量化（3）：OmniQuant

之前系列的blog地址：

量化（1）：概念介绍

量化（2）：SmoothQuant

OmniQuant 介绍

本 blog 主要介绍 Omini 量化，论文地址：OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models

正如 量化（1） 中所描述，对一个 b/fp16 或者 float32 的数据集合来说， 量化就是把这个浮点数集合映射到一个 int4/8 的数据集合，决定量化后误差的一个关键是**阈值**的选择，最简单的办法就是 **MinMax**， 即把最大值和最小值当作阈值。但是当数据分布不均匀的时候，这种 **MinMax** 量化方式误差较大。因此需要把一些 outliers 剔除，这个操作可以叫做 **weight clipping**。那么阈值究竟选择多大，tensorrt 使用最小化 KV 散度来选择。

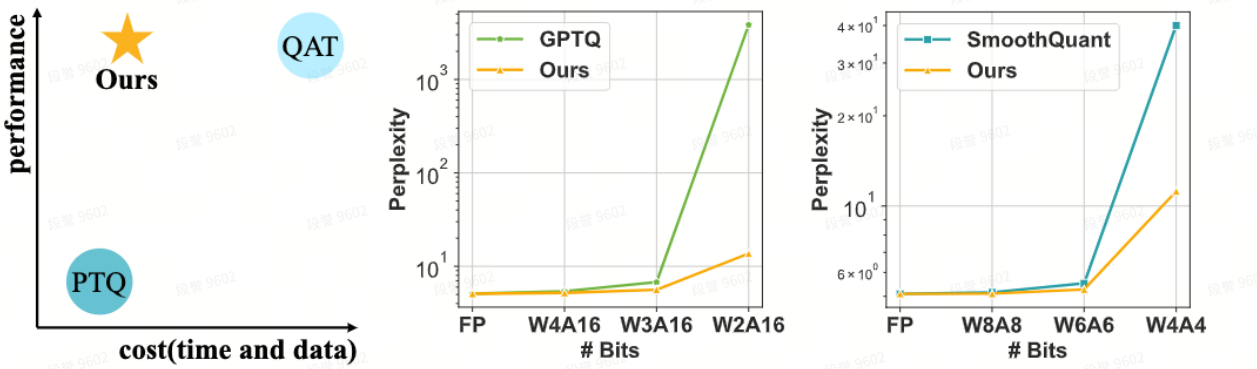
OmniQuant 使用梯度下降的方式最小化 **Block-wise error** 来选择阈值。 最小化 kl 散度的视角较小，仅考虑这两个集合（fp16和int）之间的接近程度。而 **Block-wise error minimization** 从更大的视角来考虑，即让一个 DecodeLayer 量化前后的误差最小。通过这种方式，即使是千亿模型，也只需一张 A100-80G 卡就可以完成量化（一个一个 Layer的进行）。

同时，在 weight-activation 量化中，activation 是难以量化的，因为在 channel 维度有较大的 outlier， 而weight 是 flat且容易量化的，SmoothQuant 提出（量化（2）：SmoothQuant） ，通过等价变换，将激活值的量化难度迁移到weight 上，从而使得 activation 和 weight 都是容易量化的。但是迁移的强度（migration strength） 是手动确定的，而OmniQuant 继续通过梯度下降的方式最小化 **Block-wise error** 来选择阈值更优迁移migration strength。在低bit（4，3）时，这种优势非常明显。

> we introduce a differentiable quantization technique for LLM called OmniQuant.
>
> where quantization parameters are learned with better flexibility. Towards this goal, OmniQuant is
>
> implemented with a block-wise quantization error minimization framework as presented in Sec.3.1.

> To tackle the aforementioned challenges of LLM quantization, we devise two novel strategies for
>
> additional learnable quantization parameters including a learnable weight clipping (LWC) to miti-
>
> gate the difficulty in quantizing weights and a learnable equivalent transformation (LET) to further
>
> shift the challenge of quantization from activations to weights

与 QAT 相比， OmniQuant 能够达到同等的精度，但是数据和时间成本要低的多得多。



(a) overview (b) weight-only quantization (c) weight-activation quantization

Figure 1: (a) provides a performance overview of the proposed OmniQuant, highlighting its ability to achieve quantization-aware training (QAT) performance with post-training quantization (PTQ) time and data efficiency. (b) and (c) showcase the perplexity (low is better) of quantized LLaMA-13B across different bit-widths on WikiText2.

原理

目标： 通过最小化 **Block-wise error** 得到**阈值 （weight clipping threshold）**以及 **smooth_scale**

> OmniQuant consists of two key components that incorporate different types of learnable quantization parameters, including Learnable Weight Clipping (LWC) and Learnable Equivalent Transformation (LET). Specifically, LWC modulates the extreme values of weights by optimizing the clipping threshold. In the mean-
>
> while, LET tackles activation outliers by learning mathematically equivalent transformations in a transformer encoder.

以 weight clipping threshold 为例，

$$\mathbf{W_q} = \text{clamp}(\lfloor \frac{\mathbf{W}}{h} \rceil + z, 0, 2^N - 1), \text{where } h = \frac{\gamma \max(\mathbf{W}) - \beta \min(\mathbf{W})}{2^N - 1}, z = -\lfloor \frac{\beta \min(\mathbf{W})}{h} \rceil$$

公式中，量化因子 h 的分子中 gama 和 beta 是一个 [0, 1] 之间的数，这个参数就是通过学习得到的。当 gama 和 beta 都为 1 是，就成了 MinMax quantization scheme。Clipping 之后，原本的fp16 weights就会变得很容易量化。

> 通过学习得到量化因子h后，然后对origin weight 进行量化和反量化（Clipping），得到的new weights 是 easy-quant 的。

skywork-Mixtral-8x7B-MoE Omni量化

ppl 困惑度（越小越好）	数据集： wikitext2	备注
fp16	4.894	
W4A16（OmniQuant）	5.224	
W4A16g128（OmniQuant）	5.193	channel 维度设置 group=128
W4A16（smooth + OmniQuant）	5.191	先对模型做 smooth 处理，然后进行 Omni 量化
W4A16g128（smooth + OmniQuant）	5.194	在3的基础上，量化时 channel 维度设置 group=128，发现效果并不理想
W4A16g128（AWQ + OmniQuant）	5.21	
W4A16g128（AWQ + OmniQuant）# 浩然	5.05	

评测分数	c_eval	mml	GSM8K	skyeval	alignbench
bf16	62.2	68.9	73.5	68.11	7.33
w4a16	60.847	67.04	70.88	61.71	6.71

Mixtral-8x7B-MoE 性能测试

weight_only 量化，**W4A16**

Cache kv 空间对比：

	GPU blocks	一个 block 可以存 16 个 token，每个请求按平均 3000 个 token计算，理论最大 batch 为：

bf16 双卡	24165	24165 * 16 / 3000 = 128
W4A16 单卡	23436	23436 * 16 / 3000 = 124

显存空间充足，实际线上为了保证低延迟，服务实际负载 batch 一般不高于 16

首 token 和 生成速度对比, Batch = 1

seq_len	100	1000	2000	4000
bf16 (双卡)	40.48ms / 13.60 ms	107ms / 13.61 ms	237 ms / 13.49 ms	417 ms / 13.29 ms
W4A16 （单卡）	34 ms / 11.5 ms	178.5 ms / 12.0 ms	326.9 ms / 12.01 ms	665.7 ms / 12.36 ms

bf16 数据来源宝国之前的性能测试： Mixtral 性能测试（双卡）

附件

关于梯度优化器：

1. 交叉熵损失函数：CrossEntropyLoss

关于信息量，熵，信息熵，交叉熵，KL散度这些东西，有兴趣继续深入理解，这里有个基础的视频可以看下：

如何理解信息熵

交叉熵的公式：

$$H(p,q) = -\sum_x p(x)\log q(x) = \sum_x \log \frac{p(x)}{q(x)}$$

在给定 p 的情况下，如果 q 和 p 越接近，交叉熵越小；如果 q 和 p 越远，交叉熵越大。在实际应用中，例如在多分类任务模型的学习中，p为标签，而q为模型经过softmax层的输出，代表每一类分类的概率。

因此交叉熵可以**简单理解为在衡量深度学习分类问题中，真实分类和实际分类之间的差异程度， p(x)为真实 label， q(x)为模型的预测值。**

例如一个手写数字识别，一共10类：

- 假设softmax的输出层为【0.01,0.22,0.64,0.03,0.01,0.02,0.03,0.04】

若分类正确

- 标签为【2】

则交叉熵损失为 `- log 0.64`

若分类错误时：

- 标签为【1】

则交叉熵损失为 `- log 0.01`

```
import torch
import torch.nn as nn

logits = torch.tensor([[0.2, 0.8]])
labels = torch.tensor([1])

criterion = nn.CrossEntropyLoss()
loss = criterion(logits, labels)
print("交叉熵损失:", loss.item())

import math
softmax_func=nn.Softmax(dim=1)
logits = softmax_func(logits)
print(logits)
loss = -(math.log(0.6457))
print(loss)
```

> Torch 的 CrossEntropyLoss 会先对 logits 做 softmax 处理。