高技术通讯 2024 年 第 34 卷 第 10 期:1036-1045

doi:10.3772/j.issn.1002-0470.2024.10.003

深度学习程序内存预测方法①

刘 晨② 陆 杰 李 炼③

(中国科学院计算技术研究所处理器芯片全国重点实验室 北京 100190)

(中国科学院大学 北京 100049)

摘 要 深度学习程序在广泛领域取得了巨大成功,然而其内部错误可能导致严重的资源浪费,甚至引发灾难性故障。本文分析了导致程序在实际运行中出现任务执行失败的典型缺陷及其关键影响因素,提出基于静态分析与自注意力机制网络的深度学习程序内存预测方法,在程序内存估计任务上达到平均 8.38% 的相对预测误差,可以有效预防内存溢出问题、协助合理优化硬件资源配置。

关键词 深度学习;静态分析;内存预测

在数据资源不断丰富和计算资源日益增强的共同驱动以及一系列开源学习框架[1-3]的支撑下,深度神经网络的研究与应用呈现出蓬勃发展态势。时至今日,相较于传统机器学习技术,以神经网络为核心的深度学习程序在提供解决方案上展示出了非凡能力,在图像分类[4]、语音识别[5]、语言翻译[6]、自动驾驶[7]和药物分子分析[8]等方面都取得了突破性进展。

为了加快深度学习模型执行速度、简化开发流程并优化部署效果,越来越多开发者倾向于利用图形处理器强大的并行计算能力进行模型优化,同时诸如亚马逊 SageMaker、阿里云机器学习平台 PAI (platform for AI)以及微软 Azure 机器学习平台等深度学习云计算服务也在全球范围内大规模普及,提供了从模型训练到部署的解决方案,有力推动了深度学习在工业界及学术界的广泛应用与深入研究。

然而,在深度神经网络从理论研究转向实际应用过程中,其鲁棒性、安全性和完整性问题日益凸显,逐渐成为学术界与工业界关注的焦点。如同其他软件系统一样,深度学习程序也可能会包含多种缺陷,这些缺陷可能会导致任务失败,或是结果、性能不如预期,浪费了大量处理器算力、存储及网络等共享资源。在云平台上执行深度学习程序之前,开发者必须对所消耗的图形处理单元内存进行预估。过度的内存分配可能导致资源的浪费,而内存不足则会导致程序执行失败,开发人员必须调整资源配置重新申请计算资源,或修改模型结构以适应硬件限制,这一过程无疑会降低开发效率。对深度学习程序错误根源进行深入探究,不仅有助于开发者明确应重点防范的错误类型,还有助于推进面向深度学习程序的错误检测工具的研发,辅助开发者在早期阶段发现潜在缺陷,从而显著提高研发过程效率。

目前已经有大量研究致力于检测深度学习程序缺陷,旨在帮助开发者提高深度学习程序开发效率,加强深度神经网络健壮性。Zhang 等人[9]收集了技术问答社区 StackOverflow 和开源项目托管平台 Github 上大量公开深度学习程序,分析了这些程序的缺陷,并探讨了检测缺陷的挑战。Zhang 等人[10]针对微软旗下深度学习平台上 4 960 个执行失败的深度学习程序进行了分析,对任务失败原因进行了归纳,并强调现有调试手段对于深度学习程序缺陷定位不够有效,需要针对性更强、更高效的缺陷检测

① 国家自然科学基金(62132020)资助项目。

② 男,1995 年生,博士生;研究方向:程序分析;E-mail:liuchen17z@ict.ac.cn。

③ 通信作者,E-mail:lilian@ict.ac.cn。

(收稿日期:2024-02-02)

与定位工具来协助开发者,以降低开发者试错成本与开发不确定性。

针对上述问题,本研究深入分析了导致深度学习任务失败的典型原因,并提出针对深度学习程序设计的自动化内存消耗估计工具。该工具结合自注意力机制网络与静态分析技术,无需对深度学习框架进行深入详尽解析,通过自动化提取和分析深度学习应用程序中的网络信息,能够对峰值内存使用进行预估,辅助开发者在配置硬件资源时做出更为精准的决策,确保所申请硬件资源足以支撑模型的执行,防止内存溢出和过度分配显存资源,促进资源利用最优化。

本文主要贡献有以下 3 点。

(1)本文对工业环境中深度学习程序典型缺陷特性进行了深入剖析,系统性地探究了导致深度学习任务执行失败关键因素,为理解及改进实际应用中神经网络性能提供了理论依据和实证分析。

(2)将自注意力机制网络应用于深度学习程序的内存消耗估计,避免了对深度学习框架的繁琐分析过程,提高了分析流程的自动化水平,能够准确预测神经网络在图形处理器上的峰值内存需求。

(3)基于上述理论研究与方法设计并实现了峰值内存预测工具,实验结果显示,该工具在内存预测上的平均相对误差仅为 8.38%。

1　背景介绍

在深度学习模型构建和执行过程中,张量(tensor)作为一种基本数据结构扮演着至关重要的角色。一个张量在程序中表现为一个多维度数组,形状信息包括了维度数量以及各个维度具体大小。而算子(operator)则是构成深度学习程序核心计算逻辑的基本运算单元,负责执行一系列数学运算与函数操作,连接起一个个张量,推进从输入到输出计算流程。由众多张量与算子组成的模型通过深度学习框架表达为计算图的形式,再交由后续加速库、驱动等组件进行具体执行。

在深度学习程序执行过程中,开发者通常倾向于利用图形处理器进行大规模并行计算以提升计算效率,这意味着需要将运算所需张量数据从主内存迁移至图形处理器内存(显存)中,从而实现在图形处理器上的加速运算。此过程涉及多个关键环节,包括但不限于深度学习框架在初步构建计算图时的优化、驱动程序所提供的内存管理策略以及编译器精心设计的算子级别的优化等。

内存溢出缺陷即在深度学习程序执行过程中,所需要峰值图形处理器内存大小超过已有硬件允许内存范围,从而导致程序失败。伴随着深度学习发展,大量研究与优化工作[11-15]尝试从多元视角对深度学习程序效率进行提升。其中有许多工作不可避免涉及对内存管理直接或是间接影响,例如改变计算图计算顺序以适应并行计算需求、优化数据复用以及减少内存访问等。这些优化措施使得深度学习程序内存情况呈现出动态复杂性,开发者难以仅依赖于对原始程序代码的理解,对整个计算过程所需内存进行直观预测。为了减少内存读写、提高计算并行度,研究者们尝试对计算图计算流程进行整合,如通过先验规则[11]、循环分析[12]、张量表达式融合[13]等方法,打破算子边界,将多个算子在计算时进行融合,提升指令级并行性和数据复用效率。

有大量程序分析工作[16-21]尝试对 C、C ++ 、Java 程序进行内存消耗估计且取得了显著成效。由于深度学习程序构建模型与执行模型高度分离的特性,开发者很难从高级语言层面直观洞察到程序经过多个组件复杂隐蔽的优化与分配调度策略后具体执行的流程,也难以将传统软件上的经验和技术直接迁移到深度学习程序分析上来。Gao 等人[22]提出了一种创新型方法,该方法依赖于手工编写算子内存申请规则以及对硬件内存管理机制进行模拟,对程序运行后生成的模型文件进行内存分析。然而采用手工编写规则方式来预测和量化内存消耗存在显著局限性,准确判断算子在历经多重组件和多种类型优化步骤后的内存行为逻辑是一项极具挑战性的任务,预测精度高度依赖于研究人员对软硬件整体细节的掌握程度,并不实际适用于现今复杂且迭代快速的深度学习开发与生产环境。

2　实证研究

本研究选取了某集团云平台上内部生产团队的

故障深度学习程序任务作为研究对象样本，在一个月的时间窗口内采集到了总计 12 289 个失败任务实例。对于每一个失败任务实例，通过与相关开发团队进行联系，获取了严格去敏处理后的源代码及执行日志等关键信息。由于所涉及输入数据具有高度敏感性和机密性，依照相关法规和伦理准则，未对任何原始输入数据进行获取。

图 1 展示了所研究程序代码规模分布（未统计代码中所引入第三方库代码），可以观察到整体程序规模相对较小，平均行数为 407 行（剔除注释部分），最大程序包含了 2 355 行代码。

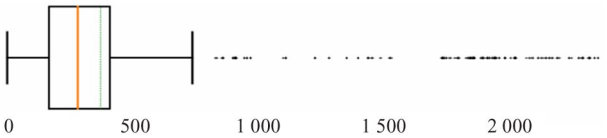

图 1　程序代码规模分布（不含空行及注释行）

研究首先基于执行日志中抛出的错误信息进行特征提取与分类，将抛出相同错误信息程序整合到同一组别中，最终划分出了 968 个失败分组。为了确保所得结果的可靠性，本研究对分组内程序进行了 630 次抽样复核检查，最终确认同一分组内的应用程序确实包含相同失败的原因。

本研究当中对任务失败原因的分析与分类涉及了人工审查，可能融入了一定程度主观性。为了确保结论客观性与准确性，本研究采取了双重验证策略，所有任务失败原因分析均由两位作者进行细致独立审查并交叉验证，还通过与原始开发者沟通交流进行进一步核实确认。此外，尽管该研究涉及任务涵盖了图像、语音、语言处理等多个深度学习领域，但由于全部来源于同一平台，其中部分结论不一定完全适用于其他平台。因此本研究核心焦点主要在于剖析深度学习程序本身引发的错误，而不对与平台本身高度相关的问题（如与执行环境相关）进行深入探究。

在所收集的总计 12 289 个失效任务样本中，有 1 612 例被判断为与环境因素相关，这类案例通常伴随着诸如远程访问失败等错误提示信息。另外有 586 例因输入数据损坏而导致任务执行失败。鉴于本研究重点关注深度学习程序自身内在原因导致的任务失败问题，故对上述由于外部环境因素或输入数据故障所引发的任务失败问题不做进一步探究。剩余的 10 091 个对象样本基于失败原因被划分为两大类别：一类为程序语言层面上的传统错误，另一类则为深度学习程序特有错误。图 2 以柱状图形式直观呈现了这 10 091 个对象样本的失败原因分布情况，其中黑色柱代表了深度学习程序特有错误，灰色柱则表示程序语言层面的错误。

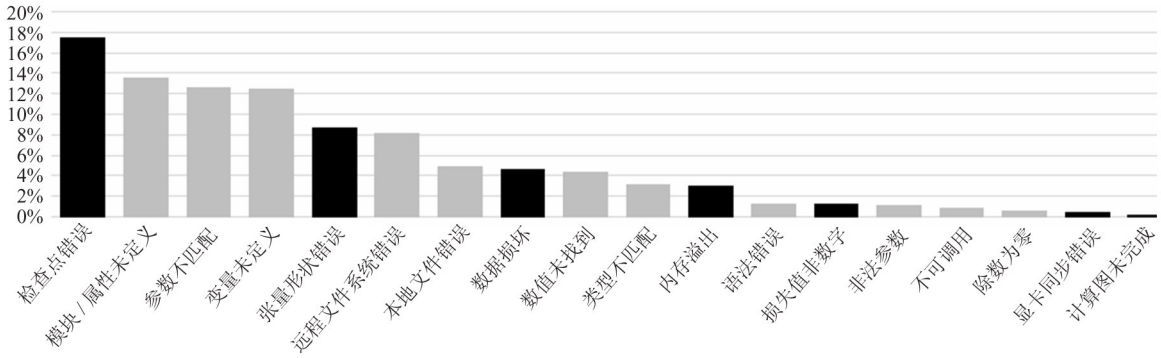

图 2　深度学习程序中的错误类型分布情况

研究结果显示，任务中有 63.69% 可以归因于程序语言层面的错误，这些错误不仅限于深度学习程序，同样普遍存在于各类通用应用程序开发中。在图 2 所示分类分布中，此类错误中占比最高一类为“模块/属性未定义”，即开发者对尚未定义或不存在模块或属性进行了引用操作，该类错误占所有错误的 13.61%。紧接其后的两类错误为：“参数不匹配”（调用函数时提供参数数量与函数定义所要求数量不一致）占比 12.67%；“变量未定义”（在变量正式声明之前对其进行使用）占比 12.51%。

在深度学习程序特有错误中,占比最高的错误类型为“检查点错误”,占比达 17.49%。开发者通常利用检查点机制将训练好的模型状态进行储存,以便后续恢复训练、进行推理或是进一步优化。然而在模型文件丢失、损坏或是所加载检查点文件与当前网络结构不匹配情况下,就会出现检查点错误。

“张量形状错误”这一类别占据了总体任务的 8.82%,此类错误出现在算子或是框架接口函数接收到具有不兼容形状的张量输入(如张量维度不符合算子计算逻辑定义)时,进而引发网络构建失败。开发者可能因难以完全掌握大量框架调用接口所蕴含复杂语义特性,以及对复杂网络传播过程中转递至特定算子处张量形状的判断发生失误,导致实践中频繁出现该问题。

图 3(a)所示为某任务中开发者在尝试调用 TensorFlow 神经网络框架中“reshape”算子时触发张量形状错误时的日志信息摘要。该算子是在保证张量中元素总数保持不变的前提下调整张量形状,因此在实际应用中极为常见。在本案例中,所涉及算子为整个程序中 29 个“reshape”算子之一,该算子接收了一个包含 1 583 个元素的张量,却被要求将该张量转换为包含 1 820 个元素的新张量,由于算子无法找到一个合适方法进行转换,因此抛出了异常并导致了程序中断。

```
tensorflow … errors_impl.InvalidArgumentError:
Input to reshape is a tensor with 1583 values, but
the requested shape has 1820.
```

(a) 张量形状错误引发的错误日志信息摘要

```
tensorflow …
errors_impl.Resource ExhaustedError:
OOM when allocating tensor with shape
[100000,141,200]
```

(b) 内存溢出错误引发的错误日志信息摘要

图 3　错误任务日志信息摘要

“内存溢出”即深度学习程序执行过程中所需内存资源超出了硬件设备实际限制。图 3(b)所示为开发者构建一个超出硬件内存限制的网络时给出的错误日志信息摘要,案例中程序尝试为一个形状为[100 000,141,200]的张量申请内存,但发现剩余内存不足以支撑该需求,因此在日志中给出了 OOM(out of memory)的提示信息。随着软硬件技术持续迭代进步,研究者们不断追求构建设计更为复杂、参数规模更庞大的深度学习模型以实现更优效果。这一趋势使得深度学习程序在执行过程中更易触及底层硬件设备内存限制,从而导致内存溢出问题频发。

其他深度学习程序特有错误类型还包括“损失值非数字”(在计算损失函数时产生无效或非数值结果)、“显卡同步错误”(由多种潜在软硬件故障共同引起与图形处理器之间的通信中断)以及“图不完整”(在构建和执行计算图过程中出现无效或缺失的结构成分)。

3　深度学习程序内存分析

深度学习程序执行过程主要涉及前向传播与反向传播 2 个核心阶段,在前向传播阶段,输入数据通过神经网络模型进行前向传播,沿着开发人员指定的各种算子进行逐步计算,最终生成预测输出结果。在此过程中,内存资源主要用于存储输入输出数据张量以及模型参数张量。随后进入反向传播阶段,首先计算损失函数关于输出激活的梯度,接着从后往前反向进行各个算子的梯度计算,并进行模型参数更新。在这个阶段,内存资源主要消耗包括反向过程中的各类梯度张量、模型参数张量以及临时变量张量等。

图 4 展示了一个仅包含单个卷积层的简化示例程序在执行过程中所涉及张量情况。图中实线圆角方框表示算子的前向计算部分,虚线圆角方框表示算子的反向计算部分,方框代表所涉及的张量。在前向传播过程中,卷积算子(conv)接收输入张量(inputs),与模型参数张量(W1)相结合执行卷积操作生成输出张量(O1),此计算过程中会产生临时张量(T1)。紧接着,输出张量与标签张量(labels)被一同传入损失函数算子(loss),以计算损失值张量(O2),同时产生额外的临时张量(T2)。在反向传播阶段,计算从损失值张量开始逆向进行,计算运算符的梯度张量(D1),并对模型参数张量进行更新,在此过程中由于不同优化器的选取,可能会产生不

同数量的临时变量(T3、T4)。

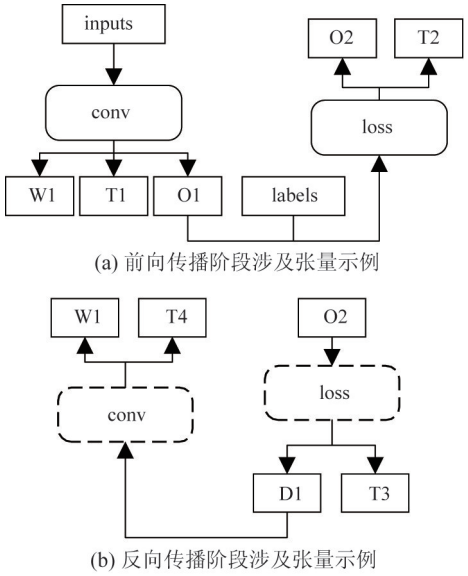

图 4　深度学习程序执行过程中涉及的张量的简化示例

在深度学习程序中,张量内存的消耗因算子的种类和具体框架实现方式而异。单个算子内存占用不仅与其自身的模型参数有关,而且还受从前序算子传递来的张量大小的影响。此外,某些编译优化方案也会对实际占用内存带来影响,如 PyTorch 框架的“reduce-overhead”策略通过增加内存以降低运行开销。图 4 展示极简化例子仅包含一个卷积算子和一个损失函数算子,要将实际开发中整个计算图在执行时可能涉及的内存和优化后的运行逻辑进行精确估计是极具挑战性的任务。因此,开发者在面临溢出问题时往往只能通过不断调整结构进行试错,提高了开发过程中的不确定性,降低了开发效率。

对于第 2 节实证研究中所提问题,当前的静态工具针对程序语言层面错误已展现出相当出色的检测能力,并且此类工具仍在持续更新与优化中。对于“张量形状错误”这一特定问题,文献[17]提出了一种名为 Pythia 的静态分析工具,通过运用数据查询语言 Datalog 对建模后的张量及算子进行分析,从而有效地识别程序是否存在错误。在此基础上,文献[23]提出了基于约束求解的检测方法,能够更为精准地解析和验证张量操作过程中的形状合法性问题,从而显著提升了针对张量形状错误的检测效果。

然而针对“内存溢出”问题,目前仅有一种人工编写计算规则进行模拟推演的方法[20]。直接从深度学习框架源码中自动化提取操作语义将会面临代码量庞大及跨语言的问题,因此手工对算子逐个进行函数摘要的方法反而更为可行。但是这对函数摘要撰写者的要求极高,不仅需要对每个算子的内存读写、数据排布等操作有非常精确的把握,还需要对框架、驱动等可能采取的优化方案有一个整体的认知,了解算子在优化后实际运行的语义。如果函数摘要的撰写者对这些机制不够了解,就可能导致提取的语义与实际运行时的语义不一致,从而产生错误的结果。随着深度学习框架的不断发展和更新,新的算子优化策略可能会频繁地被引入,这就要求函数摘要的撰写者持续投入人力进行更新和完善,以保持与框架最新版本一致。因此,尽管手工提取方法在某些情况下可能是可行的,但其并不是一个长期可持续的解决方案,亟需一个更加自动化的方法来降低开发者的试错成本,解决开发者仍需依赖个人经验知识来预防内存溢出的问题,降低开发过程中的不确定性。

4　深度学习程序内存预测方法

本文提出了一种基于自注意力机制网络的深度学习程序内存预测方法,用于预测程序运行过程中图形处理器上的内存使用峰值。该方法整体工作流程 如图5所示,主要可以分为两大阶段。首先是预

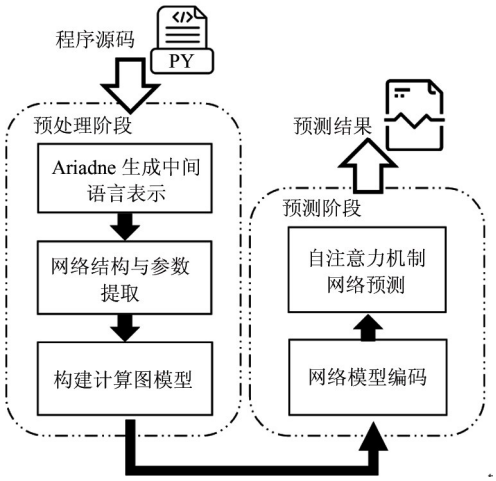

图 5　深度学习程序内存预测方法工作流程

处理阶段,该阶段旨在从程序源代码中有效提取并构建出程序中所要执行的网络结构,针对深度学习程序特征进行语义信息抽取,收集各种影响到程序显存消耗的关键因素。其次为预测阶段,此阶段接收由预处理阶段整理输出特征信息,对其进行编码处理,通过自注意力机制网络进行预测,实现对程序运行时峰值内存需求的准确估计。

4.1　深度学习程序模型计算图提取

在预处理阶段,本研究采用 Ariadne[24] 的前端分析工具对深度学习程序进行解析并将其转化为中间表示形式,基于该中间形式作进一步信息提取。

鉴于深度学习程序在构建网络部分时通常具有相对简单的控制流结构特征,因此针对每个独立程序路径生成对应计算图是可行且在效率上可接受的做法。

对于深度学习网络训练代码而言,其控制流逻辑最终会汇聚至框架内部特定的一些通用训练接口调用,例如在 TensorFlow 框架中体现为对 session.run 方法的调用。基于此特性,可以定位这些有限数量接口调用点,并从接口输出张量出发,沿 use-def 链反向进行程序切片操作。

在反向切片过程中,函数调用会被内联化处理,即将被调用函数返回值视为新节点插入到计算图中,随后继续从新添加返回值节点回溯至当前调用的参数传递源头,通过这种方式将函数调用关系纳入整个依赖图。经过这一系列过程后,所有与训练直接或间接相关的算子、张量及标量等关键元素都将被系统性地整合进计算图之中,从而实现对深度学习程序内存行为相关因素的全面提取。

4.2　基于自注意力机制网络的峰值内存预测

在预测阶段,基于预处理阶段构建得到的计算图信息,通过多头自注意力机制网络[25]捕获和推理深度学习程序执行过程中算子与张量间前后依赖关系,从而进行内存需求量的精确预测。

在未经优化加速情况下,深度学习程序在运行时所需内存主要涉及参与运算的矩阵参数、中间计算临时变量以及反向传播中梯度更新变量,这些变量所需内存空间大小与算子自身参数及关联算子参数配置紧密相关。因此,在程序执行至某一特定算子时,必须依据该算子在上下文中关联算子信息共同推断出当前算子所需内存大小,并依据现有内存分配策略,对当前内存进行适时回收与重新分配。经过一系列优化措施后,计算图内部逻辑可能发生重构,程序中算子执行顺序甚至其结构特征均可能发生调整,内存回收与分配策略也会随着提升并行效率与释放计算潜力的需求而改变。在此情况下,对内存需求进行精确预测需要运用全局视角,对整体计算图潜在并行行为进行判断,才能确保复杂环境下的准确估计。

自注意力机制提供了并行处理序列数据中所有元素的能力,能在全局视域下对任意 2 个位置元素间相互依赖关系进行评估与量化。基于这种比较分析,模型能够动态地针对每个元素分配相应加权考量,这一特性对于全面捕获和整合输入序列整体上下文信息至关重要,对于理解和精确预测复杂且不断变化的内存需求具有决定性意义。

在应用自注意力机制网络之前,需要考虑如何将计算图转化为网络能够有效处理的向量序列形式。首先,利用独热编码(one-hot encoding)对算子类型进行编码构建,形成向量:

$$\boldsymbol{O} = [o_1 \quad o_2 \quad o_3 \quad \cdots] \tag{1}$$

接着是每个特定算子所关联关键参数信息(如卷积算子中的填充(padding)、步长(stride)、卷积核(kernel)尺寸、数据类型、优化器类型、编译优化方法等),这些参数对内存的行为有着直接影响,因此将这些参数视为与算子种类同等重要的特征元素,用向量体现为

$$\boldsymbol{Arg} = [a_1 \quad a_2 \quad a_3 \quad \cdots] \tag{2}$$

通过将上述 2 类向量进行拼接,得到融合了算子类型信息以及参数信息的混合特征向量:

$$\boldsymbol{E} = [o_1 \quad o_2 \quad o_3 \quad \cdots \quad a_1 \quad a_2 \quad a_3 \quad \cdots] \tag{3}$$

需要注意的是,该方法舍弃了计算图中边连接关系信息,可能导致在某些网络估算结果上出现一定的精度损失。

在单个自注意力模块中,输入序列首先通过线性变换映射为查询(Query)、键(Key)和值(Value)3 个特征,数学表达如下:

$$\boldsymbol{Q} = \boldsymbol{E} \times \boldsymbol{W}^Q, \boldsymbol{K} = \boldsymbol{E} \times \boldsymbol{W}^K, \boldsymbol{V} = \boldsymbol{E} \times \boldsymbol{W}^V \tag{4}$$

其中，$\boldsymbol{W}^Q \in R^{d_{\text{model}} \times d_k}$、$\boldsymbol{W}^K \in R^{d_{\text{model}} \times d_k}$、$\boldsymbol{W}^V \in R^{d_{\text{model}} \times d_k}$ 分别代表 3 个仿射变换矩阵，d_{model} 代表向量维度。

接着计算查询 3 个矩阵与所有键的点积相似度并进行缩放，经过函数归一化后得到注意力权重分布。最终利用这个权重分布对所有值进行加权求和以生成输出向量，其公式表述如下：

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}\right) \cdot \boldsymbol{V} \quad (5)$$

其中，$\text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)$ 即注意力权重矩阵，对于第 n 个注意力头，其更新后向量可以写为

$$\boldsymbol{Attention}_h = \boldsymbol{Attention}(Q_h, K_h, V_h) \quad (6)$$

假设共有 H 个并行注意力头，将多个注意力头的输出进行拼接，并通过另一个可训练权重矩阵进行转换，表示如下：

$$\boldsymbol{M} = Concat(Attention_1, \cdots, Attention_H) \cdot \boldsymbol{W}^O \quad (7)$$

其中 $\boldsymbol{W}^O \in R^{H \times d_{\text{model}} \times d_k}$ 是用于输出转换的矩阵。最后的输出结果再通过全连接前馈网络处理后生成关于整个网络计算图的峰值内存回归预测数值。

5　实　验

5.1　实验设计及数据来源

为了确保训练与测试过程中能够获得充分且多样化的数据，实验使用了一种随机生成深度学习神经网络程序算法，旨在系统化构建具有代表性的数据集。为了反映实际应用中深度学习模型的复杂性和多样性，实验中考虑了不同网络数据类型（包括 float32 与 float64）、不同网络优化器（包括 SGD 和 Adam）以及 PyTorch 框架的“reduce-overhead”编译优化方法（该方法在优化运行效率的情况下可能会增加内存的使用），确保所生成数据集能够反映多种内存使用特征，从而测试方法的泛用性。

该数据生成方法整体流程如图 6 所示。每次随机化生成一个合法的网络结构与参数配置，并设定一个初始较小批量大小参数值，基于此生成初始合法程序实例，在程序执行阶段中记录其峰值内存占用情况，根据内存结果增加批量大小并进行下一轮记录，由此构成初始组数据。

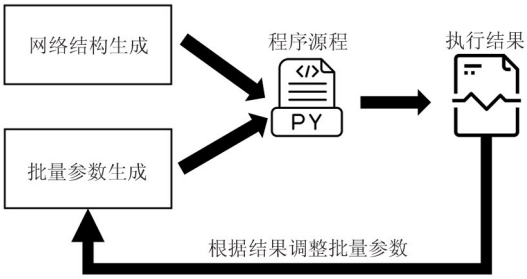

图 6　数据生成流程

为了保证程序样本数量的均衡，进一步对初始组得到的程序按照不同维度上的数量分布进行不同比例随机抽样，例如针对实验中的不同数据类型，初始组数据中 float32 类型的网络占比远高于 float64 类型的网络占比，在经过随机筛选后的两者占比分别为 53.3% 与 46.7%，实现了数据上的均衡。实验最终得到了 97 242 个随机生成目标网络实例及其对应峰值内存数据，并按比例随机划分出训练集、验证集与测试集。

本实验采用的关键评估指标为平均相对误差（mean relative errors，MRE），以此来量化内存预测性能，其数学表达式定义如下：

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|} \quad (8)$$

其中，y_i 表示第 i 个随机生成的神经网络实际被观测到的峰值内存占用大小，$\hat{y}_i$ 表示工具预测出对应网络的内存峰值大小。平均相对误差值越低，即预测结果与真实值之间差异度越小，表明工具具有较高预测准确性。

由于目前尚未有全自动化进行分析的内存预测工具，除了与文献[22]中的工具对比之外，实验中额外训练了一个由 6 层全连接层构成的网络作为基准对比。

5.2　实验结果

图 7 展示了本工具与全连接层网络在测试集上的相对误差分布情况，实验数据表明整体平均相对误差值为 8.38%，而全连接层网络的平均相对误差值为 81.13%。尽管全连接层网络在部分数据上保持了较好的性能，平均相对误差的数据小于100%

的数据占比为 68.50%,且在该部分数据上的平均相对误差为 33.71%,但是整体性能上显著低于本工具。

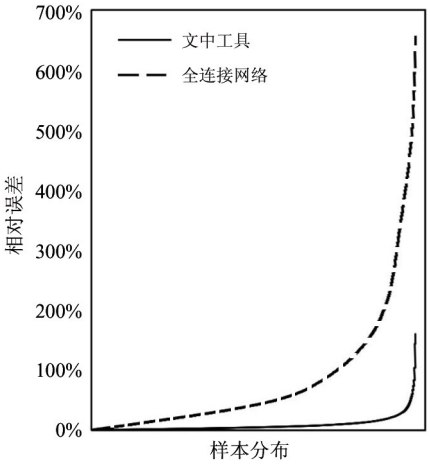

图 7　本工具与全连接网络在测试集样本上的相对误差分布

实验中对 2 个模型进行了多次训练,应用独立样本 t-检验进行统计分析,结果表明 2 个模型的准确率存在显著差异,表明观察到的性能很可能不是由随机变化引起的。尽管在少数案例中存在一定程度的异常估计现象,但对于大部分程序而言,该工具展现出了在大多数场景下的可靠性与有效性,对峰值内存有着较高的预测精确度,能够有效地为开发者提供硬件资源规划参考。在运行时间开销上,本文引入的自注意力网络在每千条数据的预测上平均耗时 0.023 46 s(预测所用硬件型号为 NVIDIA A100),能够同时承担多个网络的内存预测任务。

表 1 展示了工具在不同的数据类型、优化器以及编译优化方法开启与否情景下的预测表现,其中 D32 与 D64 分别代表数据类型为 float32 与 float64,SGD 与 Adam 代表 2 种优化器,选用的编译优化方法为PyTorch框架的“reduce-overhead”。在训练数据均衡的情况下,工具的平均相对误差范围介于 6.75% ~9.61%之间,表明工具在样本包含多种场景组合的情况下能够有效学习并适应不同场景的规则,从而提供精准预测,进一步证实了本工具在处理复杂的深度学习网络预测任务中的适应性和准确性。

表 1　针对多配置下的预测表现情况(%)

是否开启编译优化	D32 + SGD	D32 + Adam	D64 + SGD	D64 + Adam
不开启编译优化	9.40	8.83	8.67	9.61
开启编译优化	7.01	6.75	7.51	9.35

表 2 展示了工具与手工撰写规则实现的工具[22]在网络 VGG16 和 ResNet50 上对比的结果,在 ResNet50 模型上,基于网络自动学习出的规则推理效果优于手工编写规则的效果,而在 VGG16 模型上,两者表现相当。

表 2　不同模型上工具的预测平均相对误差对比(%)

模型名称	本文工具	对比工具
VGG16	8.2	9.8
ResNet50	9.5	17.4

对误差值最大的前 10% 异常预测结果进行深入分析后,发现本文工具在被测网络大小、网络数据类型、优化器类型以及是否开启编译优化上的结果均未体现出显著的特征。这表明为了进一步提高预测的准确性,可能需要进一步优化特征的选择、提取和转换过程,以更有效地捕捉数据的关键信息。此外,对网络结构和超参数的进一步优化也可能有助于获得更稳健的性能。

6　结　论

本文针对深度学习程序失败原因进行了深入研究,分析了导致程序在实际运行中出现任务执行失败典型缺陷特点及关键影响因素,为改进和优化深度学习程序可靠性提供了有力的理论依据。在此基础上,本文提出了一个基于自注意力机制网络的峰值内存预测方法,该方法能够精确地预测程序在图形处理器上执行时的峰值内存需求,有效解决了因内存资源管理不当可能导致的问题。实验数据显示,在对神经网络峰值内存需求进行预测的任务中,该工具平均相对误差仅为 8.38%,且在不同情况下表现稳定,对于深度学习在实际应用中的部署和实施具有重要价值。

本文所实现工具目前专注对单个图形处理器设备上的内存进行情况估计,对于多硬件协同场景,通过进一步扩充特征向量信息,本研究所提出的方法同样具备适用性。未来工作中可以从优化特征表达方式和改进网络模型 2 个关键方面进一步提高预测的精确度。

参考文献

[1] ABADI M, BARHAM P, CHEN J, et al. TensorFlow: a system for large-scale machine learning[C]//The 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Savannah, USA: USENIX Association, 2016:265-283.

[2] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: convolutional architecture for fast feature embedding[C]//Proceedings of the 22nd ACM International Conference on Multimedia. Orlando, USA:ACM, 2014:675-678.

[3] PASZKE A, GROSS S, MASSA F, et al. Pytorch: an imperative style, high-performance deep learning library [C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems. Vancouver, Canada: Curran Associates Inc, 2019:8026-8037.

[4] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Bision and Pattern Recognition. Las Vegas, USA: IEEE , 2016:770-778.

[5] DONG L, XU S, XU B. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition [C]//2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Calgary, Canada: IEEE, 2018:5884-5888.

[6] DI GANGI M A, NEGRI M, TURCHI M. Adapting transformer to end-to-end spoken language translation [C]//Proceedings of INTERSPEECH 2019. Graz, Austria: International Speech Communication Association (ISCA), 2019:1133-1137.

[7] OUYANG Z, NIU J, LIU Y, et al. Deep CNN-based real-time traffic light detector for self-driving vehicles[J]. IEEE Transactions on Mobile Computing, 2019:19(2): 300-313.

[8] LEE C, LIU Z, WU L, et al. Maskgan: towards diverse and interactive facial image manipulation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Seattle, USA: IEEE, 2020: 5549-5558.

[9] ZHANG Y, CHEN Y, CHEUNG S C, et al. An empirical study on TensorFlow program bugs[C]// Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. Amsterdam, Netherland: ACM, 2018:129-140.

[10] ZHANG R, XIAO W, ZHANG H, et al. An empirical study on program failures of deep learning jobs [C]// 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). Seoul, Korea: ACM, 2020: 1159-1170.

[11] NIU W, GUAN J, WANG Y, et al. DNNFusion: accelerating deep neural networks execution with advanced operator fusion[C]//Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. Copenhagen, Denmark: ACM, 2021:883-898.

[12] ZHAO J, GAO X, XIA R, et al. Apollo: automatic partition-based operator fusion through layer by layer optimization[C]//Proceedings of Machine Learning and Systems. Santa Clara, USA: IEEE, 2022,4:1-19.

[13] XIA C W, ZHAO J C, SUN Q Q, et al. Optimizing deep learning inference via global analysis and tensor expressions[C]//The ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). San Diego, USA: ACM, 2024.

[14] 吴林阳, 杜伟健, 陈小兵, 等. 一种运算和数据协同优化的深度学习编译框架[J]. 高技术通讯, 2020,30(2):120-125.

[15] 张志超, 王剑, 节隆兵, 等. 面向目标检测的卷积神经网络优化方法[J]. 高技术通讯, 2022,32(3):227-238.

[16] BRAVENBOER M, SMARAGDAKIS Y. Strictly declarative specification of sophisticated points-to analyses [C]//Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. Orlando, USA: ACM, 2009:243-262.

[17] LAGOUVARDOS S, DOLBY J, GRECH N, et al. Static analysis of shape in TensorFlow programs[C]//The 34th European Conference on Object-Oriented Programming. Berlin, Germany: Schloss Dagstuhl-Leibniz-Zentrum für

Informatik, 2020:1-29.

[18] ALBERT E, GENAIM S, GóMEZ-ZAMALLOA M. Parametric inference of memory requirements for garbage collected languages[C]//Proceedings of the 2010 International Symposium on Memory Management. Toronto, Canada: ACM, 2010:121-130.

[19] BLIN A, COURTAUD C, SOPENA J, et al. Understanding the memory consumption of the mibench embedded benchmark[C]//International Conference on Networked Systems. Marrakech, Morocco: Springer, 2016:71-86.

[20] CHU D H, JAFFAR J, MAGHAREH R. Symbolic execution for memory consumption analysis[C]//Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems. Santa Barbara, USA: ACM, 2016:62-71.

[21] HEO K, OH H, YANG H. Resource-aware program analysis via online abstraction coarsening[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal, Canada: IEEE, 2019:94-104.

[22] GAO Y, LIU Y, ZHANG H, et al. Estimating GPU memory consumption of deep learning models[C]//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Virtual: ACM, 2020:1342-1352.

[23] LIU C, LU J, LI G, et al. Detecting TensorFlow program bugs in real-world industrial environment[C]//2021 36th IEEE/ACM International Conference on Automated Software Engineering. Melbourne, Australia: IEEE, 2021:55-66.

[24] DOLBY J, SHINNAR A, ALLAIN A, et al. Ariadne: analysis for machine learning programs[C]//Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. Philadelphia, USA: ACM, 2018:1-10.

[25] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach, USA: Curran Associates Inc., 2017:6000-6010.

Memory estimation for deep learning program

LIU Chen, LU Jie, LI Lian

(State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Deep learning programs have achieved tremendous success in a wide range of fields. However, internal errors in these programs can lead to significant resource wastage and even result in catastrophic failures. This paper analyzes typical defects that cause task execution failures during the practical operation of programs, as well as their key influencing factors. A deep learning program memory prediction method is proposed based on static analysis and a self-attention mechanism network. This method achieves a mean relative error of 8.38% in memory estimation tasks, effectively preventing memory overflow issues and assisting in the rational optimization of hardware resource allocation.

Key words: deep learning, static analysis, memory estimation