正在被开发出来用于商业环境。训练的计算需求随着研究人员的增加而增加，但用于推断的计算量正比于用户数。那意味着纯粹的推断效率成了许多团队火烧眉毛的问题。

那就是用得着“量化”的地方。这是一个概括性术语，涵盖了许多不同的技术，用来存储数字并且针对这些数字进行计算，格式比 32 位浮点数更紧凑。这里着重谈八位定点数。

为何量化能工作

神经网络的训练是一个不断对权重添加细微修正的过程，这种细微修正一般需要浮点精度才能完成 (尽管也有工作试图从这个阶段开始就量化，比如二值化神经网络)。

用一个训练好的模型来做推断，则是另一回事。深度网络的一个魔力特性就是能够很好地应对较大的输入噪声。比如为了识别照片中的物体，网络必须忽略所有的 CCD 噪声、光照变化，以及其它与之前训练样本之间的非本质差异，而只关注重要的相似之处。这种能力意味着神经网络似乎把低精度计算视为另一种噪声来源，而在数值格式精度较低的情况下仍能给出准确结果。

为何要量化

神经网络可能会占据很大的存储空间，比如最初的浮点数格式的 AlexNet 大小就有 200 MB。这个大小几乎全部来自神经元连接的权重，因为单个模型里可能就有数百万或者更多个连接。由于它们都是略微不同

的浮点数，常见的像 zip 这样的压缩格式不会压缩很多。值得注意的是，神经元按层组织，在每层内的权重在一定范围内 (比如从 -3 到 6) 趋向正态分布。

一个最简单的压缩方案是，存储每层的最大和最小值，然后把这个区间线性分成 256 个离散值，于是此范围内的每个浮点数可以用八位 (二进制) 整数来表示，近似为离得最近的那个离散值。比如，最小值是 -3 而最大值是 6 的情形，0 字节表示 -3，255 表示 6，而 128 是 1.5。计算细节之后再说，因为涉及到一些微妙的东西，但大致说来就是可以把文件大小缩小 75%，使用的时候转换回普通的浮点数就可以仍旧使用原来的代码。

另一个做量化的原因是，通过完全使用八位格式的输入和输出来降低推断计算需要的计算资源。这个实现起来要难很多，因为需要修改所有涉及计算的地方，但可能带来额外的回报。八位数值的存取相对浮点数而言内存带宽降到 25%，这样可以更好地利用缓存并且避免 RAM 存取瓶颈。你还能使用“单指令多数据流” (SIMD) 操作实现在每个时钟周期进行更多操作。如果你使用能加速八位运算的数字信号处理 (DSP) 芯片，还能得到更多好处。

使用八位表示能让模型运行更快，能耗也更低 (这对移动设备尤其重要)，并且还对那些不能高效执行浮点运算的嵌入式系统开放了大门，这让面向物联网世界的大量应用成为可能。

为什么不直接用低精度表示来训练？

已经有一些使用低精度数值表示的实验了，但那些结果似乎表明，你需要高于八位的精度来处理后向传播和梯度。这让训练的实施更复杂，所以仅在推断时使用低精度比较合理。目前已经有了许多大家熟悉的基于浮点数的模型，所以如果能直接针对它们进行变换的话会很方便。

如何量化模型？

TensorFlow 自带对八位运算的生产级支持。它也能把浮点模型转换为等价的使用量化计算进行推断的图 (Graph；TensorFlow 里用来表达计算过程和内部状态的结构)。下面是一个把最近的 GoogLeNet 转换成八位表示的例子

```
curl http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz -o /tmp/ir
tar xzf /tmp/inceptionv3.tgz -C /tmp/
bazel build tensorflow/contrib/quantization/tools:quantize_graph
bazel-bin/tensorflow/contrib/quantization/tools/quantize_graph \
--input=/tmp/classify_image_graph_def.pb \
--output_node_names="softmax" --output=/tmp/quantized_graph.pb \
--mode=eightbit
```

这会生成一个新模型，执行的操作跟原来的模型一样，但内部采用八位计算。你会发现新的文件大小大致是原来的 1/4 (23 MB vs 91 MB)。你仍旧可以使用一模一样的输入，而结果应该是一致的。比如：

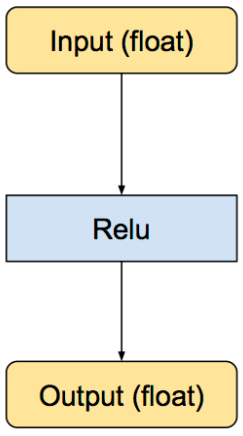
```
bazel build tensorflow/examples/label_image:label_image
bazel-bin/tensorflow/examples/label_image/label_image \
--input_graph=/tmp/quantized_graph.pb \
--input_width=299 \
--input_height=299 \
--mean_value=128 \
--std_value=128 \
--input_layer_name="Mul:0" \
--output_layer_name="softmax:0"
```

上面的命令运行了新的量化过的图，而输出的结果跟原始的很接近。

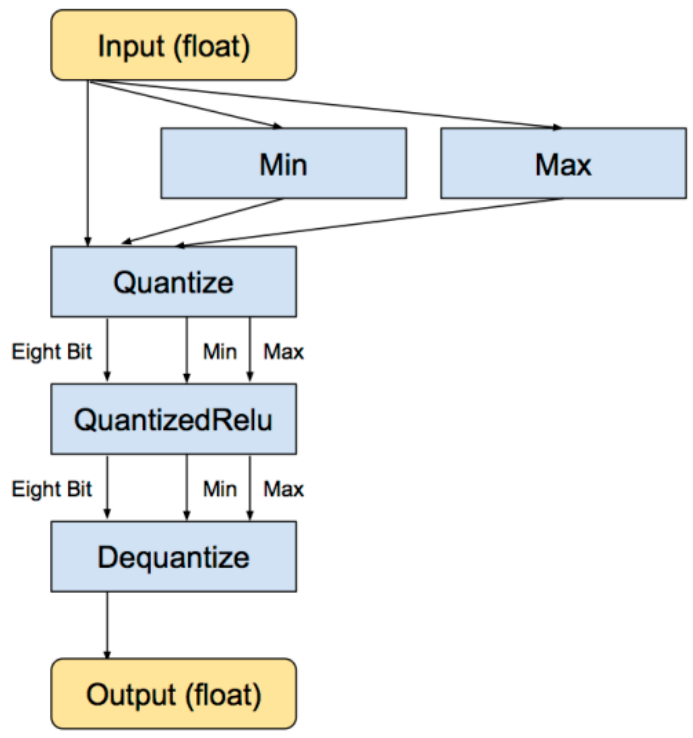
你可以对你自己的模型生成的 GraphDefs 进行同样的操作。我建议你先用 freeze_graph 脚本跑一遍，把一些中间检查点的变量转化为常数。

量化是如何进行的？

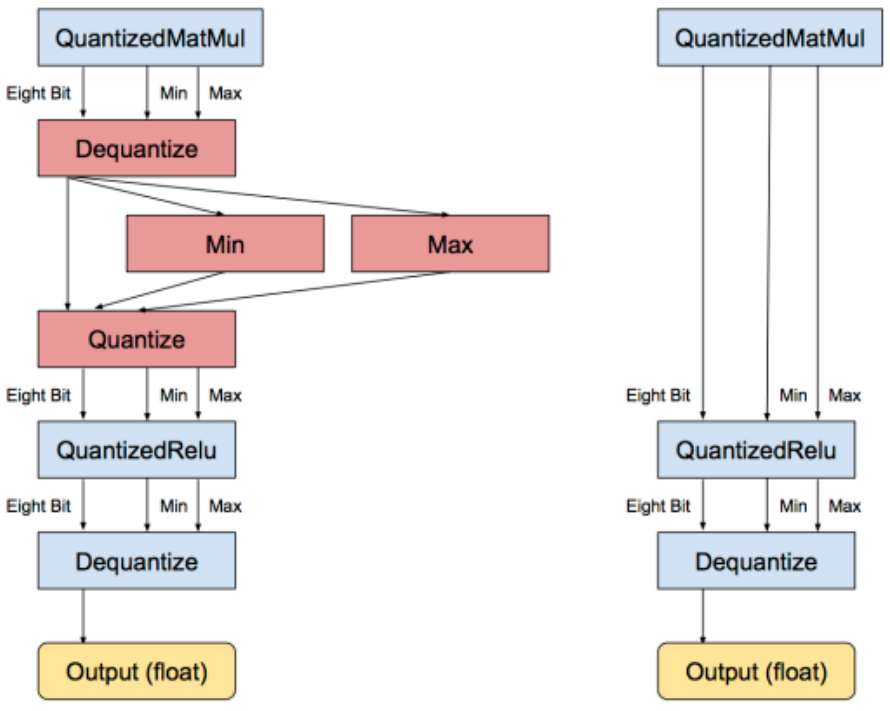
我们对量化的实现是通过把常见操作转换为等价的八位版本达到的。涉及的操作包括卷积，矩阵乘法，活化函数，池化操作，以及拼接。转换脚本先把每个已知的操作替换为等价的量化版本。这包括小的子图，它们之前之后都有转换函数，用来把数据在浮点数和八位数之间转换。下面是 ReLu 的例子，本来的模型输入和输出都是浮点数：



下面是等价的转换后的子图，输入和输出仍然是浮点数，但内部结构变化了，计算采用的是八位：



当每个操作被转换成八位之后，下一步就是移除不必要的转换。如果连续一串操作都有等价的浮点操作，则会有许多相邻的量化/去量化操作。当发现这种模式之后，转换程序会把相互抵消的那些操作移除，比如：



如果一个模型里所有的操作都能被量化，那转换后所有的操作都可用八位进行，而不需要转换成浮点数。

量化后的张量是如何表示的？

我们前面把从浮点数到八位数的转换描述为一个数据压缩问题。我们知道，训练好的神经网络模型里，权重和活化张量的数值倾向于在相对小的范围内取值 (比如，对于一个图像模型，权重可能在 -15 到 15 之间取值，而活化张量可能在 -500 到 1000 之间取值，当然精确值会有差异)。我们通过实验也知道神经网络对噪声不敏感，所以量化噪声对结果总体来说影响不会很大。我们希望采用容易执行计算的表示，特别是大矩阵乘法这种占据大部分时间的计算。

具体做法前面已经讲了，就是保存最小值和最大值的浮点数，然后通过把最大值和最小值之间分成 256 份来量化。也有别的压缩方法，比如 Song Han's code books，通过引入非线性性可以达到更小的位数，但计算上就昂贵一些。

采用清晰的量化格式的好处就是，对于那些没有量化版本的操作而言，很容易转换回去；对 debug 也方便。一个细节是，我们打算把最小值和最大值作为单独的张量传递，这样可以让图紧凑一些。

如何决定范围？

最小值和最大值经常可以被预先算出。权重参数在载入阶段就知道了，所以它们的范围可以作为常数保存。输入 (比如图片) 和活化函数也常常预先知道范围。这避免了对一个操作的输出进行分析来决定范围。对于卷积或矩阵乘法来说还是得分析，因为八位的输入会生成 32 位的结果。

如果你对八位输入进行任何算术计算，你会自然地累积得到超过八位精度能表达的数。比如，八位数相加的结果需要九位，八位数相乘的结果需要十六位。如果你把八位乘法的结果求和 (矩阵乘法需要做这个)，结果会超过十六位，通常需要至少 20 到 25 位，取决于点乘运算有多长。

这带来一个问题：我们需要把超过八位的输出缩减传递给下一个操作。对矩阵乘法而言，一个办法是根据可能的极端输入计算其输出的范围。这是安全的，因为我们可以通过分析证明其正确性。但实际上多数权重和活化值都分布得更均匀。这意味着实际的范围更窄。所以，我们使用了 QuantizeDownAndShrinkRange *[译者注：这个算符实质上就是在求一个张量各元素的最大值和最小值]* 操作符来分析实际的范围。也可以通过大量的训练数据来获得参数范围，然后把这些参数范围硬编码进去，但目前我们没有这么做。

舍入如何进行？

在量化的过程中，我们遇到的最难且最微妙的一个问题是偏差的累积。前面提过，神经网络对噪声不敏感，但如果对舍入操作不小心，会导致偏差往某个方向累积，最终影响精度。你可以去源代码中看看最终使用的公式，重要的一点是，我们需要让舍入后的输入值减去舍入后的最小值，而不是让输入值减去最小值然后才做舍入。

下一步是什么？

我们已经证明通过八位二进制数值表示而不是浮点数可以在移动设备和嵌入式设备上获得极好的性能。你可以在 gemmlowp 看到我们用于优化矩阵乘积的框架 *[译者注：看了一下，核心部分是用汇编语言实现的，比如这个文件；实现过程注意了缓存大小，参考这个文件；这个文件是针对尚无优化核心代码的 CPU 的参考实现，没有用汇编，算法就是简单的三重循环。]*。我们还需要把从 TensorFlow 操作符学到的经验应用到移动设备上以获得最佳性能。目前的量化实现可以作为一个足够快和足够精确的参考，可以促进对更多种设备的八位模型支持。

如果你感兴趣，推荐你仔细琢磨 TensorFlow 的量化代码，特别是实现操作符量化的核心部分，都包含了参考实现。

我们希望这个示范能激励社区一起探索低精度神经网络的潜力。感谢帮助实现量化支持的每一个人，真是了不起！

fujun.du@gmail.com fjdu fjdu Programming, mathematics, physics, etc.