

SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot

复现者: Zujie Lin

摘要

我们首次证明, 大规模生成式预训练 transformer (GPT) 族模型可以在不进行任何重新训练的情况下, 一次性剪枝到至少 50% 的稀疏性, 而且精度损失极小。这是通过一种名为 SparseGPT 的新剪枝方法实现的, 该方法专门设计用于高效、准确地处理大规模 GPT 族模型。我们可以在 4.5 小时内对现有最大的开源模型 OPT-175B 和 BLOOM-176B 执行 SparseGPT, 并且可以达到 60% 的非结构稀疏性, 而复杂度的增加可以忽略不计: 值得注意的是, 在推理时可以忽略这些模型中超过 1,000 亿个权重。SparseGPT 适用于半结构化 (2:4 和 4:8) 模式, 并与权重量化方法兼容。我通过实验复现这篇论文, 最终达到了接近原文中的结果。

关键词: 大语言模型; LLMs; 模型剪枝; Transformer; 模型压缩

1 引言

Generative Pretrained Transformer (GPT) 系列中的大型语言模型 (LLM) 在各种任务中都表现出了卓越的性能, 但由于其庞大的体积和计算成本而难以部署。举例来说, 性能最好的 GPT-175B 模型有 1750 亿个参数, 以半精度 (FP16) 格式计算, 总存储量至少为 320GB (1024 的倍数), 因此推理至少需要 5 个 A100 GPU, 每个 GPU 有 80GB 内存。因此, 通过模型压缩来降低这些成本自然受到了广泛关注。迄今为止, 几乎所有现有的 GPT 压缩方法都侧重于量化 [2, 4, 26, 27], 即降低模型数值表示的精度。

剪枝是压缩的一种补充方法, 它可以去除网络元素, 从单个权重 (非结构化剪枝) 到更高粒度的结构, 如权重矩阵的行/列 (结构化剪枝)。剪枝已有很长的历史 [10, 17], 并已成功应用于视觉和较小规模的语言模型 [12]。然而, 性能最好的剪枝方法需要对模型进行大量的再训练才能恢复准确性。反过来, 这对于 GPT 规模的模型来说又是极其昂贵的。虽然存在一些精确的单次剪枝方法 [4, 13], 可以在不重新训练的情况下压缩模型, 但不幸的是, 即使是这些方法, 在应用于拥有数十亿参数的模型时也会变得非常昂贵。因此, 迄今为止, 基本上还没有对十亿参数模型进行精确剪枝的工作。

在本篇论文中提出了 SparseGPT, 这是第一种精确的单次剪枝方法, 能在拥有 100+ 亿个参数的模型规模上高效工作。SparseGPT 的工作原理是将剪枝问题简化为一组极大规模的稀疏回归实例。然后, 它通过一个新的近似稀疏回归求解器来求解这些实例, 该求解器的效率足以在单个 GPU 上几个小时内执行公开可用的最大 GPT 模型 (1750 亿参数)。同时,

SparseGPT 的精确度足以在剪枝后降低到可以忽略不计的程度，无需任何微调。例如，当在最大的公开语言生成模型（OPT175B 和 BLOOM-176B）上执行时，SparseGPT 一次就能诱导出 50-60% 的稀疏性，而准确率损失很小，无论是以困惑度还是零次准确率来衡量都是如此。SparseGPT 的一个显著特点是它完全是局部的，即它只依赖于权重更新来保持每一层的输入输出关系，而权重更新的计算不需要任何全局梯度信息。

2 相关工作

2.1 剪枝方法

据我们所知，我们是第一个研究 GPT-scale 模型（例如参数超过 100 亿的模型）剪枝的人。造成这一惊人差距的一个原因是，大多数现有的剪枝方法，例如 [8, 9, 15]，需要在剪枝步骤后进行大量的再训练才能恢复准确性，而 GPT-scale 模型通常需要大量的计算和参数调整来进行训练或微调 [28]。SparseGPT 是 GPT-scale 模型的后训练方法，因为它不进行任何微调。迄今为止，只在经典 CNN 或 BERT 类型模型的规模上研究过训练后剪枝方法 [4, 13, 16]，这些模型的权重比我们感兴趣的模型少 100-1000 倍。

2.2 后训练量化

相比之下，在量化开放 GPT-scale 模型的后训练方法方面已有大量工作 [25, 28]。具体来说，ZeroQuant[27]、LLM.int8()[2] 和 nuQmm[20] 方法研究了对十亿参数模型进行四舍五入量化的可行性，结果表明通过这种方法对权重进行 8 位量化是可行的，但由于离群特征的存在，激活量化可能比较困难。Frantar 等人 [4] 利用近似二阶信息将权重精确量化到 2-4 位，适用于最大的模型，并展示了当与高效 GPU 内核结合时，生成批量大小为 1 的推理速度提高了 2-5 倍。后续工作 [26] 研究了将激活和权重量化到 8 位的联合方案，提出了一种基于平滑的方案，该方案降低了激活量化的难度，并辅以高效的 GPU 内核。Park 等人 [21] 通过四元组可学习参数解决了量化激活异常值的难题，其目标是在保持其他模型参数不变的情况下，按通道扩展激活。Dettmers 等人 [2] 研究了大规模 LLM 的模型大小、量化位数和不同准确度概念之间的比例关系，观察到在不同任务中，复杂度得分和零点精度之间存在很高的相关性。正如我们在第 3.5 节中所展示的，SparseGPT 算法可与当前最先进的权重量化算法 GPTQ 结合使用，并与激活量化方法兼容 [21, 26]。

3 本文方法

3.1 本文方法概述

本研究涉及了对 OPT、BLOOM 等大模型的精确剪枝，对于实现 SparseGPT 算法，大概可以分为以下重要算法步骤：快速近似重构、自适应掩码选择、半结构化稀疏的扩展。其中最为重要的是实现快速近似重构和自适应掩码选择，这是设计 SparseGPT 算法的关键核心。

3.1.1 逐层剪枝

对于剪枝问题，一般都可以归结为：为每一层 l 找到一个具有一定目标密度的稀疏性掩码 M_l ，并可能更新权重 \widehat{W}_l ，从而到达以下目的：

$$\operatorname{argmin}_{\text{mask } \mathbf{M}_\ell, \widehat{\mathbf{W}}_\ell} \|\mathbf{W}_\ell \mathbf{X}_\ell - (\mathbf{M}_\ell \odot \widehat{\mathbf{W}}_\ell) \mathbf{X}_\ell\|_2^2. \quad (1)$$

3.1.2 掩码选择和权重重构

(1) 式中分层剪枝问题的一个关键方面是，掩码 M_l 和剩余权重 \widehat{W}_l 是共同优化的，这使得该问题具有 NP 难度 [1]。因此，对于较大的层来说，精确求解这个问题是不现实的，这导致所有现有的方法都只能求助于近似值。一种主流方法是将问题分为**掩码选择**和**权重重构**这两个步骤 [11, 13, 16]。具体来说，首先根据某些显著性标准（如权重大小）选择一个剪枝掩码 M [29]，然后在保持掩码不变的情况下优化剩余的未剪枝权重。重要的是，一旦掩码固定下来，(1) 式就会变成一个易于优化的线性平方误差问题。

3.1.3 现有解析器

早期工作 [14] 将迭代线性回归应用于小型网络。最近，AdaPrune 方法 [13] 通过基于幅度的权重选择，然后应用 SGD 步骤重构剩余权重，在现代模型上取得了很好的效果。后续工作表明，通过取消掩码选择和权重重建之间的严格分离，可以进一步提高剪枝的准确性。迭代式 AdaPrune [5] 以渐进的步骤执行剪枝，中间进行重新优化，而 OBC [4] 则引入了一种贪婪求解器，通过高效的闭式方程，一次一次地去除权重，并在每次迭代后完全重建剩余权重。

3.1.4 扩展到 100 亿 + 参数的难度

之前的后训练技术都是为了精确压缩高达几亿个参数的模型而设计的，只需几分钟到几小时的计算时间。然而，我们的目标是将模型稀疏化到 1000 倍的规模。即使是 AdaPrune 这种在速度/精度之间进行了理想权衡的优化方法也需要几个小时才能对仅有 13 亿个参数的模型进行稀疏化，而对 175B Transformers 而言，则需要几百个小时（几周）的线性缩放。更精确的方法至少比 AdaPrune [5] 昂贵几倍，甚至比线性扩展 [4] 还差。这表明，将现有的精确后训练技术扩展到超大模型是一项具有挑战性的工作。因此，我们提出了一种新的分层求解器 SparseGPT，它基于对封闭式方程的仔细近似，在运行时间和准确性方面都能轻松扩展到大模型。

3.2 快速近似重构

3.2.1 快速近似重构算法的提出动机

对于固定的剪枝掩码 M ，可以通过解决与每个矩阵行 w^i 相对应的稀疏重构问题，精确计算出掩码中所有权重的最优值：

$$\mathbf{w}_{\mathbf{M}_i}^i = (\mathbf{X}_{\mathbf{M}_i} \mathbf{X}_{\mathbf{M}_i}^\top)^{-1} \mathbf{X}_{\mathbf{M}_i} (\mathbf{w}_{\mathbf{M}_i} \mathbf{X}_{\mathbf{M}_i})^\top \quad (2)$$

其中, X_{M_i} 仅表示第 i 行相应权重未被剪枝的输入特征子集, w_{M_i} 表示它们各自的权重。然而, 这需要对与第 i 行的剪枝掩码 M_i 所保留的值相对应的 Hessian 矩阵 $H_{M_i} = X_{M_i} X_{M_i}^\top$ 求逆, 即对所有 $1 \leq i \leq d_{row}$ 的行分别计算 $H_{M_i}^{-1}$ 。一次这样的求逆需要 $O(d_{col}^3)$ 的时间, 在 d_{row} 行上的总计算复杂度为 $O(d_{row} \cdot d_{col}^3)$ 。对于 Transformer 模型, 这意味着整体运行时间与隐藏维度 d_{hidden} 的四次方成正比; 我们需要将速度至少提高 d_{hidden} 的整数倍, 才能获得实用的算法。

3.2.2 不同的 Row - Hessian 挑战

根据公式 (2) 优化重构未剪枝权重的计算复杂度较高, 主要是由于求解每一行都需要对 $O(d_{col} \times d_{row})$ 矩阵进行单独求逆。这是因为行掩码 M_i 通常是不同的, 而且 $H_{M_i}^{-1} \neq (H^{-1})_{M_i}$ 。图 1 说明了这一点。如果所有行掩码都相同, 那么只需要计算一个共享逆, 因为 $H = X X^\top$ 只取决于层输入, 而层输入对所有行都是相同的。

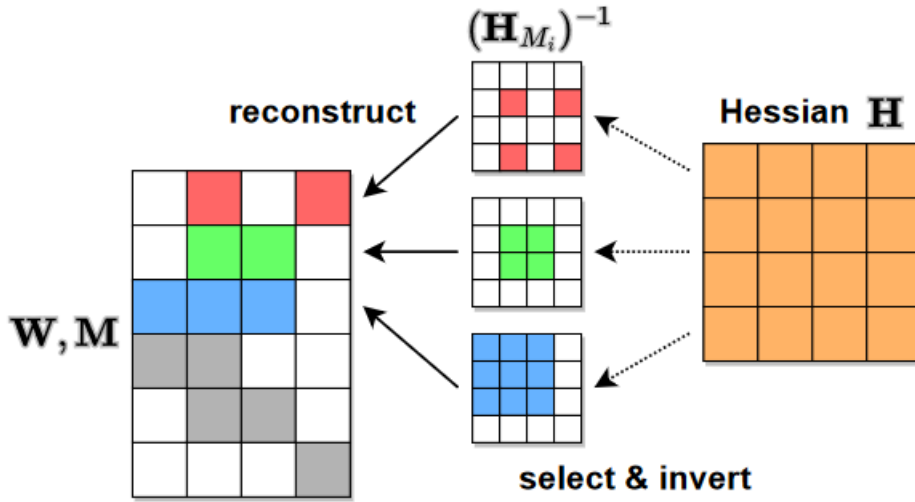


图 1. row-Hessian 挑战的说明: 行是独立稀疏化的, 剪枝后的权重是白色的。

这种约束可以在选择掩码时强制执行, 但这将对最终模型的准确性产生重大影响, 在大结构 (如整列) 中稀疏权重比单独剪枝权重要困难得多。设计一种既准确又高效的近似算法, 关键在于如何在具有不同剪枝掩码的行之间重复使用 Hessians。

3.2.3 等效迭代

为了实现快速近似重构算法, 这里要从不同的迭代角度, 利用经典的 OBS[7, 10, 23] 更新行级权重重构。假定损失为二次近似值, 当前权重 w 为最优权重, 那么 OBS 更新 δ_m 将对剩余权重进行最优调整, 以弥补索引 m 处权重的移除, 从而产生误差 ϵ_m :

$$\delta_m = -\frac{w_m}{[\mathbf{H}^{-1}]_{mm}} \cdot \mathbf{H}_{:,m}^{-1}, \quad \epsilon_m = \frac{w_m^2}{[\mathbf{H}^{-1}]_{mm}}. \quad (3)$$

由于对 W 的一行进行分层剪枝所对应的损失函数是二次函数, 因此 OBS 公式在这种情况下是精确的。因此, $w + \delta_m$ 是对应于掩码 $\{m\}^C$ 的最优权重重建。

3.2.4 最优部分更新

应用 OBS 更新 δ_m 可能会调整 (当前掩码 M 中的) 所有可用参数的值, 以补偿 w_m 的删除。但是, 如果我们只更新剩余未剪枝权重中 $U \subset M$ 子集的权重, 仍然可以从误差补偿中获益, 只使用 U 中的权重, 同时降低应用 OBS 的成本。所以这里只需要使用 H_U 计算 OBS 更新, 而不是用 H_M , 只更新 w_U 就可以实现这部分更新。最重要的一点在于, 由于 $|U| < |M|$, 那么求逆 H_U 会比求逆 H_M 快很多。

3.2.5 Hessian 同步

假设输入特征 $j = 1, \dots, d_{col}$, 然后递归定义 d_{col} 索引子集 U_j 的序列为:

$$U_{j+1} = U_j - \{j\} \text{ with } U_1 = \{1, \dots, d_{col}\} \quad (4)$$

即从所有索引集合 U_1 开始, 每个子集 U_{j+1} 都是通过从上一个子集 U_j 中删除最小的索引来创建的。最重要的是, 根据研究 [4], 通过进一步高斯消元, 在 $O(d_{col}^2)$ 时间内从 $B = (H_{U_j})^{-1}$ 中移除与原始 H 中 j 相应的第一行和第一列, 就可以高效计算出更新后的逆值 $(H_{U_{j+1}})^{-1}$:

$$(\mathbf{H}_{U_{j+1}})^{-1} = \left(\mathbf{B} - \frac{1}{[\mathbf{B}]_{11}} \cdot \mathbf{B}_{:,1} \mathbf{B}_{1,:} \right)_{2:,2:} \quad (5)$$

这里 $(H_{U_1})^{-1} = H^{-1}$ 。因此, 递归计算求逆整个 d_{col} 海森矩阵只需要 $O(d_{col}^3)$ 时间。

一旦某个权重 w_k 被剪枝, 就不需要再更新。当剪枝 w_k 时, 我们希望更新尽可能多的未剪枝权重, 以获得最大的误差补偿。这就导致了以下策略: 依次遍历 U_j 及其相应的 $(H_{U_j})^{-1}$, 如果 $j \notin M_i$ 则剪枝 M_i 。每个 $(H_{U_j})^{-1}$ 只计算一次, 并在权重 j 属于剪枝掩码的所有行中重复使用以去除权重 j 。图2是该算法的可视化示意图。

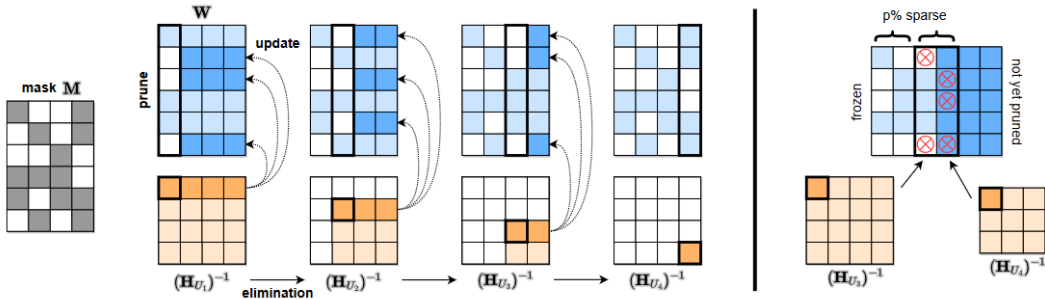


图 2. SparseGPT 重建算法的可视化。

3.2.6 权重冻结

虽然我们已将 SparseGPT 算法作为使用最优部分更新的精确重构的近似方法, 但是我们可以考虑一种精确贪婪框架, 它可以逐列压缩权重矩阵, 在每一步中 [6] 始终优化更新所有尚未压缩的权重。看起来 SparseGPT 似乎并不符合这一框架, 因为我们只压缩了每列中的部分权重, 而且也只更新了未压缩权重的子集。然而, 从机制上讲, “压缩” 权重最终意味着将其固定为某个特定值, 并确保其在未来的更新中不再被 “解压缩”, 即被冻结。因此, 通过将列式压缩定义为:

$$\text{compress}(\mathbf{w}^j)_i = 0 \text{ if } j \notin M_i \text{ and } w_i^j \text{ otherwise} \quad (6)$$

也就是将不在掩码中的权重归零，并将其余权重固定为当前值，我们的算法可以解释为一种精确的列式贪婪方案。从这个角度出发，我们可以将稀疏化和量化合并为一个压缩过程。

3.3 自适应掩码选择

最近的研究 [4] 表明，由于相关性，剪枝过程中的更新会显著改变权重，在选择掩码时考虑到这一点会产生更好的结果。通过在运行重构时自适应地选择掩码，SparseGPT 可以整合这一观点。

普通的方法是在压缩每列 i 时，选择最容易剪枝的 $p\%$ 权重，从而实现 $p\%$ 的整体稀疏性。但是这种方法的缺点是，稀疏性不能非均匀地分布在各列中，从而带来额外的不必要结构。这对于大规模语言模型来说尤其棘手，因为这些模型有少量高度敏感的离群特征 [2, 26]。

我们通过迭代阻塞消除了这一缺点。更确切地说，我们每次都会根据公式 (3) 中的 OBS 重建误差 ϵ ，利用海森矩阵中的对角线值，选择 $B_s = 128$ 列的剪枝掩码。然后，进行下一个 B_s 权重更新，再为下一个区块选择掩码，依此类推。这一过程允许对每一列进行非均匀选择，特别是还使用了相应的海森矩阵信息，同时还考虑了之前的权重更新进行选择。

4 复现细节

4.1 与已有开源代码对比

本论文在 [Github](#) 已开源，本人在复现工作过程中，认真学习作者是如何实现 SparseGPT 算法，参考引用了快速近似重构算法和自适应掩码选择算法的实现。引用代码部分如图3所示：

```
H = self.H
del self.H
# 这行代码首先调用 torch.diag(H) 来提取张量 H 的对角线元素，然后通过 == 0 比较操作，检查这些对角线元素是否等于0。
# 如果对角线元素等于0，则对应的结果为 True；否则为 False。这样，dead 成为一个布尔型张量，其每个元素表示 H 的对角线上相应元素是否为0。
dead = torch.diag(H) == 0
# 这行代码使用 dead 张量作为索引，来修改 H 张量。具体来说，它将 H 的对角线上那些原本为0的元素设置为1。
# 这是一种常见的技术，用于避免数值计算中的除以零错误或提高数值稳定性。
H[dead, dead] = 1
# 这行代码同样使用 dead 张量作为索引，但这次是对 W 张量进行操作，它将 W 中所有与 dead 中为 True 的列对应的元素设置为0。
# 这意味着如果 H 的某个对角线元素原本是0（即 dead 中相应元素为 True），那么 W 中对应的整列都会被设置为0。
W[:, dead] = 0

Losses = torch.zeros(self.rows, device=self.dev)

# 对于较小的模型，采用dampening，即在H的对角线元素上添加一个小常数（我们总是选择平均对角线值的 1%），似乎足以避免数值问题。
damp = percdamp * torch.mean(torch.diag(H))
# 这个张量用于海森矩阵H的对角线元素
diag = torch.arange(self.columns, device=self.dev)
# 在H的对角线元素上添加dampening
H[diag, diag] += damp
# 这行代码执行Cholesky分解。Cholesky分解是一种将正定矩阵分解为一个下三角矩阵和其转置的上三角矩阵的乘积的方法。
H = torch.linalg.cholesky(H)
# 这行代码计算Cholesky分解后的矩阵 H 的逆。torch.cholesky_inverse 是一种高效计算逆矩阵的方法，特别是当矩阵已经通过Cholesky分解时。
H = torch.cholesky_inverse(H)
# 这行代码再次执行Cholesky分解，但这次是生成上三角矩阵。参数 upper=True 指定了生成的是上三角矩阵。
H = torch.linalg.cholesky(H, upper=True)
# 这行代码将 H 赋值给 H_inv。这里 H_inv 可能代表了 H 的逆矩阵。
H_inv = H
```

图 3. 引用快速近似重构算法和自适应掩码选择算法的部分实现。

在模型压缩领域，剪枝是一种重要的方法，但是量化也是一种实用技巧。在 follow 了作者另一篇关于模型量化的工作后，作者提出了一种后训练量化方法 GPTQ[6]。因此我考虑将剪枝和量化这两种方法相结合，在模型剪枝过程中辅以量化技巧，这可以进一步压缩模型，提升效果。图4中的代码就是线性量化的核心部分：

```
"""
    首先将浮点数x量化为整数表示，然后再将这些量化的整数值转换回接近原始浮点数值的形式
    r = s * (q - z)
"""
def quantize(x, scale, zero, maxq):
    # 将x进行量化到整数q
    q = torch.clamp(torch.round(x / scale) + zero, 0, maxq)
    # 然后根据整数q，反量化还原出它对应的浮点数张量r' = scale * (q - zero)
    return scale * (q - zero)
```

图 4. 线性量化的核心实现部分。

4.2 实验部分

4.2.1 实验环境搭建

在本次实验中，完全使用 PyTorch 框架 [22] 实现，并且使用 HuggingFace Transformers library[24] 处理模型和数据集。所有剪枝实验均在内存为 80GB 的单块 NVIDIA A800 GPU 上进行。

4.2.2 模型，数据集，评估验证

这里主要使用 OPT 模型 [28]，但也考虑了 1760 亿参数版本的 BLOOM[25]。虽然重点是最大的变体，但同时也展示了一些较小模型的结果，以提供更广泛的信息。

在度量指标方面，主要关注困惑度 perplexity，困惑度是一个具有挑战性的稳定指标，非常适合评估压缩方法的准确性 [3, 4, 27]。此外还考虑了 raw-WikiText2 测试集 [19] 和 PTB 测试集 [18] 以及 C4 验证数据的子集，这些都是 LLM 压缩文献中常用的 benchmarks[6, 20, 26, 27]。我们注意到，我们评估的重点在于稀疏模型相对于密集基线的准确性，而不是绝对数字。不同的预处理可能会影响绝对准确性，但对我们的相对要求影响不大。困惑度的计算完全遵循 HuggingFace 所描述的程序，使用全步长。

4.2.3 Baselines

这里将其与标准幅度剪枝基线 [29] 进行了比较，后者是使用了 layer-wise，可扩展到最大的模型。对于高达 1B 参数的模型，还与 AdaPrune[13] 进行了比较，这是现有精确训练后剪枝方法中最有效的一种。

4.3 使用说明

在命令行中使用 Python 命令可以指定要运行的模型和数据集，联网下载好的模型（比如 OPT、BLOOM 模型）是放在 `/.cache/huggingface/hub` 文件夹下，联网下载好的数据集（比如 raw-wikitext2、ptb、c4）是放在 `/.cache/huggingface/datasets` 文件夹下。在项目目录中：

1. `sparsegpt.py` 文件定义了 SparseGPT 类，实现了本篇论文所提出的 SparseGPT 剪枝算法。
2. `quant.py` 文件是在模型剪枝过程中辅以进行线性量化，实现了线性量化的逻辑过程。
3. `opt.py` 文件中就是使用构建好的 SparseGPT 算法来对 OPT 模型进行剪枝。
4. `modelutils.py` 文件中实现了递归搜索模型中所有指定类型的层，并返回一个字典，其中包含了所有指定类型的层及其在模型中的名称。
5. `datautils.py` 文件就是一个工具文件，里面设置了随机数种子，根据模型名加载其相应的分词器 tokenizer，加载 Wikitext-2 数据集、ptb 数据集、c4 数据集，最后根据提供的数据集名称 name、样本数量 nsamples、随机种子 seed、序列长度 seqlen、模型名称 model 来获取相应的数据加载器。

4.4 创新点

在本次实验的复现过程中，我们注意到所选用的模型在新数据集上的拟合效果较差。如果只对模型进行剪枝，效果其实不算太好。根据作者另一项工作提出的量化方法 GPTQ，在此结合了量化技巧，在剪枝过程中同时进行了线性量化。此时发现当把剪枝和量化这两种模型压缩技巧相结合后，最终效果比只做模型剪枝还要好一些。接下来，我将详细阐述线性量化的核心思路和求解过程。

4.4.1 量化的基本理解

量化是将输入从连续的或大的数值集限制为离散的数值集的过程。一般地，量化可分为 K-Means-based Quantization 和 Linear Quantization。而线性量化又可分为对称量化和非对称量化，我在复现过程中使用的是对称量化。图5展示了量化的核心概念：

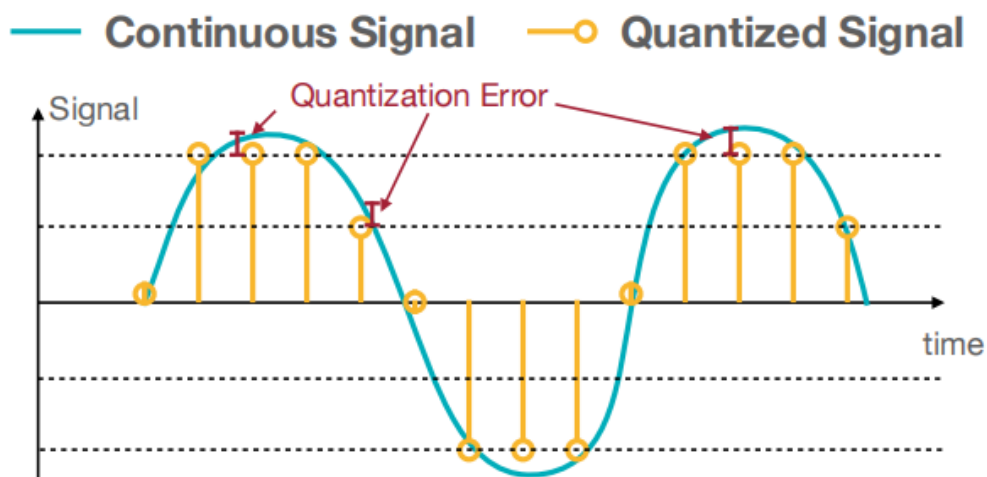


图 5. 量化过程的核心概念理解。

图中输入值与其量化值之间的差值 Quantization Error 称为量化误差。

4.4.2 线性量化的核心思路

线性量化实现了整数到实数的仿射映射，它涉及将连续的或具有很高精度的信号转换为较低精度的形式。线性量化特指这种映射过程是线性的，即离散级别均匀分布。它最关键的核在于：

$$r = S(q - Z) \quad (7)$$

公式中的 r, S 都是浮点数， q, Z 都是整数。 r 是输入的原始向量， S 是缩放因子， Z 是零点， q 是量化后的整数。

图 6 展示了如何利用公式 (7) 来进行量化：

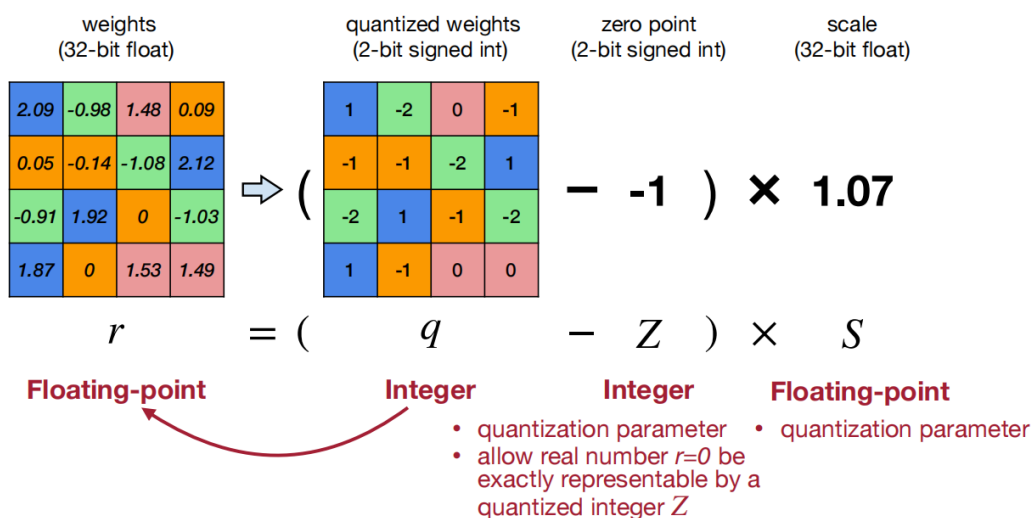


图 6. 线性量化过程的理解。

图 7 展示了线性量化的核心求解思路：

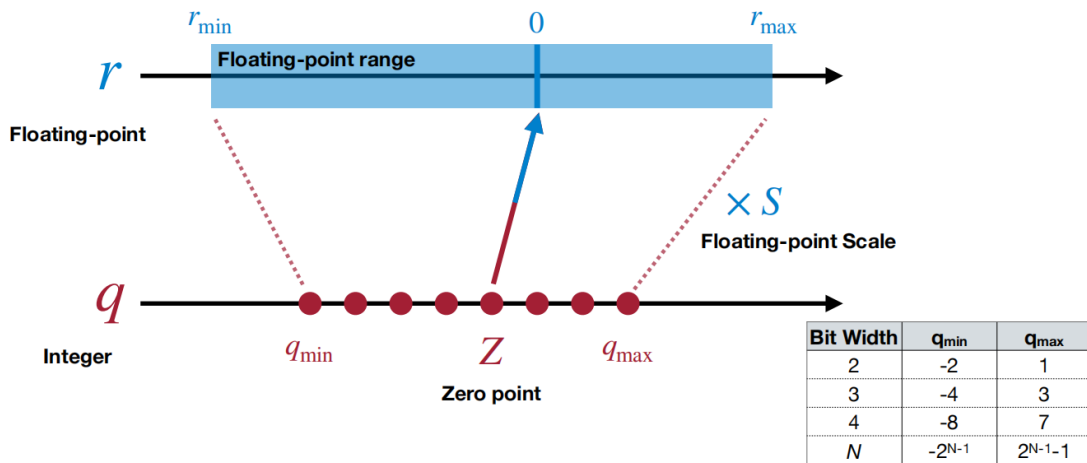
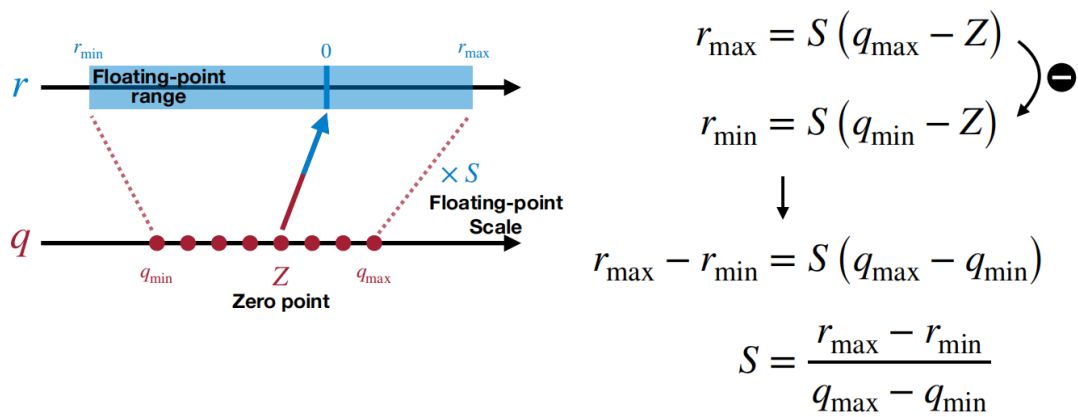


图 7. 线性量化的核心求解思路。

首先需要得到位宽范围，即 q_{\min} 和 q_{\max} 。如果是一个 n -bit 整数，那么可以得到位宽范围是 $[-2^{n-1}, 2^{n-1} - 1]$ 。例如一个 8-bit 整数，那么其位宽范围就是 $[-128, 127]$ 。其次需要得到缩放因子 S ，图 8 展示了缩放因子 S 的求解过程：



$$\begin{aligned}
 r_{\max} &= S(q_{\max} - Z) \\
 r_{\min} &= S(q_{\min} - Z) \\
 &\quad \ominus \\
 \downarrow \\
 r_{\max} - r_{\min} &= S(q_{\max} - q_{\min}) \\
 S &= \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}
 \end{aligned}$$

图 8. 缩放因子 S 的求解过程。

接着需要得到零点 Zero Point，图 9 展示了零点 Z 的求解过程：

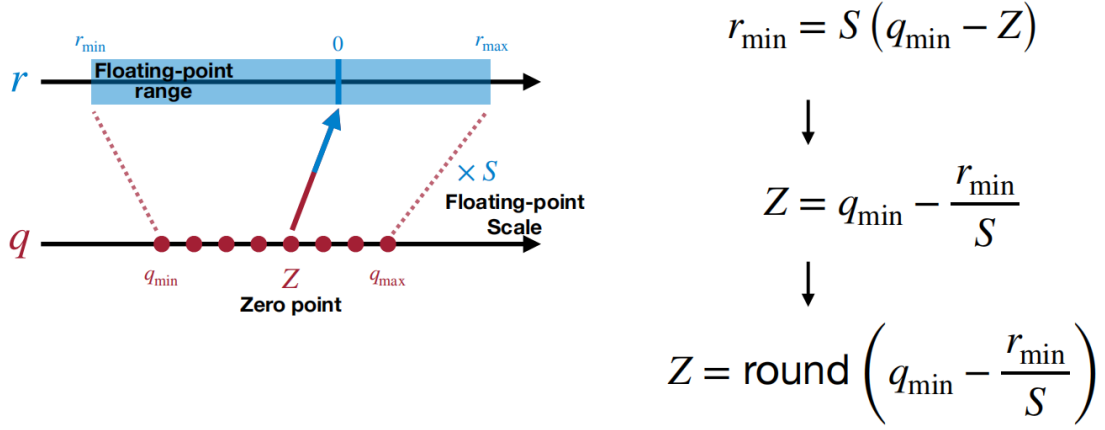


图 9. 零点 Z 的求解过程。

至此就完成了线性量化的核心解决思路，这里我再总结以下我所做的线性量化过程：一般来说 r 都是会被传参给定的浮点数，它是原始向量。当我们想要求出量化后的整数 q ，根据公式 (7) 可知，首先必须知道缩放因子 S 和零点 Z 。我们可以利用公式 (8) 求解出缩放因子 S ：

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \quad (8)$$

我们可以利用公式 (9) 求解出零点 Z ：

$$Z = \text{round}(q_{\min} - \frac{r_{\min}}{S}) \quad (9)$$

当求出缩放因子 S 和零点 Z 后，再利用公式 (7) 反解出量化后的整数 q ：

$$q = \text{int}(\text{round}(\frac{r}{S})) + Z \quad (10)$$

求出量化后的整数 q 后，再利用公式 (7) 去还原经过量化后的向量 r' ：

$$r' = S(q - Z) \quad (11)$$

此时 r 和 r' 之间会产生量化误差，那么只需要微调寻找合适的缩放因子 S 和零点 Z 即可。

5 实验结果分析

在本次实验中，我主要比较了不同的模型（OPT、BLOOM）在三种不同数据集上（wiki-text2、ptb、c4）的测试效果。评价指标是困惑度 perplexity，通过观察困惑度的变化情况来看评价 SparseGPT 算法的可行性，分析将其运用到不同模型的性能表现。

在表1中，展示了 OPT 模型在 raw-WikiText2 数据集上的困惑度情况：

在表2中，展示了 OPT 模型在 raw-WikiText2 数据集上的困惑度情况：

在表3中，展示了 OPT 模型在 c4 数据集上的困惑度情况：

表 1. 测试 OPT 模型在 raw-WikiText2 数据集上的困惑度

OPT-50%	125M	350M	1.3B	OPT	Sparsity	2.7B	6.7B	13B	30B	66B
Dense	27.66	22.00	14.62	Dense	0%	12.47	10.86	10.13	9.56	9.34
Magnitude	193.0	97.80	1.7e4	Magnitude	50%	265.0	969.0	1.2e4	168.0	4.2e3
AdaPrune	58.66	48.46	32.52	SparseGPT	50%	12.88	11.30	10.81	9.84	9.38
SparseGPT	33.22	28.79	26.61	SparseGPT	4:8	13.64	11.92	11.20	10.15	9.53
				SparseGPT	2:4	14.90	12.94	11.80	10.44	9.79

表 2. 测试 OPT 模型在 ptb 数据集上的困惑度

OPT-50%	125M	350M	1.3B	OPT	Sparsity	2.7B	6.7B	13B	30B	66B
Dense	38.99	31.07	20.29	Dense	0%	17.97	15.77	14.52	14.04	13.36
Magnitude	276.0	126.0	3.1e3	Magnitude	50%	265.0	969.0	1.2e4	168.0	4.2e3
AdaPrune	92.14	64.64	41.60	SparseGPT	50%	12.88	16.34	15.07	14.28	13.62
SparseGPT	54.23	41.29	24.18	SparseGPT	4:8	19.72	15.83	14.28	14.51	13.76
				SparseGPT	2:4	21.47	18.58	15.71	14.47	13.97

在复现完成之后，通过实验结果发现，当模型较小时（比如 125M、350M、1.3B）得到的结果比原文还要稍微好一些。但是当模型规模增大时（比如 2.7B、6.7B、13B、30B、66B）得到的结果虽然和原文相差不大，但是总体效果还是不如原文。我总结这个问题的原因如下：

1. 随机数种子。选取不同的随机数种子会导致不同的结果，由于并不清楚作者在实验室设定的随机数种子，因此这也会导致运行结果和原文相比有所优劣不同。
2. 算法逻辑实现不同。在实现快速近似重构算法和自适应随机选择算法时，对于区块大小的选择、随机掩码的选择会与作者实验时的不同，因此会有结果上的差异。
3. 线性量化过程不够准确完善。在剪枝过程中进行线性量化时，可能由于缩放因子 S 和零点 Z 没有寻找到最合适解，这就可能导致线性量化没有做到最优化，因此导致效果不如预期。

图 10 展示了当使用 SparseGPT 将整个 OPT 模型族压缩到不同的稀疏模式时，所呈现的模型和稀疏类型的困惑度。

表 3. 测试 OPT 模型在 ptb 数据集上的困惑度

OPT-50%	125M	350M	1.3B
Dense	26.56	22.59	16.07
Magnitude	141.0	77.04	403.0
AdaPrune	48.84	39.15	28.56
SparseGPT	33.22	28.79	26.61

OPT	Sparsity	2.7B	6.7B	13B	30B	66B
Dense	0%	14.34	12.71	12.06	11.45	10.99
Magnitude	50%	63.43	334.0	1.1e4	98.49	2.9e3
SparseGPT	50%	18.32	15.71	13.43	12.96	11.41
SparseGPT	4:8	18.66	15.68	14.31	12.47	11.77
SparseGPT	2:4	21.47	17.23	15.71	14.21	13.07

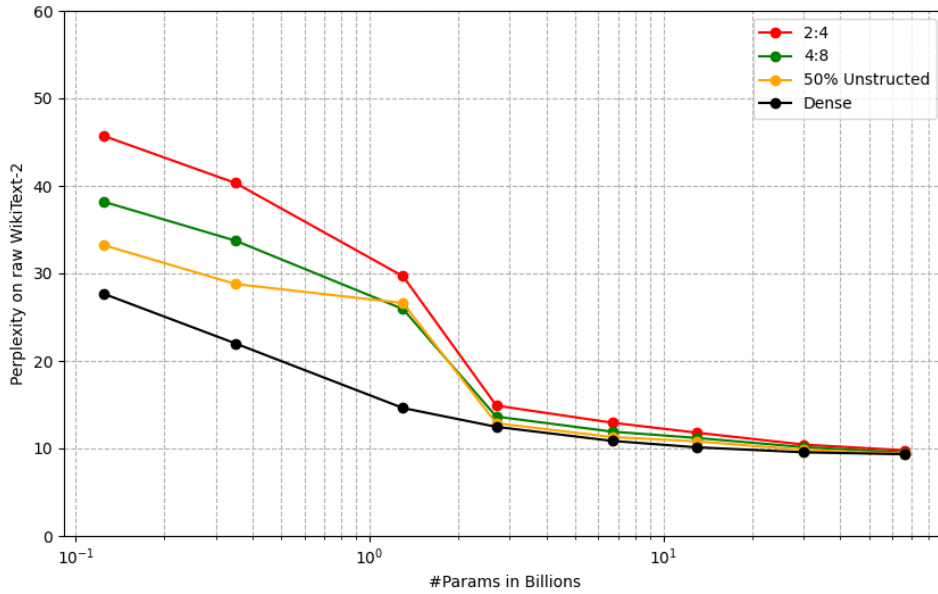


图 10. 当使用 SparseGPT 将整个 OPT 模型族压缩到不同的稀疏模式时，所呈现的模型和稀疏类型的困惑度。

从图中可以看出，当使用 SparseGPT 算法对 OPT 模型进行剪枝时，总体上来说随着参数量增大时，模型困惑度在下降。当模型特别大时，困惑度也能降到很低。这说明使用 SparseGPT 算法进行模型剪枝具有显著的效果，具备大模型剪枝的可行性，能够明显减少参数量，减轻了模型大小，提升模型性能，使得模型更加轻量化，更容易部署和推理。

6 总结与展望

本篇论文提出了一种新的训练后剪枝方法，称为 SparseGPT，专门为 GPT 系列的大规模语言模型定制。我们的研究结果首次表明，大规模生成预训练的 Transformer family 模型可以通过权重剪枝一次性压缩为高稀疏性模型，而不需要任何重新训练，并且准确率损失较低，这可以 perplexity 和 zero-shot 两方面来衡量。在复现论文进行实验中，我发现当模型较小时（比如 125M、350M、1.3B）得到的结果比原文还要要好。但是当模型规模增大时（比如 2.7B、6.7B、13B、30B、66B）得到的结果虽然和原文相差不大，但是总体效果还是不如原文。我并没有在最大规模 175B 上进行模型性能测试，这主要是因为目前 175B 模型已经不开源。但是

经过图表分析可知，对于 SparseGPT 算法来说，当模型越大时，它的剪枝效果越好，特别是配合量化技巧，模型压缩效率更高，困惑度会显著降低，模型性能更好，模型更小更轻量化。

在复现过程中，我有如下收获：

1. 本篇论文主要是关于大语言模型压缩，方法是利用一种后训练剪枝。在复现时，我掌握了模型剪枝的压缩技巧，对于模型剪枝理解更加深刻。

2. 在复现原文时，我也关注到了该作者团队相关工作，follow 了他们团队所提出了另一种模型压缩的技巧，也就是大语言模型量化。这我也对模型量化有了全新的认识和更加全面的理解。

3. 量化是一种特殊的剪枝，通过复现工作，我同时掌握了模型压缩的剪枝、量化这两种技巧，夯实了理论基础，提升了工程能力，同时这对于我接下来的研究课题提供了更多好的 insight 和 idea。

总的来说，这次实验为模型压缩领域特别是剪枝方面的研究提供了一系列有意义的探索和启示。但鉴于时间短暂、设备限制等原因，没有对该模型进行更为细致的研究与比较。未来，可以进一步探究更多模型剪枝方法，以期进一步提高模型压缩性能。同时，可以考虑结合其他特征工程方法或深度学习技术，来进一步优化模型压缩能力，使得模型更小更轻量化，更加容易推理和部署。同时在未来的科研工作中，我会将其和其他模型压缩技巧（比如量化、知识蒸馏、低秩分解等）结合起来，将这些方法运用到我研究课题中，进一步探索代码生成大模型领域的压缩和推理加速。

参考文献

- [1] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14:629–654, 2008.
- [2] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.
- [3] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pages 7750–7774. PMLR, 2023.
- [4] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.
- [5] Elias Frantar and Dan Alistarh. Spdy: Accurate pruning with speedup guarantees. In *International Conference on Machine Learning*, pages 6726–6743. PMLR, 2022.
- [6] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- [7] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34:14873–14886, 2021.

- [8] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.
- [9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [10] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.
- [11] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.
- [12] Torsten Hoefer, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.
- [13] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find $n:m$ transposable masks. *Advances in neural information processing systems*, 34:21099–21111, 2021.
- [14] Jason Kingdon and Jason Kingdon. Hypothesising neural nets. *Intelligent Systems and Financial Forecasting*, pages 81–106, 1997.
- [15] Eldar Kurtic and Dan Alistarh. Gmp*: Well-tuned global magnitude pruning can outperform most bert-pruning methods. *arXiv preprint arXiv:2210.06384*, 2022.
- [16] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35:24101–24116, 2022.
- [17] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [18] Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.
- [19] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.
- [20] Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

- [21] Minseop Park, Jaeseong You, Markus Nagel, and Simyung Chang. Quadapter: Adapter for gpt-2 quantization. *arXiv preprint arXiv:2211.16912*, 2022.
- [22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [23] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.
- [24] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface’s transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.
- [25] BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.
- [26] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.
- [27] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.
- [28] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models, 2022. URL <https://arxiv.org/abs/2205.01068>.
- [29] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.