6. 深度学习编译器 - 低精度计算之量化

除了上一节介绍的图优化外，采用低精度计算是另一种常见的优化网络运行性能的手段。通常，神经网络计算使用的是32-bit float 类型的数, 而所谓低精度计算，则是指用更少的比特表示神经网络中的数值，比如，float16 或者 int8 等。

为什么可以采用低精度计算呢？ 最重要的原因是神经网络对噪声具有较好的容错能力。比如对于图像分类任务，在输入图像叠加少量白噪声，一般并不会影响分类结果。低精度引入的误差也可以看做是一种噪声，在一定幅度下不会影响结果的准确性。

这样做有以下几方面的好处：

1. 减少访存开销和需要的内存/显存空间；比如将数值表示从 float 改为 int8 后, 访存数据量将会减少为原来的四分之一，需要的内存/显存也会减少为原来的四分之一；
2. 加快计算的速度；同一个芯片上，低精度计算的峰值算力通常数倍于 float, 比如，T4 float 类型的峰值算力为 8.1Tflops, 而 float16 的峰值算力为 65Tflops；
3. 扩展可以运行的设备；一些嵌入式微控制器只支持整数运算，通过量化将 float 转为 int16 或 int8 等低精度表示后，可以直接在这些设备上运行神经网络

本节主要关注将 float 量化为 int8/int16 的原理和实现方法。

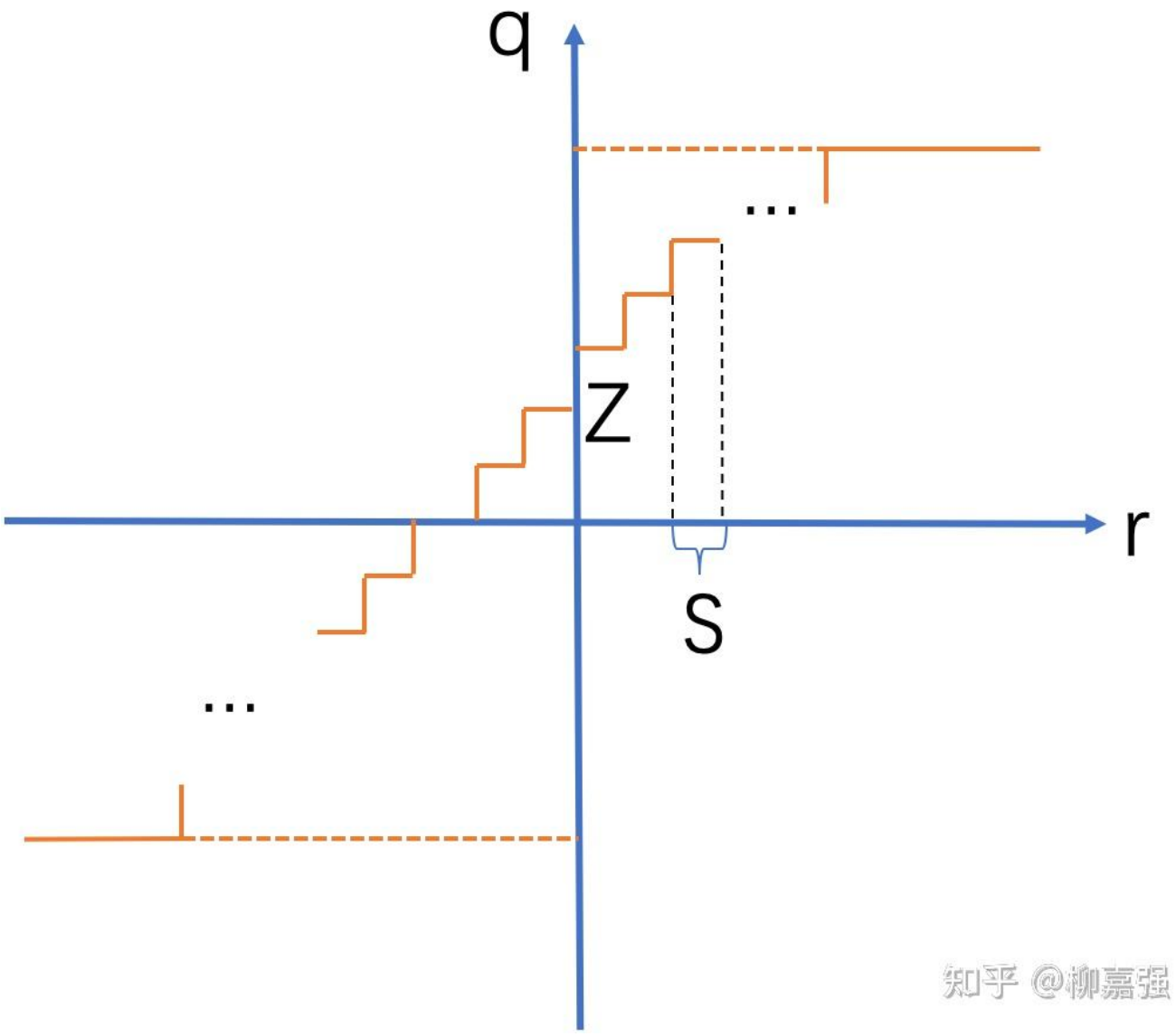
6.1 量化原理

下面用 rr 表示量化之前的浮点数，qq 表示量化后的整数。最常用的量化方法是线性量化，即 rr 和 qq 之间满足线性映射关系:

$$r = S(q - Z)$$

r = S(q-Z) \\

其中 SS 和 ZZ 是量化参数， SS 为浮点数， 表示量化的系数； ZZ 是整数，为量化后浮点 0 的表示方法。



是整数，为量化后浮点 0 的表示方法。

上图是线性量化原理示意图。可以直观看出，当 r = 0 时，量化后的结果为 Z 。 r 每增加 S ，对应的量化值增加 1。N 比特整数能表示的数字范围是 -2^{N-1} \sim 2^{N-1} – 1 。当 r 超过该阈值后，只能通过饱和用边界值的整数表示。 S 越大，饱和的阈值则越大；但同时量化后数值的分辨率越低。

根据上面的公式， \{S, Z, q\} 合起来可以看做浮点数 r 的另一种表示形式。为了达到减小内存开销，只使用整数运算等目的，若干个浮点数需要共享一组参数 \{S, Z\} 。最常用的共享方式包括每个张量共享一组参数，或者每个张量中的某一子块共享一组参数，例如，在二维矩阵中每行或每列共享一组参数；在图像特征图中，每个 Channel 共享一组参数。

后续讨论假设一个 Tensor 共享一组参数（Tensor 中某一子块共享 参数的情况可以参考这种情况进行分析）。对于 Tensor A 中的第 i 个数，用 r_A(i) 表示，对应的量化结果用 q_A(i) 表示，参数用 S_A 和 Z_A 表示。

6.2 量化后神经网络运算规则

下面讨论如何使用量化后的数完成神经网络中的常用运算。这里需要注意的是： 虽然量化后的数为整数，但它所代表的数值并不是通常意义上该整数的数值，而是和量化参数一起共同决定的一个数值。量化后神经网络运算的基本原则是确保其结果和量化前的一致性。根据量化所考虑的范围的不同，可以分为逐层量化和跨层量化两种，下面分别讨论。

6.2.1 逐层量化

逐层量化是指输入是浮点数，在计算之前将浮点数量化为整数，利用整数进行运算并将结果转回浮点数。该方式的主要目的是利用整数运算替换浮点数运算。逐层量化的优点是量化和反量化都是在一层内完成的，与整个网络的其他层无关，因此可以灵活的选择一些层进行量化。缺点是开销较大，每一层都要在输入时进行量化操作，且计算结果要转成浮点数。逐层量化经常应用在矩阵乘法运算中，接下来介绍这种方式矩阵乘法的计算流程。

C=Dot(A,B)

$C_{i,j}= \sum_k A_{ik} B_{kj} = \sum_k S_A (q_{i,k}^A - Z_A) S_B (q_{k,j}^B - Z_B)$

$C_{i,j} = S_A S_B (\sum_k q_{ik}^A q_{kj}^B - Z_A \sum_k q_{kj}^B - Z_B \sum_k q_{ik}^A + k Z_A Z_B)$

根据上面的推导，假设量化后 A，B 分别用 q^A 和 q^B 表示，则可以先用 q^A , q^B 进行矩阵乘, 减去矩阵 A、B分别按行、按列相加后的结果，最后加上一个固定的偏置 kZ_AZ_B 。然后将矩阵乘之后的结果乘以 S_AS_B 。

为了简化上述运算，在实际使用时通常会选择用对称量化，也即 Z_A=0 , Z_B = 0 。此时上述过程变为按照量化后的整数进行矩阵乘，然后将计算结果乘以固定系数 S_AS_B 即可。

6.2.2 跨层量化

逐层量化的输入和输出仍然是浮点数，因此并不能降低访存开销。另外，一些硬件只能支持定点运算，而逐层量化每层开始的量化过程以及最后的反量化过程仍然需要浮点计算，因此无法适用。跨层量化的目标就是解决上述问题，它的基本思路是每个 OP 的输入是整数，计算结果也是整数，可以直接参与下一个 OP 的计算。如果网络中的所有 OP 都能支持跨层量化，最终该网络的前向计算过程就可以完全通过整数运算完成，从而可以在只支持整数运算的硬件上运行，比如微控制器、 EdgeTPU 等。

我们还是以矩阵乘法为例讨论跨层量化的运算规则。根据前面的推导，我们有如下公式：

$C_{i,j} = S_A S_B (\sum_k q_{ik}^A q_{kj}^B - Z_A \sum_k q_{kj}^B - Z_B \sum_k q_{ik}^A + k Z_A Z_B)$

假设我们在之后的计算中对 C 进行量化：

$C_{ij} = S_c (q_{ij}^C - Z_c)$

则有：

$q_{ij}^C = Z_C + C_{ij}/S_C = Z_C + (S_A S_B)/S_C (\sum_k q_{ik}^A q_{kj}^B - Z_A \sum_k q_{kj}^B - Z_B \sum_k q_{ik}^A + k Z_A Z_B)$

根据上面的公式，可以直接在上一层的计算完成后，通过乘以 S_AS_B/S_C , 然后加上 Z_C 后作为结果传递到下一层。这种方式相当于预量化，即在上一层计算完成后就进行量化，然后将

量化后的结果传递到下一层。但是注意到 S_AS_B/S_C 是浮点数，如果直接相乘的话还是需要浮点运算。如果只支持整数运算，则需要将乘以 S_AS_B/S_C 的过程也通过整数运算完成。

解决方法是将 S_AS_B/S_C 转换为定点表示。参考文献 Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference 的做法，选择整数 n, 使得 (S_A S_B)/S_C = 2^{-n} M_0,0.5≤M_0<1 。然后将 M_0 表示为定点数。也即：

Q_{M_0}= round(2^{31}*M_0)

然后有：q_{ij}^C=Z_C+ C_{ij}/S_C

q_{ij}^C =Z_C+(S_A S_B)/S_C (∑_kq_{ik}^A q_{kj}^B - Z_A ∑_kq_{kj}^B - Z_B ∑_kq_{ik}^A + kZ_A Z_B)

q_{ij}^C ≈ Z_C+2^{-31-n} Q_{M_0} (∑_kq_{ik}^A q_{kj}^B - Z_A ∑_kq_{kj}^B - Z_B ∑_kq_{ik}^A + kZ_A Z_B)

上面的计算过程就全部为整数了。不过要注意因为后面有移位的操作，所以中间乘法的计算结果要用64 位来保存。具体实现可以参 考 https://github.com/google/gemmlowp/blob/2483d846ad865dd4190fe4a1a1ba2d9cfcea78e1/fixedpoint/fixedpoint.h 。

6.3 量化参数的设置

前面介绍了基本的量化原理，为了对一个 Tensor 进行量化，需要确定量化参数 S 和 Z 。量化参数的设置方法和实际应用场景相关，大体包括三类方法：

- 根据 Tensor 内数值分布提前或实时选择
- 根据典型样本数据下 Tensor 数值分布提前设置
- 通过训练获取。

6.3.1 根据 Tensor 内数值分布提前或实时选择

给定一个 Tensor, 可以找到 Tensor 中的最小值和最大值。假设最小值为 r_1 , 最大值 r_2 , 我们可以分别将其映射到量化后的最小值和最大值，假设为 q_{min} 和 q_{max} 。根据量化公式，可以得到：

r_1 = S(q_{min} - Z) , r_2 = S(q_{max} - Z)

因此

S = (q_{max} – q_{min}) / (r_2 – r_1), Z = (r_2 * q_{min} – r_1 * q_{max}) / (r_2 – r1) 。

由于 Z 需要为整数，因此在实际应用中，会通过微调 r_2 和 r_1 的值使 Z 为整数。

该方法主要适合两种场景，一种是针对权重（Weight）的量化。在预测之前，权重的数值已经完全确定且在预测的过程中不会改变，因此可以离线获取量化参数。另一种场景是在预测时针对中间层的输出，在参与下一层计算之前使用上述方法进行实时量化。

6.3.2 根据典型样本数据下 Tensor 数值分布提前设置

预测时中间层的输出是不确定的，上一种方法是根据实际的数据确定量化范围和量化参数，这种方法有两个缺点：一是每次重新确定量化参数会有额外开销；二是量化过程涉及浮点运算，一些硬件不支持。

解决方案是提前确定量化参数，并将重新量化的过程转换为只有整数运算。

怎么提前确定量化参数呢？通常的做法是选择一些典型的样本数据，收集用它们作为输入时神经网络每个 Tensor 的输出值，然后对于每个 Tensor，根据所有收集到的输出值的分布确定量化参数。

最后这步有不同的实现方法，比如最简单的，可以参考上一小节的公式，根据所有输出值的最小值和最大值确定。在实际应用中，TensorFlow 选择用指数滑动平均的方法确定最小值和最大值。具体来说，假设输入为第 k 个样本数据时 Tensor A 的取值范围为 [a_k, b_k] , 则用下面的公式更新 a 和 b (a, b 表示用于最终确定量化参数的最小值和最大值)：

a = \alpha * a + (1 - \alpha) * a_k , b = \alpha * b + (1 - \alpha) * b_k

通过调整 \alpha 的大小，可以控制 a 和 b 的更新速度。TensorFlow 中选择 \alpha 的值接近 1。

TensorRT 使用了对称量化策略，也即总是假设 Z = 0 。基于这种假设，TensorRT 有多种选择 S 的方法，其中一种是利用 K-L (Kullback-Leibler) 距离选择最优的量化系数 S 。K-L 距离通常用来衡量两个概率分布之间的差异，直观的讲，如果两个概率分布差异越大，则他们的 K-L距离也越大。假设 P、Q为某个离散随机变量的两种概率分布，P[i]、 Q[i] 表示两种分布下该变量取值为 i 的概率，则：

KL(P,Q)=∑_iP[i]*log(P[i]/Q[i])

TensorRT 利用 K-L 距离的上述特性，选择最优的 S 使得量化后数据的概率分布和量化之前数据的概率分布之间的 K-L 距离最小。

6.3.3 通过训练获取

上述方法是在训练完成后, 预测过程中确定量化参数的。好处是不需要修改模型训练过程，缺点是很难提前获得全局最优的量化参数。另一种方法是将量化参数看做神经网络参数的一部分，通过训练获取最优的量化参数 [7]。这种方法可以看做用梯度下降的更新方法替代了指数滑动平均和 K-L 距离的度量，能够端到端的考虑神经网络中各张量数值分布，在动态范围和精度之间做到更好的折中。具体方案可参考论文 "Trained Quantization Thresholds For Accurate and Efficient Fixed-Point Inference of Deep Neural Networks".