模型量化基础

1.背景

1.1概念

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，可理解成一种信息压缩的方法。常规精度一般使用 FP32（32位浮点，单精度）存储模型权重；低精度则表示 FP16（半精度浮点），INT8（8位的定点整数）等等数值格式。目前，低精度往往指代INT8。

1.1.1数值精度

float32 (FP32)

Float32 (FP32) 称为单精度浮点数。FP32 可以表示非常大和非常小的数值，并且具有很高的精度。在深度学习模型中，FP32通常用于训练阶段，因为它可以提供更高的数值稳定性和精度。然而，使用FP32也会增加模型的计算和存储需求，从而降低模型的性能和效率。在 FP32 中，为“指数”保留了 8 位，为“尾数”保留了 23 位，为符号保留了 1 位。因为是标准数据类型，所以大部分硬件都支持 FP32 运算指令。

float16(FP16)

Float16 (FP16) 称为半精度浮点数。FP16 相比 FP32 具有较小的数值范围和精度，但它可以减少模型的计算和存储需求，提高模型的性能和效率。在现代 GPU 和 TPU 等硬件设备上，FP16 通常具有更高的计算速度和能效比。因此，在深度学习模型推理阶段，通常会使用 FP16 以获得更快的速度和更高的能效。数据类型中，指数保留 5 位，尾数保留 10 位。这使得 FP16 数字的数值范围远低于 FP32。因此 FP16 存在上溢 (当用于表示非常大的数时) 和下溢 (当用于表示非常小的数时) 的风险。

INT8

INT8 是一种低精度数值类型，它使用 8 位来表示整数。INT8 可以进一步减少模型的计算和存储需求，提高模型的性能和效率。在深度学习模型推理阶段，INT8 通常用于量化技术，将模型的权重和激活从 FP32 或 FP16 转换为 INT8，从而减少模型的计算和存储需求，提高模型的性能和效率。然而，INT8 的数值范围和精度较低，可能会导致模型的数值稳定性降低，因此需要仔细选择量化策略和校准方法。

Bfloat16(BF16)

Bfloat16 (BF16) 为指数保留了 8 位 (与 FP32 相同)，为小数保留了 7 位。这意味着使用 BF16 我们可以保留与 FP32 相同的动态范围。但是相对于 FP16，我们损失了 3 位精度。因此，在使用 BF16 精度时，大数值绝对没有问题，但是精度会比 FP16 差。
FP32 称为全精度 (4 字节)，而 BF16 和 FP16 称为半精度 (2 字节)。

1.2目的

训练深度神经网络时，往往都会使用FP32的数据精度来表示权值、偏置、激活值等。在深度学习模型性能提高的同时，计算也越来越复杂，计算开销和内存需求逐渐增加。庞大的网络参数意味着更大的内存存储，而增长的浮点型计算次数意味着训练成本和计算时间的增长，这极大地限制了在资源受限设备，例如智能手机、智能手环等上的部署。如表2所示，深度模型在 Samsung Galaxy S6 的推理时间远超 Titan X 桌面级显卡，实时性较差，无法满足实际应用的需要。

模型量化的目的：

1. **更小的模型尺寸。**对原始模型进行量化可以显著减小尺寸，更易于部署在存储资源有限的设备上，如移动设备或嵌入式系统。
2. **更低的运算功耗。**计算量相似的情况下，在一些硬件上，整数运算消耗的能源较少。
3. **更快的计算速度。**计算机对于整数运算通常比浮点运算更快，特别是在没有专门的浮点硬件支持的设备上。
4. **持平的推理精度。**与量化前的推理精度基本持平。

1.3必要条件

量化想要带来实质性加速，需要一定的条件。

理论计算峰值：单位时钟周期内能完成的计算个数乘上芯片频率。

想要量化带来速度提升，需要满足两个条件：

1. 量化数值的计算在部署硬件上的峰值性能更高 。
2. 量化算法引入的额外计算少 。

1.4 量化对象

模型量化的对象主要包括以下几个方面：

- **权重（weight）**：weight的量化是最常规也是最常见的。量化weight可减少模型大小内存和占用空间。
- **激活（activation）**：实际中activation往往是占内存使用的大头，因此量化activation不仅可以大大减少内存占用，而且结合weight的量化可以充分利用整数计算获得性能提升。**激活值不像权值那样有固定数量，所以这时就得需要标定数据，即计算激活输出的动态范围，求得量化参数，然后再量化**。一般使用100个小批量数据就足够估算出激活输出的动态范围了。
- **KV cache**：量化 KV 缓存对于提高长序列生成的吞吐量至关重要。
- **梯度（Gradients）**：相对上面两者略微小众一些，因为**主要用于训练**。在训练深度学习模型时，梯度通常是浮点数，它主要作用是在分布式计算中减少通信开销，同时，也可以减少backward时的开销。

2.量化分类

2.1 线性量化与非线性量化

模型量化方法本质上是函数映射。量化建立了高精度的浮点数值和量化后低精度的定点数值之间的数据映射。**分为线性量化和非线性量化。**

- 线性量化：8bit量化（又分为对称量化、非对称量化）
- 非线性量化：二值量化 (1 bit量化)、聚类量化、对数量化

其中最常用的是8bit量化，已在工业界中成熟使用。

线性量化原理

假设r表示量化前的浮点数，量化后的整数q可以表示为：

和分别表示**取整**和**截断**操作，和是**量化后的最小值和最大值**。**s是数据量化的间隔，z是表示数据偏移的偏置，z为0的量化被称为对称（Symmetric）量化，不为0的量化称为非对称（Asymmetric）量化。**

浮点模型参数与定点模型参数之间如何转换呢?

- 浮点转定点（量化）公式为:

$$Q = round(\frac{R}{S} + Z)$$

定点转浮点（反量化）公式为:

$$R = (Q - Z) * S$$

【R】原始的浮点数据
【Q】量化后的定点数据
【Z】偏移量（或零点/最小值对应的量化数值），又被称为 Zero Point
【S】缩放系数，又被成为Scale

那S和Z如何获取呢?

- 可以知道浮点和定点参数的最值 R_{max}、R_{min}、Q_{max}、Q_{min}，则有:

$$S = \frac{R_{max} - R_{min}}{Q_{max} - Q_{min}}$$

$$Z = Q_{max} - \frac{R_{max}}{S}$$

对称量化可以避免量化算子在推理中计算z相关的部分，降低推理时的计算复杂度；非对称量化可以根据实际数据的分布确定最小值和最小值，可以更加充分的利用量化数据信息，使得量化导致的损失更低。

2.2 逐层量化、逐组量化和逐通道量化

根据**量化参数s 和z 的共享范围**（即**量化粒度**），量化方法可以分为**逐层量化（per-tensor）、逐通道（per-token & per-channel）量化**和**逐组量化（per-group）**。

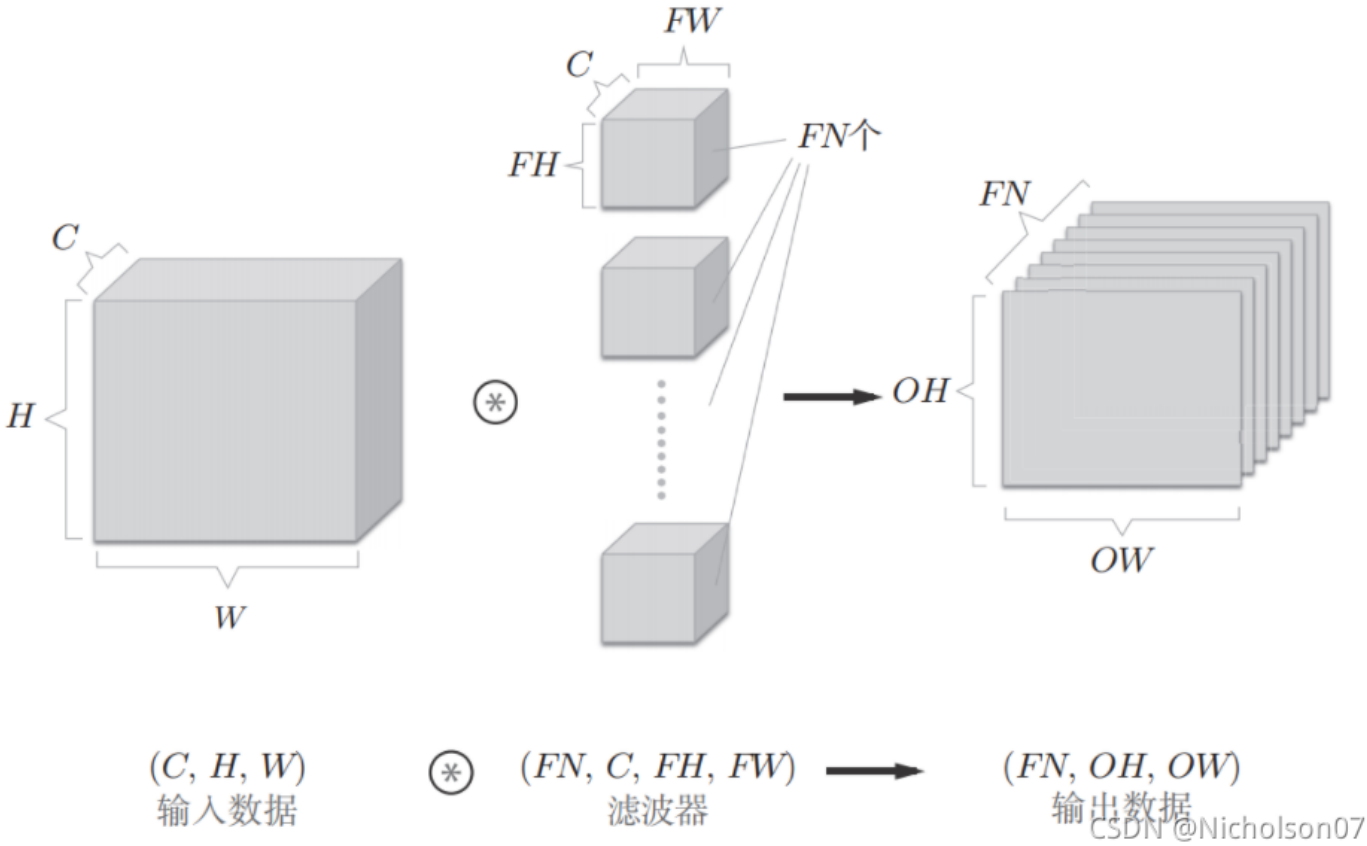
- **逐层量化**，这是最简单的一种方式，也是范围最大的粒度。以一层网络为量化单位，每层网络一组量化参数；
- **逐通道量化，以一层网络的每个量化通道为单位，每个通道单独使用一组量化参数**。逐通道量化由于量化粒度更细，能获得更高的量化精度，但计算也更复杂。
  - per-token：针对激活 x 而言：每行对应一个量化系数。
  - per-channel：针对权重 w 而言：每列对应一个量化系数。
- **逐组量化**以组为单位，**每个group使用一组s 和z ；它的粒度处于 per-tensor 和 per-channel 之间。**，当 group=1 时，逐组量化与逐层量化等价；当 group=num_filters（如：dw（Depthwise）将卷积核变成单通道）时，逐组量化与逐通道量化等价。

2.3 N比特量化

根据存储一个权重元素所需的位数，可以将其分为8bit量化、4bit量化、2bit量化和1bit量化等。

2.4 权重量化和权重激活量化

以一个简单深度学习网络为例：



其中滤波器就是权重，而输入和输出数据则是分别是上一层和当前层的激活值，假设输入数据为[3,224,224]，滤波器为[2,3,3,3]，使用如下公式可以计算得到输出数据为[2,222,222]。

因此，权重有 `2*3*3*3=54` 个（不含偏置），上一层的激活值有 `3*224*224=150528` 个，下一层的激活值有 `2*222*222=98568` 个，显然激活值的数量远大于权重。

根据需要量化的参数可以分类两类：权重量化和权重激活量化。

- 权重量化，即仅仅需要对网络中的权重执行量化操作。由于网络的权重一般都保存下来了，因而我们可以提前根据权重获得相应的量化参数S和Z，而不需要额外的校准数据集。一般来说，推理过程中，权重值的数量远小于激活值，仅仅对权重执行量化的方法能带来的压缩力度和加速效果都一般。
- 权重激活量化，即不仅对网络中的权重进行量化，还对激活值进行量化。由于激活层的范围通常不容易提前获得，因而需要在网络推理的过程中进行计算或者根据模型进行大致的预测。

根据激活值的量化方式，可以分为**在线量化**和**离线量化**。

- 在线量化，即指激活值的S和Z在实际推断过程中根据实际的激活值动态计算；
- 离线量化，即指提前确定好激活值的S和Z，需要小批量的一些校准数据集支持。由于不需要动态计算量化参数，通常离线量化的推断速度更快些。
  通常使用以下的三种方法来确定相关的量化参数。
- 指数平滑法，即将校准数据集送入模型，收集每个量化层的输出特征图，计算每个batch的S和Z值，并通过指数平滑法来更新S和Z值。

- 直方图截断法，即在计算量化参数Z和S的过程中，由于有的特征图会出现偏离较远的奇异值，导致max非常大，所以可以通过直方图截取的形式，比如抛弃最大的前1%数据，以前1%分界点的数值作为max计算量化参数。
- KL散度校准法，即通过计算KL散度（也称为相对熵，用以描述两个分布之间的差异）来评估量化前后的两个分布之间存在的差异，搜索并选取KL散度最小的量化参数Z和S作为最终的结果。TensorRT中就采用了这种方法。

2.5 按训练方式分类

- 训练后量化（Post-Training Quantization,PTQ），PTQ不需要再训练，因此是一种轻量级的量化方法。在大多数情况下，PTQ足以实现接近FP32性能的INT8量化。然而，它也有局限性，特别是针对激活更低位的量化，如4bit、2bit。这时就有了训练时量化的用武之地。
- 训练时量化也叫量化感知训练（Quantization-Aware-Training,QAT），它可以获得高精度的低位量化，但是缺点也很明显，就是需要修改训练代码，并且反向传播过程对梯度的量化误差很大，也容易导致不能收敛。

3.量化实现方式

3.1 pytorch

PyTorch对量化的支持目前有如下两种方式：

- PTQ（Post Training Quantization），模型训练完毕后的量化；
- QAT（Quantization Aware Training），模型训练中开启量化。

【训练后量化】

训练后量化过程：

1. 准备数据集。
2. 以训练好的高精度模型为基准，使用校正数据集对其量化。
3. 统计权重和激活值的数值范围，确定量化参数。
4. 使用量化参数对模型进行量化。

【量化感知训练】

量化过程不可避免的会带来模型精度的损失，为了能够尽量保持原模型的精度，通常会对量化后的模型做fine tuning，或者进行重新训练。这种方式称作“量化感知训练”。

如果模型量化的精度能够满足使用要求，则会忽略掉finetuning和重训过程，这种简单直接的量化方式称作“训练后量化”。

量化感知训练能够获得更高的量化后模型精度，但是量化过程较为繁琐；训练后量化过程简单快速，但是模型精度损失会较大。尤其是规模较小的模型，有时可能会导致无法使用。

量化感知训练过程：

- 训练一个高比特的浮点模型。
- 确定模型网络中需要量化的部分，并相应位置插入伪量化的算子。
- 在模拟量化过程（伪量化）的模式下迭代训练。
- 存储量化参数，降低精度，生成量化推理模型。
- 使用量化模型执行推理。

QAT的关键思想是在模型训练过程中引入量化的操作，让模型“意识”到量化过程，并通过反向传播优化模型参数，以适应量化带来的影响。具体来说，QAT遵循以下步骤：

- 模拟量化：在模型的前向传播过程中，将权重和激活值通过量化和反量化的过程，模拟量化在实际部署中的效果。这意味着，权重和激活值先被量化到低位宽的整数表示，然后再被反量化回浮点数，以供后续的计算使用。
- 梯度近似：由于量化操作（如取整）是不可微分的，为了在反向传播过程中计算梯度，QAT采用了梯度近似的技术。常见的方法包括直接通过量化操作传递梯度（即假设量化操作的梯度为1）或使用“直通估计”（Straight Through Estimator, STE）。
- 优化参数：通过模拟量化的前向传播和梯度近似的反向传播，模型参数在训练过程中得到优化，使模型适应量化后的表示。

QAT优点：

- 减少量化损失：由于QAT在训练过程中考虑了量化的影响，它可以显著减少量化对模型精度的负面影响，相比于PTQ，通常能够获得更好的性能。
- 提高模型兼容性：QAT使模型适应了量化后的权重和激活值的分布，从而提高了模型在特定硬件上的兼容性和运行效率。
- 灵活性和适应性：QAT允许开发者根据目标平台的特定需求，调整量化方案（如量化位宽、量化策略等），优化模型的性能。

注意，QAT和PTQ不是对立关系，QAT得到一个准备好量化的预训练模型，PTQ（或者其他高阶的量化技术）对这个预训练模型进行量化压缩，QAT的目的是通过在训练阶段就考虑量化的效果，为量化后的模型提供了一种“内生”的适应性，让PTQ压缩时模型精度最大程度保留。但QAT能会**增加模型训练的时间和计算资源消耗**。

3.2 TensorRT

TensorRT是一种可编程的推理加速器，可促进对NVIDIA图形处理单元（GPU）的高性能推理。 它可以与TensorFlow，Caffe，PyTorch，MXNet等训练框架以互补的方式工作。它专门致力于在GPU上快速有效地运行已经训练好的网络，以生成结果。TensorFlow已经集成了TensorRT，因此可以将其用于框架内加速推理。 TensorRT的量化工具支持PTQ和QAT量化

【PTQ量化】

流程：具体使用就是，我们导出ONNX模型，转换为TensorRT的过程中可以使用trt提供的Calibration方法去校准。可以直接使用trt官方提供的trtexec命令去实现，也可以使用trt提供的python或者C++的API接口去量化。

TensorRT实现int8量化

- 对权重直接使用了最大值量化
- 对偏移直接忽略
- 对激活值采用饱和量化

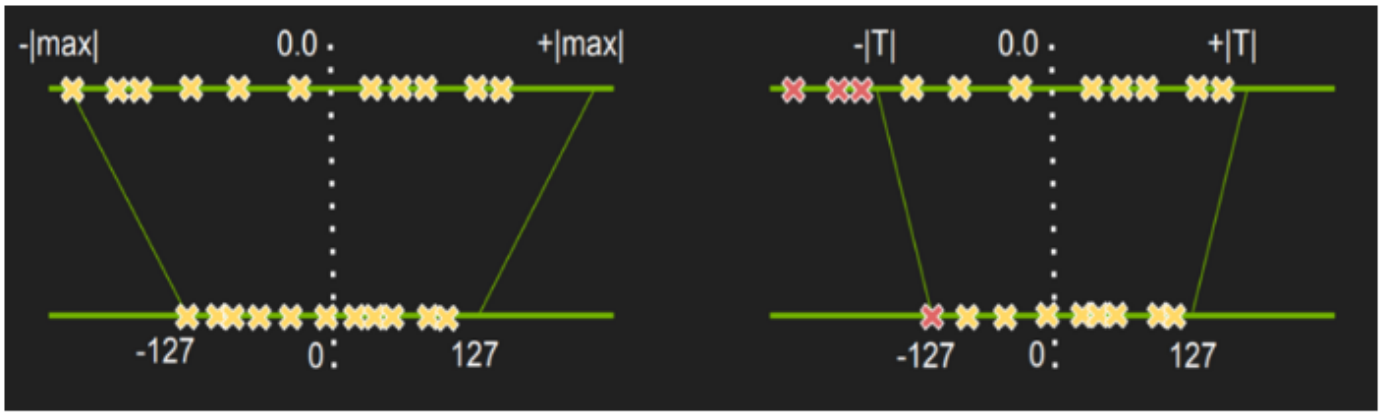


图1. 非饱和量化（左图）和饱和量化（右图）

非饱和量化：将浮点数正负绝对值的最大值对应映射到整数的最大最小值，但有效值的动态范围在int8上会很小。输入信号的幅值范围可以超过量化器的最大幅值范围。即输入信号可以被量化为超过量化器能够表示的最大值或最小值的幅值。可以提供更大的动态范围，但可能引入较大的量化误差。

饱和量化：输入信号的幅值范围被限制在量化器的最大幅值范围内。如果输入信号的复制超过量化器的额最大幅值范围，那么它将会被饱和到最大或最小幅值。饱和方式量化可以确保输出信号不会超过量化器范围，但可能导致动态范围过小。

为什么会有非饱和？因为激活值通常分布不均匀，

【×】直接使用非饱和量化会使得量化后的值都挤在一个很小的范围从而浪费了INT8范围内的其他空间，也就是说没有充分利用INT8（-128~+127）的值域；

【√】而进行饱和量化后，使得映射后的-128~+127范围内分布相对均匀，这相当于去掉了一些不重要的因素，保留了主要成分。

如何寻找这个阈值T就成了INT量化的关键，怎么找呢？

不同模型的激活值分布差异很大，这就需要进行动态的量化。

于是，NVIDIA就选择了KL散度也即相对熵来对量化前后的激活值分布进行评价，来找出使得量化后INT8分布相对于原来的FP32分布，信息损失最小的那个阈值。

INT8量化校准过程

- 先在一个校准数据集上跑一遍原FP32的模型；

- 然后，对每一层都收集激活值的直方图（默认2048个bin），并生成在不同阈值下的饱和量化分布；
- 最后，找出使得KL散度最小的那个阈值T，即为所求。

【TensorRT量化感知训练QAT】

量化感知训练(QAT)是TensorRT8新出的一个“新特性”，这个特性其实是指TensorRT有直接加载QAT模型的能力。QAT模型这里是指包含QDQ操作的量化模型。实际上QAT过程和TensorRT没有太大关系，trt只是一个推理框架，实际的训练中量化操作一般都是在训练框架中去做，比如我们熟悉的Pytorch。（当然也不排除之后一些优化框架也会有训练功能，因此同样可以在优化框架中做）

TensorRT-8可以显式地load包含有QAT量化信息的ONNX模型，实现一系列优化后，可以生成INT8的engine。