Tempo: 通过减少内存占用来加速基于Transformer的模型训练

摘要

深度学习模型训练所需的计算成本可能相当高。过去的研究表明，增加批量大小有望提升整体吞吐量。然而，由于存储反向传播所需的激活/特征图，批量大小通常受到加速器内存容量的限制，因为较大的批量大小需要存储更大的特征图。最近，基于Transformer的模型因其优越性能和广泛适用性而备受欢迎，但也面临类似问题。为了解决这一问题，Tempo提出了一种有效利用加速器（例如GPU）内存资源来训练基于Transformer的模型的新方法。它主要对GELU、LayerNorm和Attention层进行了直接替代，降低了内存使用，从而实现更高效的训练。本文对Tempo 方法进行了复现，并将其应用于预训练语言模型Bert，通过实验评估了其在中文文本分类等下游任务中的性能。结果表明，Tempo相较于Baseline实现了约13%的显存优化，且不会导致模型识别精度下降。

关键词： 显存优化；深度学习；Transformer

1 引言

基于 Transformer 的模型，例如 BERT [1] 和 GPT-2 [2]，已经在许多一般自然语言处理任务中取得了成功，包括问答 [3]、释义 [4]、自然语言推理 [5]，甚至是外部领域图像识别等语言任务 [6]。然而，训练此类模型在时间和金钱资源方面可能非常昂贵 [7, 8]。例如，BERTLARGE 的预训练需要 4 天才能在 16 个 Cloud TPU（总共 64 个 TPU 芯片）上完成 [1]，成本约为10,000 美元 [9]。训练一个更新的基于 Transformer 的模型 GPT-3 的成本更加惊人——1200 万美元。因此，基于 Transformer 的模型的端到端训练时间即使小幅减少也很重要。尽管过去几年在使用专用硬件（例如 Google TPU、NVIDIA Tensor Cores）加速 Transformer 方面取得了重大进展，但依然存在的一个根本问题是它们受到以下因素的限制：硬件加速器的内存容量。例如，在训练序列长度为 512 的 BERT 时，即使批量大小为 1 也不适合具有 12GB 内存的现代 GPU。减少内存占用 [10–12]是允许更大批量训练的可行选择，从而提高硬件利用率并最终提高训练吞吐量 [13],。许多现有的减少内存占用的方法（例如卸载 [14]、检查点 [14] 和数据压缩/编码 [15, 16]）要么具有较高的计算开销，要么不适用直接转换为基于 Transformer 的模型。先前的方法分为两大类，对于基于 Transformer 的模型情况来说，这两种方法都不能令人满意。首先，这些技术可能过于笼统 [11, 16–19]，无法很好地利用基于 Transformer 的模型的细节，例如 Transformer [20]中使用的多头注意力机制，或者特定层，例如 LayerNorm层。检查点通过丢弃一些激活内存，在反向传播过程中进行重新计算，它可以有效减少模型的常驻内存，可以显着扩大批量大小，但它也带来很高的重计算开销（例如，

在一些先前的工作中观察到 30% 的性能下降 [12])。其次，如果之前的工作是具体的，那么它们会关注其他类型的模型/层，其想法不适用于 Transformer。例如，Gist 和 In-Place ABN 处理 CNN [15, 21]。

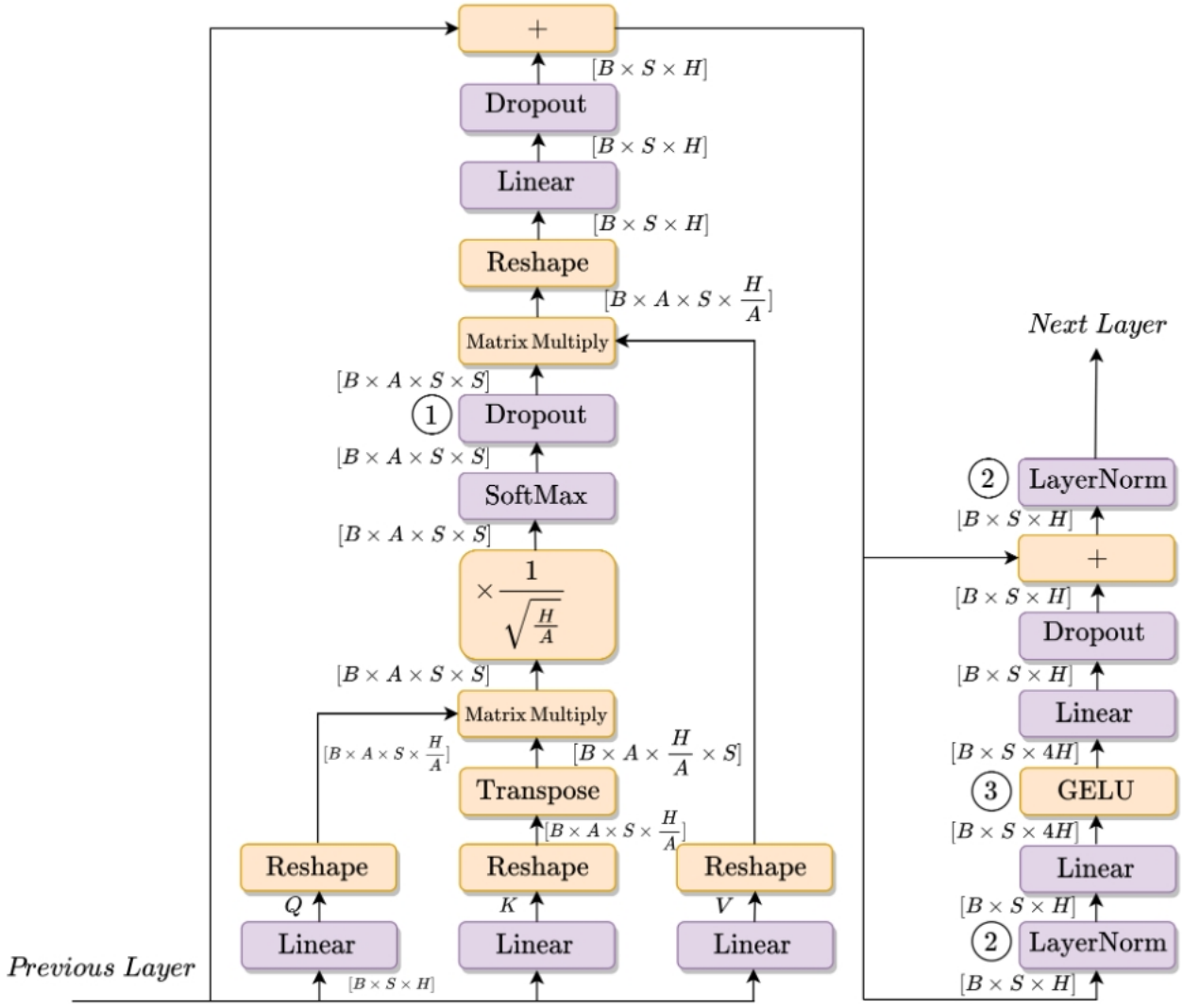


图 1. Bert中的Encoder结构

之前基于 Transformer 的模型的研究表明，虽然模型参数会影响内存占用，但训练期间的主要内存容量消耗实际上是激活特征图 [19, 22]。以Bert为例，大部分激活内存将用于每个BERT Transformer编码器层。对批量大小为 32、序列长度为128的MRPC微调任务上的 Huggingface BERTBASE 实现 [23] 进行分析表明，这些编码器激活特征图占用了总内存的 66%，因此如果能够找到减小激活特征图的方法，便有机会实现更低的内存开销。

Tempo证明了低开销内存占用的减少可以带来吞吐量的积极提高。此外，与之前的工作不利用基于 Transformer 的模型的细节不同，他们提出了一种专门为基于 Transformer 的模型量身定制的新方法。该方法提供了对GELU、LayerNorm、Dropout的层优化： (i) In-place GELU (ii) In-place LayerNorm (iii) Sub-Layer Dropout Recomputation。 In-place GELU 和 In-place LayerNorm 都使用替代推导来计算这些层的向后传递。这些推导允许丢弃通常在前向传递期间保留的一些激活（稍后在后向传递中使用），从而实现更高效的内存效率。 Sub-Layer Dropout Recomputation在前向传递期间丢弃高内存占用注意机制中的激活内存，然后

在向后传递期间重新计算这些激活值，而无需重新计算额外的不必要的张量。 Tempo 能够通过减少训练期间模型的总内存占用来提高更大批量的训练吞吐量。这是第一项专门针对基于 Transformer 的层探索内存占用优化的工作，它不仅显示了内存占用的减少，而且还显示了使用额外内存节省来实际增加吞吐量。 Tempo 通过瞄准总占用空间的主要部分——激活内存（反向传播所需的模型前向传播过程中保存的特征图来减少训练基于 Transformer 的模型的内存占用。所有提出的技术都可以大幅减少内存占用，同时吞吐量下降非常低（低至 1%）。Tempo实验结果显示，在现代 GPU 上，序列长度为 512 的 BERTLARGE 预训练批量大小提高了 2 倍，同时训练吞吐量提高了 16%。

2 相关工作

有许多工作 [24, 25]专注于开发与 BERT 和其他基于 Transformer 的模型保持竞争力的模型，同时需要更少的内存和计算。已采取的一种方法是解决注意力机制的 $O(S_2)$ 性质 [26]。其中，在减少内存占用方面，一些有趣且相关的想法涉及稀疏注意力机制 [27, 28]，或使用可分解的 softmax 层以避免首先进行 $O(S_2)$ 计算 [29]。另外还有以下方法可用于减少模型内存占用。

2.1 知识蒸馏

知识蒸馏的主要目标是在保持模型性能的同时减少模型的计算资源需求，核心思想是利用教师模型在训练数据上的丰富知识，指导学生模型学习。（例如 TinyBERT [25] 和 DistilBERT [24]）采用不同的方法来减小总模型大小。在这些作品中，蒸馏技术用于从较大的教师网络训练较小的学生网络。

2.2 检查点

这个方向的早期工作能够将层数一般线性模型的内存成本降低到$O(\sqrt{n})$。进一步的工作已在此基础上展开。 Echo 针对特定模型进行了创新性优化 [13]；Checkmate 计算最佳检查点计划，最大限度地减少特定内存预算的重新计算时间 [19]；而Dynamic Tensor Rematerialization则使用在线启发式方法来最大限度地减少训练时的重新计算时间 [18]。此外，该领域的一些工作将检查点与其他技术（例如下面详细介绍的卸载）相结合。

2.3 卸载

这个方向的初始工作（例如 vDNN）专注于根据计算成本卸载层输出的特定方案 [14]。还考虑了预取特征图，以预期它们在向后传递中的使用。这方面的更多工作集中在几个不同的方向。其一，超级神经元 [30] 结合了检查点和卸载。在这项工作中，他们检查计算成本低的层，避免传输开销，同时卸载计算成本高的层，避免这些情况下的重新计算开销。Capuchin 进一步考虑了张量级别的访问，以及在决定应用哪种策略时考虑运行时获取和计算时间 [11]。此外，ZeRO-Infinity 将特征图卸载与模型状态卸载和其他优化相结合 [17]。

2.4 混合精度

以前的 PLM（例如 BERT [1]）主要使用 32 位浮点数（FP32）进行预训练。近年来，为了预训练极大的语言模型，一些研究 [31] 开始利用 16 位浮点数（FP16），以减少内存使用和通信开销。此外，由于流行的 NVIDIA GPU（例如 A100）具有的 FP16 计算单元是 FP32 的两倍，FP16 的计算效率可以进一步提高。然而，现有的研究发现，FP16 可能导致计算精度的损失 [32]，影响最终的模型性能。为了缓解这个问题，一种替代方案称为 Brain Floating Point (BF16)已被用于训练，它比 FP16 分配更多的指数位和更少的有效位。对于预训练而言，BF16 通常比 FP16 在表示准确性方面表现更好 [33]。

2.5 模型压缩

像 Gist [15]和ActNN [16] 这样的作品都包含有损压缩的形式。 Gist 专门针对基于 CNN 的模型，通过各种不同的有损和无损优化，采用与我们类似的方法来仔细检查模型结构。然而，由于这是一项专注于 CNN 的工作，因此所描述的技术并不直接适用于Transformer。另一方面，ActNN 是一种更通用的技术，它使用一种量化策略，旨在减少特征图存储所需的位数，同时保留有关模型转换的某些理论保证。相对于Capuchin [11] 等技术，它还显示出良好的性能。然而，这项工作没有考虑基于Transformer的模型。

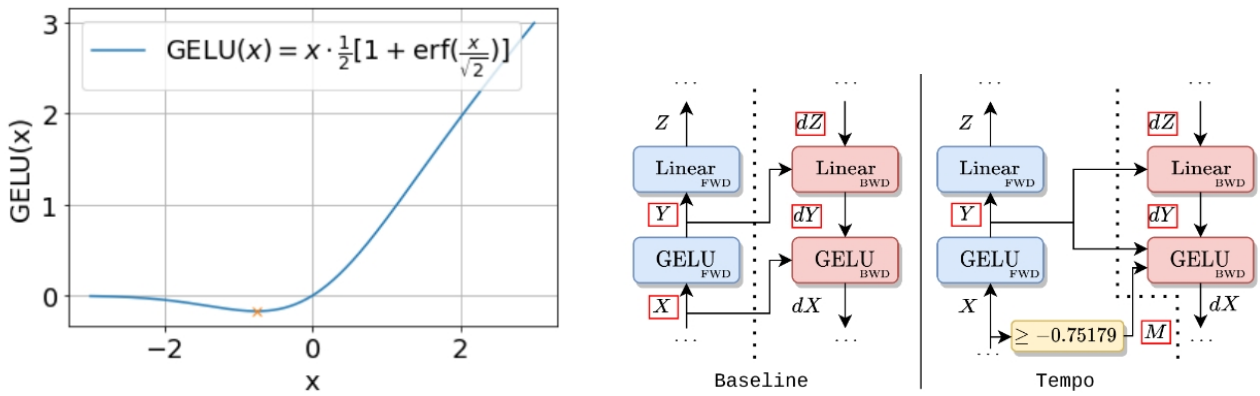
3 本文方法

3.1 本文方法概述

Tempo设计的主要思想：(i)In-place GELU (ii)In-place LayerNorm (iii)Sub-Layer Dropout Recomputation。所有这些想法背后的主题是在正常计算反向传播的同时，使用更少的存储空间。为此，In-place GELU 和 In- place LayerNorm 在原地计算每一层的输出，而不是使用输出激活来计算梯度。Sub- Layer Dropout Recomputation 也丢弃输出，并通过对 Dropout 层结构的仔细观察，能够在不进行过多的重新计算的情况下重新计算输出。

3.2 In-place GELU

GELU层被用作BERT层的前馈部分的激活函数（如图1中的标号③所示）。该函数的图形在图2(a)中展示。关于图2(b)中的基准线，请注意在反向传播中存储了X和Y。Y对于下游的全连接层是必要的，而X则为GELU层本身存储。之前的研究表明，某些激活函数，如ReLU，可以进行原地计算 [15]，即在不额外存储的情况下进行计算。如果我们能够在原地计算GELU函数，可能通过在反向传播中从输出中恢复输入，我们就能够节省存储X所需的空间。然而，这直接做到是不可能的。关于GELU函数的一个关键观察是它不是双射的——因此没有函数能够仅凭输出计算出输入而不需要额外的信息。然而，从图像可以观察到，GELU 函数既是连续的，又只有一个极值，即在 $x \approx \grave{}0.75179$ 处有一个最小值，如图2(a)所示。这意味着只需要额外的一小部分信息：输入是来自最小值的哪一侧，就可以计算出 GELU 函数的反函数。这是因为在最小值的每一侧，函数都是一对一的，因此在每个部分中都可以从输出中恢复输入。基于这一关键观察，我们可以舍弃输入，只保留 GELU 的输出，以及关于输入



(a) GELU函数图像　　(b) Baseline GEKU和In-place GELU

图 2. GELU函数

是否大于或等于最小值发生的位置的额外信息。图2(b)说明了Tempo与基准方法之间的差异。为了在实际系统上高效执行这个操作，可以将原始导数关于输入的表达式与函数的反函数进行复合，从而创建一个复合核。这个核包含对这个复合函数的多项式近似，这种近似是必要的，因为 GELU 是超越函数，因此反函数无法用初等函数解决.简而言之，这里的复合核是通过将原始导数与函数的反函数进行复合，然后进行多项式近似，以便在实际系统上高效执行 GELU 函数的操作。这种方法是一种权衡，因为 GELU 的反函数不能直接通过初等函数表示，所以采用了近似的方式来处理。

3.3 In-place LayerNorm

LayerNorm 层在 Transformer 编码器层中的多个位置使用，在图1中用②表示。通常情况下，LayerNorm 的梯度计算依赖于来自下一层的梯度输入，以及为此计算而存储的输入特征图 [34]。类似于 GELU，Tempo推导出 LayerNorm 层梯度的表达式，作为其输出的函数。在这个背景下，为了计算后续全连接层的梯度，必须存储 LayerNorm 的输出。通过这种方法，LayerNorm 的内存开销仅限于正向传播中计算的中间均值和方差。

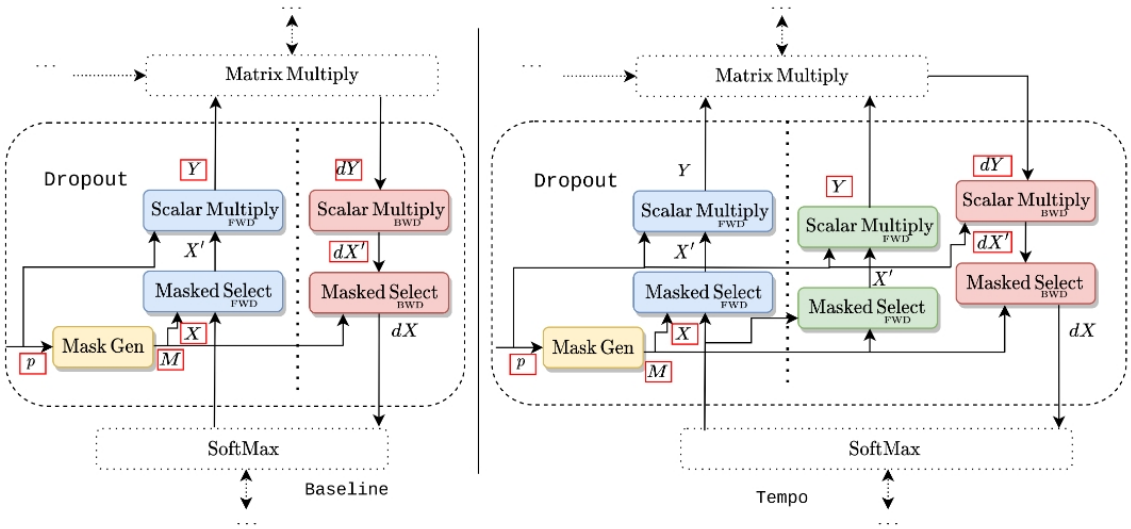


图 3. 基准dropout和Tempo实现的dropout

3.4 Sub-Layer Dropout Recomputation

在这一部分，Tempo探讨了子层面的检查点技术的思想，或者说应用于图1中①处的Dropout 层 [35]的部分重计算。Dropout 层的作用是将传入特征图中 $p\%$ 的条目的输出设置为零（"丢弃" 这些输出），然后将剩余的输出按比例缩放，缩放因子为$1/(1-p)$。这使得网络对前一层的任何输出不太敏感，从而使网络更加健壮。 Tempo将子层重新计算定义为一种技术，其中仅需要重新计算某些特征图，以实现给定层输出可能产生的向后传递。我们观察到，如果我们在层存储多个输出的情况下仔细解构层，则可以采用更好的重新计算策略。该观察结果可以直接应用于 Dropout 层。在 Dropout 的计算中，会产生掩码（记录在 Dropout的实现中设置为零的条目）和输出。如果使用基于层的检查点实现，层被设置了检查点，则会导致掩码和输出在向后传递中重新计算，从而需要更高的开销。然而，如果仅存储掩码只会减少重新计算（包括内存传输）时间，而掩码本身只有布尔值这一事实使我们能够保留重新计算的大部分内存优势。这种方式可以有效节省关键 $O(S^2)$ 注意力部分（图1中的③）输出所需的存储空间，以代替简单掩码乘法的成本。该技术如图3 所示。

3.5 其余优化

PyTorch 在实现 softmax 函数时使用了一种内存效率较低的方法，这种方法在反向传播（backward pass）时保留了函数的输入和输出 [34]。然而实际上，只需要输出就足够了。这种优化在 Huggingface 库中的某些模型中也已经实现过。为了进一步减少激活内存压力，这种优化方法也被用在了注意力机制（attention mechanism）的实现中。

4 复现细节

4.1 与已有开源代码对比

论文提供了源代码，但是在复现过程中发现其代码运行环境陈旧，存在大量兼容性问题，因此本报告在本地复现了源代码, 并使用Bert预训练语言模型进行中文文本分类实验，分析复现源代码的有效性。

4.2 实验设置

设备：我在一台拥有 Nvidia GeForce RTX 4090 Laptop GPU 的笔记本电脑上进行了实验，显存大小为16G。

中文文本分类实验：我使用Bert模型进行中文文本分类实验，采用经典的中文数据集THUCNews中的20万条新闻标题进行增量学习，比较了Tempo方法和Baseline方法在显存占用、模型收敛等方面的性能差异。 THUCNews数据集的新闻标题涵盖了丰富的主题，包括政治、经济、科技、娱乐等多个领域，文本长度在20到30之间。一共10个类别，每类2万条。数据以字为单位输入模型。

指标：我关注的第一个指标是Tempo与Baseline相比的总内存占用量,比较相同参数的给定常用批量大小下 PyTorch 使用的总内存。我们使用的第二个指标是模型的性能，比如模型的准确率和损失变化，比较损失曲线并显示由于我们的有损优化而带来的变化。以确保没有显著的精度偏差。

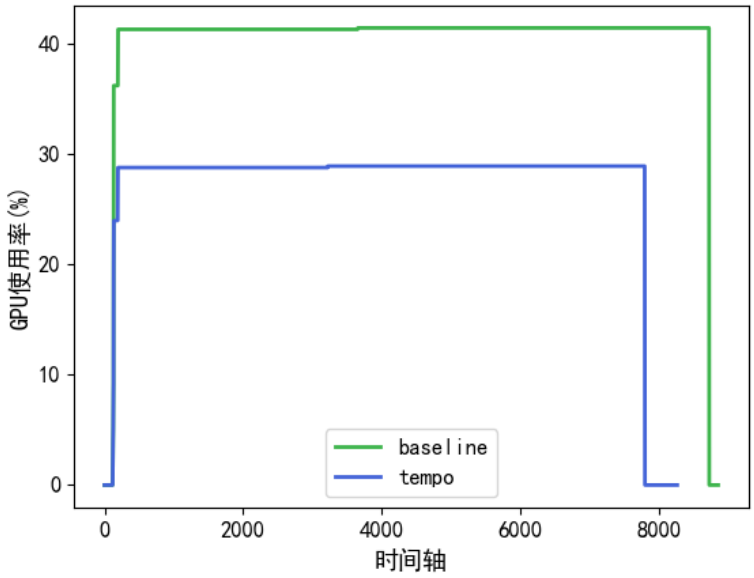
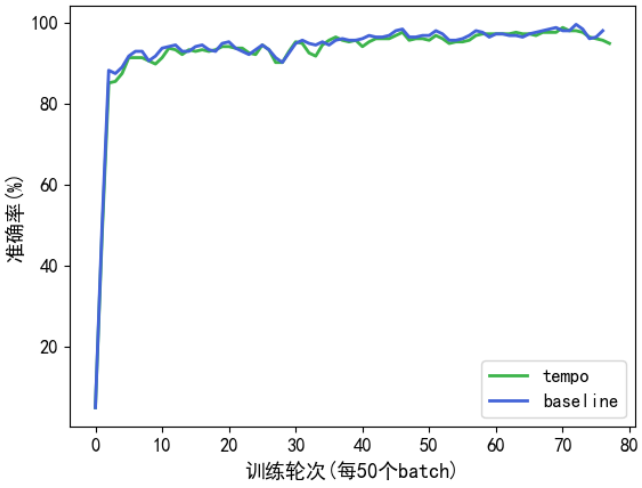
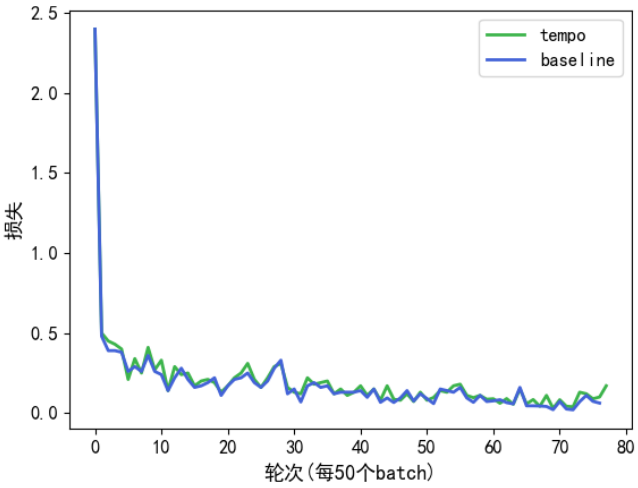


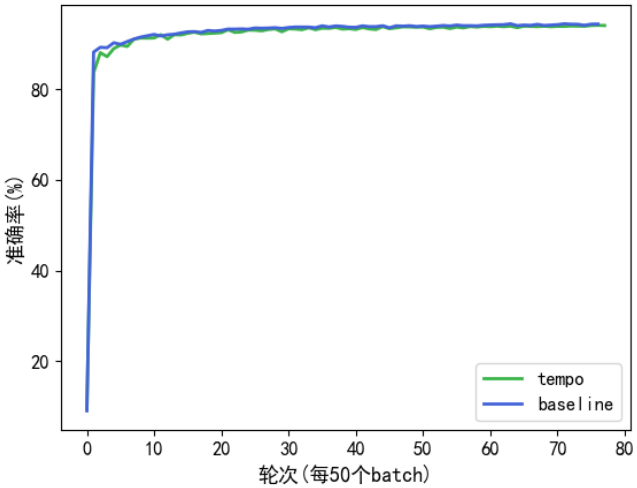
图 4. Tempo与Baseline相比的总内存占用量



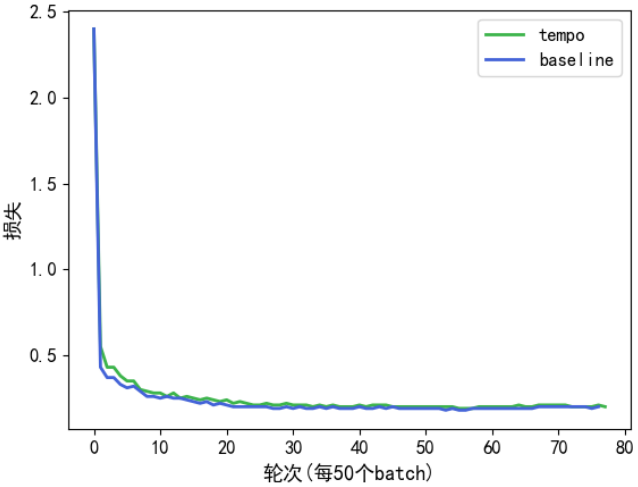
(a) 训练集准确率



(b) 训练集损失



(c) 测试集准确率



(d) 测试集损失

图 5. Bert-Tempo和Bert-Baseline在THUCNews数据集上的表现

5 实验结果分析

5.1 显存占比

我们将Baseline和Tempo技术用于Bert模型，在相同的checkpoint参数基础上，并用相同的THUNews数据集进行模型的训练。在模型训练开始的同时，我们运行一个python脚本用于GPU监控。我们设置监控的采样间隔为100ms，得到的系统GPU占比曲线如图4。在模型刚开始训练的时候，两个模型训练的准确率都明显上升，但Tempo优化后的模型曲线率先到达最大值。Tempo的GPU占比相比于Baseline降低了12.6%，节省了约为2.06G显存。这表明了复现的Tempo 技术对比于Baseline方法有着明显的显存优化。

5.2 模型性能影响

上一节的实验探讨了Tempo对于显存占比的优化，这一节的实验针对该优化方案是否会对模型的性能产生影响。我在训练的过程中记录了每50个batch时,模型在训练集、测试集上准确率和损失的变化趋势。如图5所示，可以发现两种方法模型训练的损失和准确率曲线趋势基本一致，这验证了Tempo技术能够在不损失模型精度的情况下减少显存占用，减少的存储空间可以用于提高模型训练的批量大小，进而增大训练吞吐量。

6 总结与展望

本报告对Tempo的三个优化层In-place GELU、In-place LayerNorm以及Sub-Layer Dropout Recomputation进行了复现，并比较了复现结果与Baseline技术在Bert中文文本分类这一下游任务上的差异，取得了一定的优化效果，验证了复现的有效性。但是复现过程中依然存在一些不足，我只验证了论文提供的加速方法对于某一特定下游任务的加速效果，但是该论文实现的方法与其他的加速方法是相互正交的，这意味着我们可以进一步探究哪一些加速方法可以与Tempo相互结合，在具体的任务中能够实现更高的加速比。此外，Tempo论文中所使用的掩码、分段反函数输入输出恢复的优化思想也很值得我们借鉴学习。

参考文献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[4] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[5] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems*, 3:711–732, 2021.

[8] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[9] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

[10] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[11] Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 891–905, 2020.

[12] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[13] Bojian Zheng, Nandita Vijaykumar, and Gennady Pekhimenko. Echo: Compiler-based gpu memory footprint reduction for lstm rnn training. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1089–1102. IEEE, 2020.

[14] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[15] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789. IEEE, 2018.

[16] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pages 1803–1813. PMLR, 2021.

[17] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[18] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization. *arXiv preprint arXiv:2006.09616*, 2020.

[19] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[21] Samuel Rota Bulo, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5639–5647, 2018.

[22] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. Benchmarking and analyzing deep neural network training. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 88–100. IEEE, 2018.

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[24] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[25] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[26] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[27] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[28] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[29] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[30] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pages 41–53, 2018.

[31] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[32] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[33] 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.

[34] 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.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.