如何在英特尔® 平台上实现高效的大语言模型训练后量化

提升 SmoothQuant 量化方法的效力

作者：英特尔公司 陆峯彤、何欣、郭恒、程文华、王畅、王梦妮、沈海豪

本文介绍了可提升大语言模型的训练后量化表现的增强型 SmoothQuant 技术，说明了这项技术的用法，并证明了其在准确率方面的优势。此方法已整合至英特尔® Neural Compressor（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/neural-compressor.html）中。英特尔® Neural Compressor 是一个包含量化、剪枝（稀疏性）、蒸馏（知识提炼）和神经架构搜索等多种常用模型压缩技术的开源 Python 库。目前，诸如 TensorFlow、英特尔® Extension for TensorFlow（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/optimization-for-tensorflow.html）、PyTorch、英特尔® Extension for PyTorch（https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/optimization-for-pytorch.html）、ONNX Runtime 和 MXNet 等主流框架，都能与之兼容。

英特尔® Neural Compressor 已经支持多款英特尔® 架构的硬件，比如英特尔® 至强® 可扩展处理器（https://www.intel.cn/content/www/cn/zh/products/details/processors/xeon/scalable.html）、英特尔® 至强® CPU Max 系列（https://www.intel.cn/content/www/cn/zh/products/details/processors/xeon/max-series.html）、英特尔® 数据中心 GPU Flex 系列（https://www.intel.cn/content/www/cn/zh/products/details/discrete-gpus/data-center-gpu/flex-series.html）和英特尔® 数据中心 GPU Max 系列（https://www.intel.com/content/www/us/en/products/details/discrete-gpus/data-center-gpu/max-series.html）。本文涉及的实验基于第四代英特尔® 至强® 可扩展处理器（https://www.intel.cn/content/www/cn/zh/events/accelerate-with-xeon.html）进行。

大语言模型

大语言模型 (Large Language Model, LLM) 需基于海量数据集进行训练，可能拥有数十亿权重参数。其先进的网络结构和庞大的参数量，使它们能够很好地应对自然语言本身的复杂性。完成训练后的大语言模型，可针对各种下游的自然语言处理 (NLP) 和自然语言生成 (NLG) 任务进行调优，让其更适合对话式聊天机器人（如 ChatGPT）、机器翻译、文本分类、欺诈检测和情感分析等任务场景。

大语言模型部署面临的挑战

大语言模型在执行自然语言处理和自然语言生成任务方面表现出色，但其训练和部署颇为复杂，主要面临以下挑战：

- AI 与内存墙（https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8）瓶颈问题：算力每两年提高 3.1 倍，内存带宽却只提高 1.4 倍；
- 网络带宽挑战：训练大语言模型需要采用分布式系统，这对网络带宽提出了较高要求；
- 系统资源有限：训练后的模型往往会部署在算力和内存资源均有限的系统上。

因此，采用训练后量化的方法来为大语言模型瘦身，对于实现低时延推理至关重要。

大语言模型的量化

量化是一种常见的压缩操作，可以减少模型占用的内存空间，提高推理性能。采用量化方法可以降低大语言模型部署的难度。具体来说，量化是将浮点矩阵转换为整数矩阵：

$$X_{int8} = round(X_{fp32}/S) + Z$$

其中 X_fp32、S 和 Z 分别为输入矩阵、比例因子和整数零点。

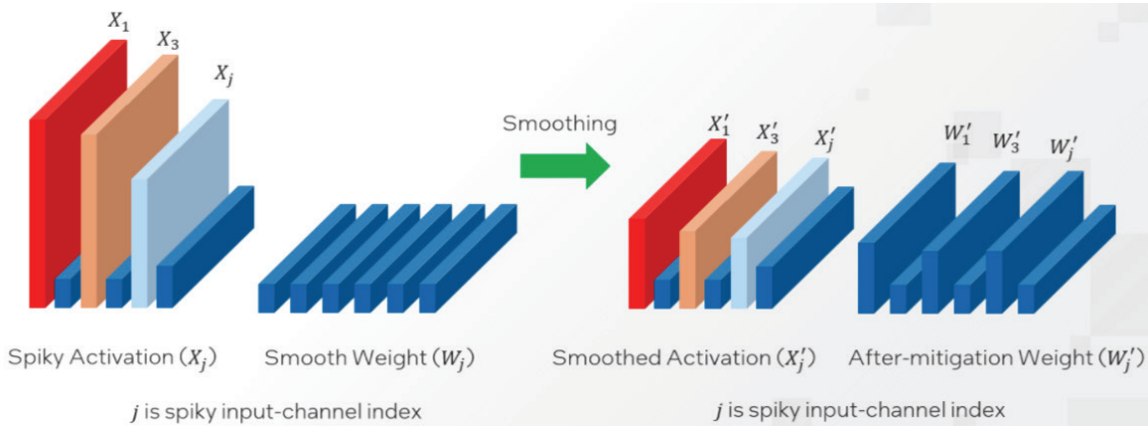
有关每通道 (per-channel) 量化策略虽然可能会减少量化损失，但不能用于激活值量化的原因，请参看 SmoothQuant 相关文档（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。不过，激活值量化误差损失却是导致模型量化准确率下降的重要因素。为此，人们提出了很多方法来降低激活值量化损失，例如：SPIQ（https://arxiv.org/abs/2203.14642）、Outlier Suppression（https://arxiv.org/abs/2209.13325）和 SmoothQuant（https://arxiv.org/abs/2211.10438）。这三种方法思路相似，即把激活值量化的难度转移到权重量化上，只是三者在转移难度的多少上有所不同。

增强型 SmoothQuant

SmoothQuant 引入了一个超参数 α 作为平滑因子来计算每个通道的量化比例因子，并平衡激活值和权重的量化难度。

$$s_j = max(|X_j|)^{\alpha} / max(|W_j|)^{1-\alpha}$$

其中 j 是输入通道索引。



对于 OPT 和 BLOOM 等大多数模型来说，α=0.5 是一个能够较好实现权重和激活值量化难度分割的平衡值。模型的激活异常值越大，就越需要使用更大的 α 值来将更多的量化难度转移到权重上。

原始的 SmoothQuant 旨在通过针对整个模型使用一个固定值 α 来分割权重和激活值的量化难度。然而，由于激活异常值的分布不仅在不同模型之间存在差异，而且在同一模型的不同层之间也不尽相同，因此，本文推荐使用英特尔® Neural Compressor 的自动调优能力，逐层获取最佳 α 值。

相关方法包括以下五个主要步骤（伪代码如下所示）：

1. 通过特殊的回调函数 register_forward_hook 捕获 (hook) 模型各层的输入和输出值。
2. 根据用户定义的 α 范围和步长生成一个 α 值列表。
3. 根据给定的 α 值重新计算平滑因子并调整参数（权重值和激活值）。

4. 对权重执行每通道量化与反量化（quantization_dequantization），对输入值执行每张量（per-tensor）量化与反量化，以预测与给定 α 值对应的每层输出值。
5. 计算相对实际输出值的均方损失，将调整后的参数恢复回来，并保存每层的最佳 α 值。

Algorithm: α auto-tuning

Data: neural-network model of layers [L_1, L_2,...,L_n]

Result: Layer-wise optimal α values {L_1: α_1, L_2: α_2,..., L_n: α_n}

1. Hook input & output values of all layers
2. Generate user-defined α values [α_1, α_2, ..., α_n]
3. for L_i in [L_1, L_2,...,L_n]:
4. for α_j in [α_1, α_2, ..., α_n]:
5. recalculate smoothing factor s(α_j, $W_i(L_i)$, $X_i(L_i)$)
6. adjust W_i& X_i -> W_i = X
7. $X_i = X_i \, \mathrm{diag}(s)^{-1}$
8. $W_i = \mathrm{diag}(s) \, W_i$ # Adjust parameters
9. $Xq_i = per\_tensor\_quant\_dequant(X_i)$
10. $Wq_i = per\_channel\_quant\_dequant(W_i)$
11. $Yq_i = Wq_i * Xq_i$ # Predict output
12. $loss = MSE(Yq_i, Y_i)$
13. record loss and recover parameters

本文提出的方法支持用多个标准（如最小值、最大值和平均值）来确定 Transformer 块的输入层归一化（LayerNorm）操作的 α 值。实验发现，将 α 范围设为 [0.3, 0.7]，步长设为 0.05，对大多数模型来说都能达到很好的平衡。

这一方法有两个显著特点：一是全自动化，二是比原始方法支持的融合模式多。

下图提供了在 BLOOM-1b7 模型上执行 SmoothQuant α 值自动调优的样例代码：

```
from neural_compressor.adaptor.torch_utils.smooth_quant import TorchSmoothQuant
model_name = 'bigscience/bloom-1b7'
model = transformers.AutoModelForCausalLM.from_pretrained(model_name, torchscript=True)
alpha = 'auto'
sq = TorchSmoothQuant(model, dataloader)
sq.transform(alpha)
```

启用增强型 SmoothQuant 的样例代码

用户只需传递一个模型名称（model_name）和一个数据加载器。值得注意的是，模型分析主要依靠的是 Torch JIT。用户可以在加载 Hugging Face 模型（https://huggingface.co/models）时将 torchscript 设置为 True，或将 return_dict 设置为 False。更多信息请参阅英特尔® Neural Compressor 文档（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。

结果

本文提出的增强型 SmoothQuant 的主要优势在于提高了准确率。

经过对多种主流大语言模型的评估，具备自动调优能力的 INT8 SmoothQuant 最后一个词元（last-token）的预测准确率要高于原始 INT8 SmoothQuant 和 FP32 基线方法。详见下图：

Model\Last token accuracy	FP32	INT8 (w/o SmoothQuant)	INT8 (w/ SmoothQuant)	INT8 (w/ SmoothQuant auto tuning)
bigscience/bloom-560m	65.20%	63.44%	66.48% (alpha=0.5)	64.76% (alpha: 95.9% over 0.6, 4.1% in [0.4, 0.6])
bigscience/bloom-1b7	71.43%	67.78%	72.56% (alpha=0.5)	72.58% (alpha: 55.1% over 0.6, 30.6% in [0.4, 0.6], 14.3% under 0.4)
bigscience/bloom-3b	73.97%	69.99%	74.02% (alpha=0.5)	74.16% (alpha: 100% over 0.6)
bigscience/bloom-7b1	77.44%	75.46%	77.02%(alpha=0.5)	77.45% (alpha: 91.8% over 0.6, 4.9% in [0.4, 0.6], 3.3% under 0.4)
bigscience/bloom-176b	84.17%	82.13%	83.52% (alpha=0.6)	-
facebook/opt-125m	63.89%	63.48%	63.44% (alpha=0.5)	64.14% (alpha: 59.4% over 0.6, 8.1% in [0.4, 0.6], 32.4% under 0.4)
facebook/opt-1.3b	75.41%	73.59%	70.94% (alpha=0.5)	74.80% (alpha: 69.9% over 0.6, 24.7% in [0.4, 0.6], 5.5% under 0.4)
facebook/opt-2.7b	77.79%	78.57%	78.60%(alpha=0.5)	78.25% (alpha: 73.2% over 0.6, 21.6% in [0.4, 0.6], 5.2% under 0.4)
facebook/opt-6.7b	81.26%	76.65%	81.58%(alpha=0.5)	81.39% (alpha: 68.0% over 0.6, 26.8% in [0.4, 0.6], 5.2% under 0.4)
EleutherAI/gpt-j-6B	79.17%	78.82%	78.84%(alpha=0.6)	79.29% (alpha: 96.4% over 0.6, 3.6% in [0.4, 0.6])

FP32 基线方法、INT8（启用和不启用 SmoothQuant）以及 INT8（启用本文提出的增强型 SmoothQuant）的准确率对比

从上图可以看出，在 OPT-1.3b 和 BLOOM-1b7 模型上，本文提出的增强型 SmoothQuant 的准确率比默认的 SmoothQuant 分别高 5.4% 和 1.6%。量化后的模型也缩小到 FP32 模型的四分之一，大大减少了内存占用空间，从而有效地提升大模型在英特尔® 平台上的推理性能。

更全面的结果请见 GitHub 存储库（https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md）。同时，也欢迎您创建拉取请求或就 GitHub 问题（https://github.com/intel/neural-compressor/issues）发表评论。期待听到您的反馈意见和建议。

作者:

英特尔公司人工智能资深架构师沈海豪、英特尔公司人工智能资深软件工程师程文华、英特尔公司人工智能软件工程师陆崟彤、何欣、郭恒、王畅、王梦妮，他们都在从事模型量化及压缩的研究与优化工作。