计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(1):245−253 [doi: 10.15888/j.cnki.csa.009283]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于输入特征稀疏化的图神经网络训练加速①

马煜昕[1], 许胤龙[1,3], 李　诚[1,3], 钟　锦[2,3]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230026)
[2](合肥师范学院 计算机与人工智能学院, 合肥 230601)
[3](合肥综合性国家科学中心 人工智能研究院, 合肥 230026)
通信作者: 许胤龙, E-mail: ylxu@ustc.edu.cn

摘　要: 图神经网络 (graph neural network, GNN) 是处理图数据的重要方法. 由于计算复杂、图数据容量大, 在大规模图上训练图神经网络依赖于 CPU-GPU 协作和图采样训练方法, 其中图结构和特征数据存储在 CPU 内存中, 而采样得到的子图及其特征则传输至 GPU 进行训练. 然而, 这种方法面临着严重的图特征数据加载瓶颈, 显著降低了端到端训练性能, 且图特征占用过多内存, 严重限制了可训练的图规模. 为了解决这些问题, 我们提出了基于输入特征稀疏化的数据加载方法, 显著减少 CPU 内存占用和跨 PCIe 总线传输的数据量, 大幅缩短数据加载时间, 加速 GNN 的训练, 使其可以充分利用 GPU 计算资源. 针对图特征和 GNN 计算特性, 我们提出了适用于图特征数据的稀疏化方法, 在压缩比和模型准确度之间达到平衡. 我们在 3 个常见 GNN 模型和 3 个不同规模的数据集上进行了实验评估, 包括最大的公开数据集之一 MAG240M. 结果表明, 此方法将特征尺寸减小了一个数量级以上, 并实现 1.6–6.7 倍的端到端训练加速, 而模型准确度的降低不超过 1%. 此外, 在仅使用 4 个 GPU 的情况下, 仅需 40 min 就可以在 MAG240M 上完成 GraphSAGE 模型的训练并达到目标准确度.

关键词: 图神经网络; 数据加载; 稀疏化; 压缩; 特征分析

引用格式: 马煜昕,许胤龙,李诚,钟锦.基于输入特征稀疏化的图神经网络训练加速.计算机系统应用,2024,33(1):245–253. http://www.c-s-a.org.cn/1003-3254/9283.html

Accelerating Graph Neural Network Training with Feature Data Sparsification

MA Yu-Xin[1], XU Yin-Long[1,3], LI Cheng[1,3], ZHONG Jin[2,3]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)
[2](School of Computer and Artificial Intelligence, Hefei Normal University, Hefei 230601, China)
[3](Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230026, China)

Abstract: Graph neural network (GNN) has become an important method for handling graph data. Due to the complexity of calculation and large capacity of graph data, training GNNs on large-scale graphs relies on CPU-GPU cooperation and graph sampling, which stores graph structure and feature data in CPU memory and transfers sampled subgraphs and their features to GPU for training. However, this approach faces a serious bottleneck in graph feature data loading, leading to a significant decrease in end-to-end training performance and severely limiting graph scale that can be trained as graph features take up too much memory. To address these challenges, this study proposes a data loading approach based on input feature sparsification, which significantly reduces CPU memory usage and data transfer across the PCIe bus, significantly shortens data loading time, accelerates GNN training, and enables full utilization of GPU resources. In view of the graph features and GNN computational characteristics, the study proposes a sparsification method suitable for the graph feature data, which achieves a balance between compression ratio and model accuracy. The study also conducts

① 基金项目: 国家自然科学基金 (62141216); 安徽高校协同创新项目 (GXXT-2022-045)
收稿时间: 2023-03-16; 修改时间: 2023-04-28; 采用时间: 2023-05-25; csa 在线出版时间: 2023-11-24
CNKI 网络首发时间: 2023-11-28

experimental evaluations on three common GNN models and three datasets of different sizes, including MAG240M, one of the largest publicly available datasets. The results show that this method reduces the feature size by more than one order of magnitude and achieves 1.6–6.7 times end-to-end training acceleration, while the model accuracy is reduced by less than 1%. In addition, with only four GPUs, the GraphSAGE model can be trained on the MAG240M in just 40 minutes with expected accuracy.

Key words: graph neural network (GNN); data loading; sparsification; compression; feature analysis

1 引言

近年来, 随着深度学习技术的发展, 图神经网络已成为处理图数据的重要方法, 在风控系统、推荐系统和药物研发等领域得到广泛应用[1−3]. 然而, 早期的图神经网络, 如图卷积网络[4], 由于其较大的内存需求难以在大规模图上利用 GPU 进行训练. 直到 GraphSAGE[5] 提出了基于采样的图神经网络训练方法, 该方法在不影响最终训练准确度的前提下, 大幅降低训练内存需求. 因此, 图采样训练方法很快被广泛应用于图神经网络的训练中.

图采样训练方法的训练过程分为采样、数据加载和模型计算 3 部分, 并利用 CPU-GPU 协同计算, 训练过程中, 由于 CPU 和 GPU 之间数据传输量大且频繁, 数据加载开销可占训练总耗时的 80%–90%, GPU 资源利用率和端到端训练速度受到 CPU 和 PCIe 资源的严重限制, 导致训练效率和可处理图规模的低下.

现有研究通常采用 GPU 内部缓存的方法来加速数据加载[6−8], 但由于缓存大小有限且数据访问随机性高, 这种方法在处理大规模图时效果不佳. 另一些工作通过优化 GNN 模型来减少数据加载工作量[9,10], 但代价是准确度下降或更慢的收敛速度.

考虑到上述问题, 本文提出了一种新的适用于大部分 GNN 模型的数据加载流程, 通过将输入特征稀疏化来减少 CPU 和 GPU 之间的数据传输量, 并提高可处理的图规模. 我们发现, 与传统的深度神经网络相比, GNN 对输入特征的误差具有更高的容忍度, 因此, 将输入特征稀疏化可以在几乎不影响模型准确性的情况下减少数据传输量.

本文主要贡献如下: (1) 提出了一种适用于图特征的稀疏化方法, 并将该方法集成到 GNN 训练流程中, 以加速 GNN 训练并提高可处理的图规模, 该方法可以在任何具有聚合步骤的 GNN 模型上使用. (2) 对 3 个 GNN 模型和 3 个数据集进行的评估表明, 该方法可以将需加载的特征数据量降低一个数量级以上, 并提供 1.8–4.2 倍的端到端训练加速, 而精度损失小于 1%. 我们还做到仅用 4 个 GPU 在 40 min 内在目前最大的公开图数据集 MAG240M[11] 上完成 GraphSAGE 的训练.

本文第 2 节介绍背景及相关工作. 第 3 节介绍本文提出的基于稀疏化的特征数据加载流程. 第 4 节介绍数据集、实验环境、实验结果与分析. 第 5 节为总结和展望.

2 背景及相关工作

2.1 图采样训练流程

GNN 通过聚合其他节点的特征来提取图中的信息. 实际生产中图通常非常大, 有大量的节点和边, 以及相关的特征. 例如, 如表 1 所示, MAG240M 数据集由超过 2.4 亿个节点和 17 亿条边组成, 其中每个节点有 768 维特征. 其总大小可达数百 GB, 大大超出了目前最先进的商用 GPU (NVIDIA H100) 的 GPU 显存容量 (至多 80 GB). 因此, 不可能将图结构和特征装入 GPU 显存中进行全图训练. 为了解决有限的 GPU 显存带来的挑战, GraphSAGE、FastGCN[12]和 Cluster-GCN[13] 等流行的 GNN 模型不使用全图训练, 而是利用基于采样的小批量训练.

表 1 数据集统计信息

数据集	节点数	边数	特征维数
Reddit[5]	232 965	114 848 857	602
OGBN-Papers100M[14]	111 059 956	3 231 371 744	128
MAG240M[11]	244 160 499	1 728 364 232	768

常见的图采样训练流程如图 1 所示, 包括采样 (①)、数据加载 (又可细分为特征收集②和数据传输③两步) 和模型计算 (④) 3 个阶段. 根据每一小批量的种子节点, 先在 CPU 上从原图中采样得到其邻域构成的子图 (①), 从 CPU 内存中收集子图中各节点对应的

特征数据 (②), 再经由 PCIe 总线将特征和子图结构传输到 GPU (③), 最后在 GPU 上依据子图进行图神经网络的前向计算和反向传播 (④).

这样的图采样训练流程已经被广泛采用, 并成为主流 GNN 训练框架如 DGL[15]和 PyG[16]支持的原生训练方法.

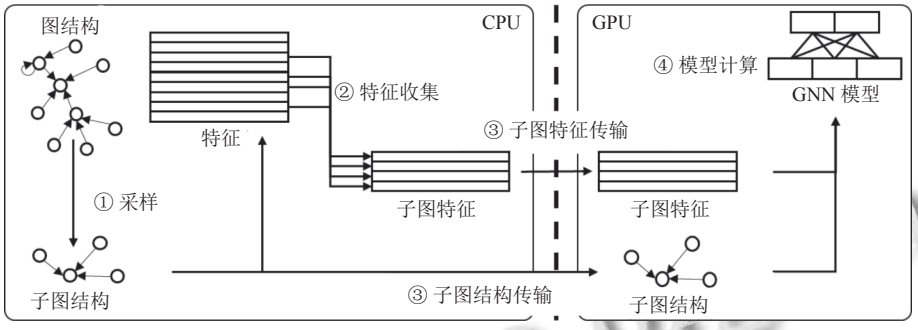

图 1 基于 CPU-GPU 协作的图采样训练流程

2.2 数据加载瓶颈

基于采样的 GNN 训练虽然能够在大规模图上训练, 但存在数据加载缓慢的问题, 数据加载步骤耗时可达 90% 的端到端训练时间. 这是因为在训练过程中, 大量的图数据, 包括采样得到的子图结构信息和对应的特征, 需要通过带宽有限的 PCIe 链路不断地从 CPU 内存传输到 GPU. 这导致了宝贵的 GPU 资源的浪费.

为了对数据加载流程进行深入分析, 我们使用 DGL 训练框架在 Reddit、OGBN-Papers100M 和 MAG240M 数据集上训练 GraphSAGE. 这些数据集的统计数据可以在表 1 中找到. 我们的主要发现如下.

首先, 与结构信息相比, 特征数据在图数据的尺寸中占主导地位. 如图 2 所示, 对于最大的图 MAG240M, 其特征占 698 GB 左右, 占其总尺寸的 98.2%. 在训练过程中, 采样得到的子图比原图更稀疏, 因此特征数据占比更大, 经 PCIe 加载的数据约 99.3% 的尺寸由特征组成. 因此, 无论是内存占用还是 PCIe 带宽占用上, 图数据的主体都是特征数据.

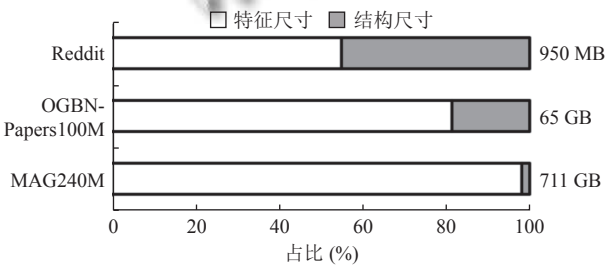

图 2 各图数据集中特征数据占比

其次, 数据加载是限制训练速度的主要瓶颈. 具体来说, 如图 3 所示, 在 3 个训练任务中, 数据加载 (包括特征收集和数据传输) 时间分别占训练时间成本的 93%、90% 和 96%. 具体来说, GNN 的数据加载包括特征收集和数据传输. 特征收集即根据采样得到的子图中的节点序号, 查找并复制其特征向量到一个连续的缓冲区, 而数据传输包括子图结构和特征从内存到 GPU 显存的传输, 但子图结构尺寸通常仅为特征的 1% 左右. 因而, GNN 数据加载开销的主体是特征数据的收集和传输, 与特征尺寸成正相关, 数据加载瓶颈的根本原因在于特征尺寸巨大.

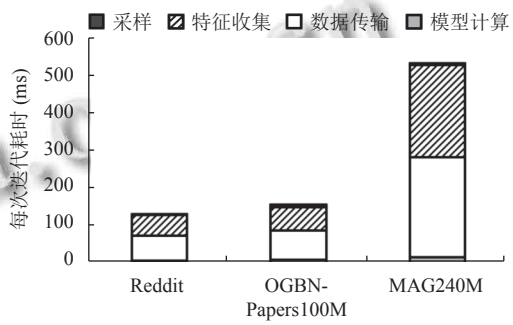

图 3 图神经网络训练中每一轮迭代各步骤的运行时间

此外, 与一般深度神经网络相比, GNN 模型通常参数相对较少, 因此与数据加载相比, 模型的前向和反向传播的计算时间非常小. 这也使得特征数据的加载频繁, 且很难被计算隐藏, 并导致 GPU 利用率低.

以上发现表明图特征数据加载工作量大, 因而在大规模图上加速 GNN 训练的关键是降低特征加载的开销.

2.3 相关工作

在本节我们介绍一些相关工作, 包括利用缓存和修改模型结构来降低训练过程中的数据加载需求.

PaGraph[6]利用空闲的 GPU 显存来缓存出度最高的一部分节点的图特征数据, 从而减少需从 CPU 传输的数据量, 加快数据加载. GNNLab[7]进一步采用预采样的方法来提高缓存命中率, 具体来说, 基于训练集节点和采样方法, 在训练前先计算出每个节点被采样到的概率, 并缓存概率最高的一部分节点特征.

这些方法对特征尺寸不超过数十 GB 的图表现良好, 但难以适应更大规模的图. 其缓存命中率和加速效果会随着图规模的增加而降低. 这是因为有限的 GPU 显存可缓存的特征比例随着图增大而被不断降低, 而图采样给输入节点的选择引入了极大的随机性, 导致小缓存的命中率不高. 在诸如 MAG240M 这样的大图上可缓存的特征比例仅为 0.5%, 而命中率仅有约 10%, 考虑到缓存带来的一些额外开销, 几乎无法带来实质性的加速效果. 因此, 在处理大规模图形时, 缓存的效果往往非常有限.

此外, 全局邻居采样[8]在维持一个小缓存的同时修改采样算法, 以更高的优先级对缓存中的节点进行采样. 这可以有效提高缓存命中率, 但也会带来精度下降的问题, 需要修改前向计算来进行缓解, 导致计算过程变慢, 总体加速有限.

另一些工作通过修改模型结构来减少所需的数据加载量, 如 GNNAutoScale[9]在前向计算时使用历史嵌入而不是在训练的每一次迭代重新计算邻居的嵌入. 由此每次迭代时只需要加载种子节点的直接邻居的特征和历史嵌入, 大大减少了涉及的节点, 从而降低了数据加载量, 减少了 GPU 显存消耗和数据加载时间. 然而, 这需要在主机内存中保存每一层的历史嵌入, 这可能数倍于原始特征大小, 使得面对大规模图时往往因为主机端内存不足而无法使用. 而 VQ-GNN[10]通过一种基于码本更新和近似消息传递的方法来扩展 GNN 模型, 减少了 GPU 显存占用和推理计算成本, 但并未着眼于数据加载过程, 且扩展到大规模图时引入了大量向量量化开销.

稀疏化是一种较为常见的有损压缩方法, 在深度学习领域, 有一些现有工作将稀疏化用于梯度压缩和模型剪枝[17]. 此外, 值得一提的是, 在图领域, 图稀疏化[18]指的是裁剪图中的节点和边, 从而减小图的大小和复杂度. 这些工作的目的和方法均与我们不同, 也没有着眼于输入特征, 和我们的工作是正交的.

上述研究表明, 在面对大规模图时, 由于较高的 GPU 计算能力和相对有限的 CPU 和 PCIe 带宽资源之间的不匹配, 每一次迭代的子图特征加载限制了总体训练速度和 GPU 资源利用率. 而且随着近年 GPU 技术的蓬勃发展[19]和计算方法优化[20], GPU 计算能力的增长快于 CPU 性能和 CPU-GPU 链接带宽的增长速度, 计算能力和数据加载的不匹配正日益严重. 而现有工作往往无法在大规模图上获得显著效果. 我们认为, 减少特征数据尺寸是从根本上加速 GNN 训练中的数据加载和提高 GPU 资源利用率的主要方法.

3 稀疏化的特征数据加载

3.1 稀疏化数据加载方法

本文提出一种新的数据加载方法, 通过稀疏化输入特征, 显著减少数据加载量来加速 GNN 训练并提高可处理的图规模.

图 4 给出了使用稀疏化数据加载方法的系统架构. 在训练开始之前, 先对图数据集的特征进行稀疏化 (⓪). 在每次训练迭代中, 先采样得到子图 (①), 然后收集和传输子图以及对应的稀疏化后的特征 (②, ③), 在 GPU 上解压缩特征, 将其恢复原本的格式 (④) 最后进行前向和反向计算 (⑤).

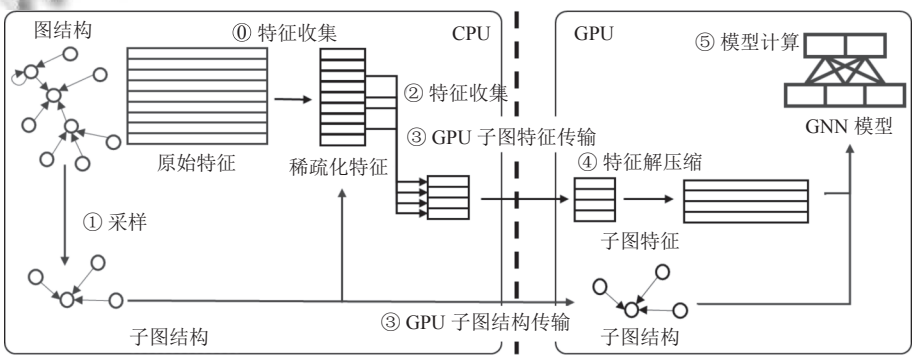

图 4 应用了稀疏化数据加载方法后的图采样训练流程

使用稀疏化数据加载方法相对于原本的训练流程增加了稀疏化 (⓪) 和解压缩 (④) 两个步骤, 并降低了特征收集 (②) 和数据传输 (③) 涉及的数据量. 其中稀疏化是一次性的, 而解压缩在每次迭代过程中反复进行.

解压缩的步骤是为了与 GNN 训练框架的无缝集成所必需的, 它使得最终加载的数据形状及格式保持不变, 从而无需对模型结构进行任何修改, 就可兼容绝大多数常见图神经网络模型. 因此, 不同于修改模型结构的优化方法, 稀疏化数据加载方法可以很容易地应用在现有的 GNN 训练过程中, 仅需在训练开始前和数据加载时分别增加数行代码即可.

我们的方法通过压缩特征数据, 来节约数据加载开销, 提高训练速度和 GPU 资源利用率. 同时节约了 CPU 内存资源, 使得可训练的图规模更大.

3.2 图特征稀疏化

减少特征数据尺寸是解决现有数据加载瓶颈主要方法, 但是通常的无损压缩方法不适应于图特征数据, 这是因为其压缩比 (原数据尺寸与压缩后尺寸的比值) 不固定且较低, 解压缩过程较为复杂, 难以在 GPU 上高效实现. 所幸根据实验观察, 图神经网络对于输入特征的误差容忍性较高, 因此我们选用有损的稀疏化方法.

虽然原本的图特征可能是稠密的, 但考虑到特征向量中包含较为重要信息的维度往往有较大绝对值, 可以使用稀疏化方法裁剪掉部分不重要的维度来减少数据尺寸.

稀疏化的主要思想是对一个稠密的向量只保留少量较为重要的维度, 而去除大部分不重要的维度, 从而减小数据尺寸. 以 GraphSAGE 模型为例, 图神经网络首先将一个节点的多个邻居的特征聚合 (取均值), 然后通过一个全连接层得到中间结果, 反复如此得到多层图神经网络的输出. 在这样的过程中, 具有较大绝对值的特征会对结果造成更大的影响, 而接近零的特征的影响较小. 而且聚合阶段在大量节点之间取均值, 可以将有正有负的输入误差部分抵消, 因此图神经网络相对一般神经网络对输入特征的误差的容忍程度更高, 可以使用更激进的压缩策略.

因此, 我们基于 Top-k 的稀疏化方法提出了适合图特征数据的稀疏化压缩方法, 将特征每 256 维一组, 记录其中数值最大和最小的 k 个值的位置, 不记录这些值本身, 而是使用另外的码本记下每个顺位的值在不同节点间的平均数值. 这样做是由于不同节点间特征数值的分布大致均匀, 不记录值可以节约大量的空间以提高压缩比.

具体来说, 我们提出的稀疏化方法如算法 1.

算法 1. 图特征稀疏化算法

1) 对于每个节点的特征, 按每 256 维为一组进行分组, 按组进行压缩.
2) 对每个节点中的每一组, 取最大和最小的 k 个值, 用 $2k$ 个字节记录其位置.
3) 每一组在不同节点间将最大和最小的 k 个数值分别取均值, 得到 $2k$ 个 32 位浮点数, 记为码本.

由此, 每个节点的每一组的数据大小从 256 个 32 位浮点数减少到 $2k$ 个字节, 即压缩比为 $512/k$. 此外额外需要 $8k$ 个字节的码本, 但这与节点数无关, 相对于特征的尺寸几乎可以忽略, 对压缩比的影响很小.

相应的, 在 GPU 上进行的解压缩步骤如算法 2.

算法 2. 稀疏化特征的解压缩算法

1) 创建原始特征形状的 32 位浮点数格式的数组, 初始值为零.
2) 对每个节点中的每一组特征, 根据 $2k$ 个地址, 将码本中相应顺位的浮点数值填充入相应位置.

算法 2 中的码本在训练开始时预先保存到 GPU 显存中, 得益于 GPU 的高度并行和较快的显存访问, 解压缩步骤的开销较低.

使用稀疏化压缩图特征时, 压缩比并不会随着图规模的扩大而变化. 因此, 与缓存方法不同, 在任意规模的图上使用稀疏化都可以带来显著的加速效果.

3.3 稀疏化对模型的影响

我们也通过实验验证稀疏化对模型中间结果和最终输出的影响. 我们测试并记录了不同的稀疏化设置情况下的输入特征和 GNN 每一层的输出. 以原始特征计算的结果作为基准, 考虑到模型的输出往往会用 Softmax 等方式进行归一化, 不同维度间数值的比例关系较为重要, 因此我们使用与基准的余弦相似度作为主要指标. 我们使用 MAG240M 数据集和 3 层 GraphSAGE 模型进行实验, 使用不同稀疏化设置时, 训练开始时的各项结果和基准的余弦相似度如表 2 所示. 与预期相符, 模型的聚合过程可以有效抵消误差, 每经过一层聚合, 和基准的相似度都有所提高. 较保守的稀疏化策略对输入特征的影响很小, 余弦相似度大于 0.9, 随着压缩比的增大, 相似度也有所下降, 但即便压缩比达到 64, 得益于聚合步骤, 最终输出值的改变也不是特别大, 因此训练时的损失和梯度也较为接近. 我们还观

察到, 随着训练的进行, 模型输出相对于基准的相似度还会有所提高, 最终收敛得到的模型的预测结果和基准几乎一致.

稀疏化压缩不同于降维方法, 而是依然保留了原本的维数, 只是舍弃部分数值, 这相对保留了更多信息. 我们也将稀疏化与常见的降维方法主成分分析 (PCA) 比较, 结果如图 5 所示, 可以看到稀疏化方法可以在更高的压缩比下更好地保持模型准确度. 作为对比, 随着压缩比的提升, PCA 方法的准确度下降迅速.

表 2 不同稀疏化设置下中间结果与基准的余弦相似度

中间结果类型	稀疏化压缩比			
	8	16	32	64
输入特征	0.963	0.868	0.746	0.623
第1层输出	0.981	0.918	0.818	0.704
第2层输出	0.989	0.944	0.859	0.752
第3层输出	0.991	0.949	0.867	0.762

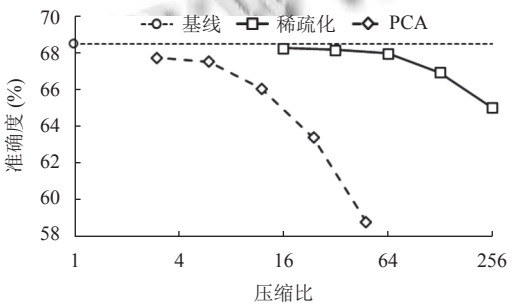

图 5 稀疏化与主成分分析分别在 MAG240M 上训练 GraphSAGE 的准确度比较

3.4 稀疏化开销

使用稀疏化数据加载方法需要在实际训练之前对特征进行稀疏化操作. 稀疏化仅需使用 CPU 进行, 时间上, 稀疏化的计算步骤较为简单, 压缩最大的 MAG-240M 数据集耗时不超过 5 min, 不会对训练耗时造成过大影响. 空间上, 我们对图中节点分批次进行读取和稀疏化, 对于码本的计算则依据随机选取的一小部分节点进行, 因此在稀疏化过程中几乎没有额外的内存空间开销, 峰值内存占用仅略大于压缩后的数据集尺寸. 此外, 图特征的稀疏化是一次性的, 稀疏化的特征可以在不同模型和超参数设置的多次训练中重复使用.

4 实验与分析

4.1 实验设置

我们在装有 4 个 NVIDIA Geforce 1080Ti GPU (11 GB)、一个 16 核 Intel Xeon CPU (2.10 GHz) 和 512 GB 内存的服务器上进行实验. 软件上使用 Ubuntu 20.04、PyTorch 1.12.1 和 DGL 0.9.1. 我们首先主要关注单卡测试的结果, 然后报告多卡测试结果.

我们主要关注节点分类任务, 因为这是在大规模图上较为常见的任务类型, 我们在 Reddit、OGBN-Papers100M 和 MAG240M 这 3 个图数据集上训练了 3 个常见的 GNN 模型, 分别为 GraphSAGE、GAT[21] 和 Cluster-GCN.

表 1 展示了 3 个数据集的统计数据. Reddit 虽然尺寸较小, 但已经被广泛用于准确性的验证. MAG240M 是目前最大的公开图数据集, 它是一个包含文献、作者和机构 3 类节点的异构图, 我们将其转化为无向同构图, 由于其只提供了文献节点的特征, 我们将相邻节点特征的均值作为作者和机构节点的特征, 这使得特征数据尺寸增长到约 698 GB、由于其超出了所用机器的内存容量, 为了完成基线测试, 在训练过程中使用一个包含训练集 3 跳内节点的子图, 包括原图中大约 53% 的节点, 特征尺寸约 368 GB. 使用该子图只会略微加快采样速度, 而不会影响其余步骤的速度和模型准确度, 而且考虑到采样异步进行且不是瓶颈, 其对训练速度的影响可忽略.

我们将原始的 DGL 训练流程作为基线, 用“Full”表示, 调整至常用且较佳的超参数设置, 将稀疏化数据加载的训练流程用“SP”表示. 此外, 我们还探索将 PaGraph 的缓存和稀疏化方法同时使用的效果. 对于稀疏化的训练, 我们使用与基线相同的模型设置和超参数, 并使用令模型精度下降不超过 1% 的最高压缩比.

4.2 准确性验证

表 3 总结了精度结果以及与基线相比的压缩比. 我们得出以下结论: 图特征的稀疏化设置可以非常激进, 即便是表现最差的 OGBN-Papers100M 数据集, 稀疏化特征仍可以在压缩比达到 16 的情况下保证精度损失不超过 1%, 在另两个数据集中的可达到的压缩比则超过 50. 此外, 相同的压缩比 (稀疏化设置) 通常在不同的模型上均有良好的效果, 稀疏化结果可以被有效复用.

我们也评估了稀疏化的特征对非 GNN 模型的影响. 为此, 我们使用多层感知机, 将其结构和参数设置为与测试中使用的 GraphSAGE 模型相同, 它与 Graph-SAGE 的唯一区别在于但没有对邻居节点的聚合操作. 有趣的是, 相同压缩比的稀疏化使得多层感知机的精

度下降了 6%–11%, 这说明了不包含聚合操作的一般神经网络模型不适合使用稀疏化特征, 而与 GNN 结果的比较恰恰说明了 GNN 可以容忍输入误差的主要原因也是其聚合步骤抵消了大部分的误差.

表 3 使用稀疏化数据加载与基线的 GNN 训练压缩比、准确度和单个迭代轮次耗时

模型	方法	Reddit			OGBN-Papers100M			MAG240M		
		压缩比	准确度 (%)	耗时 (s)	压缩比	准确度 (%)	耗时 (s)	压缩比	准确度 (%)	耗时 (s)
GraphSAGE	Full	—	96.3	16.8	—	66.1	188.6	—	68.4	564.2
	SP	50.2	95.9	2.5	16	65.6	49.2	64	68.2	120.9
GAT	Full	—	95.2	18.5	—	66.0	225.2	—	67.9	582.8
	SP	50.2	94.9	4.6	16	65.1	126.7	64	67.2	164.2
Cluster-GCN	Full	—	95.9	1.3	—	62.2	152.8	预处理时内存不足		
	SP	50.2	95.7	0.7	16	61.5	98.1			

4.3 训练加速

我们测试了上述节点分类任务的平均每个迭代轮次的训练耗时, 结果如表 3 所示. 对于 3 个模型, 稀疏化的数据加载方法实现了 1.56–6.72 倍的加速效果. 其中 Cluster-GCN 表现出的加速比略低, 这是因为它的数据加载的工作量没有另两个模型那么大, 但也仍获得了可观的收益.

在最大的数据集 MAG240M 上的结果良好, 达到了平均 4.1 倍的加速, 这主要是因为该数据集特征维数多, 数据加载开销占比高. 而在特征维数较少的 OGBN-Papers100M 上的加速相对较小, 平均为 2.4 倍.

为了公平地比较训练速度, 我们还研究了稀疏化的数据加载对收敛率的影响. 图 6 展示了在 Reddit 上训练 GraphSAGE 时, 测试准确度和迭代批次之间的关系. 可以看出, 使用稀疏化的数据加载方法前后, GNN 模型的收敛速度没有显著区别, 在相同的迭代轮次时稀疏化后的 Loss 略低, 而模型准确度始终相近. 结合上述每一轮次运行时间的加速, 我们可以得到结论, 使用稀疏化的数据加载可以极大加速 GNN 的端到端训练速度.

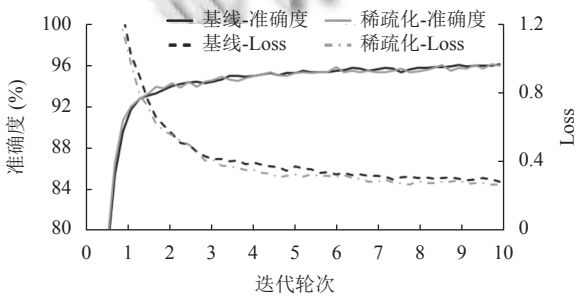

图 6 应用稀疏化前后收敛率比较

训练速度的提升主要来自于特征数据尺寸的大幅降低, 我们以在 MAG240M 上训练 GraphSAGE 为例, 对训练过程各步骤的耗时进行分解以了解加速效果的具体由来, 结果如图 7 所示, 稀疏化的数据加载方法将特征收集和传输成本降低了约 90%, 虽然额外引入了解压缩成本, 但并不显著影响加速效果.

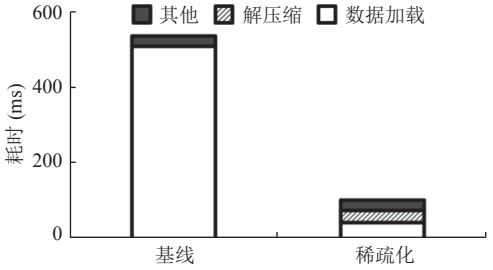

图 7 迭代过程各步骤耗时分解

在训练中的每次迭代, 需加载的子图结构尺寸约 3.8 MB, 而原始 32 位浮点数的特征尺寸高达 1.28 GB, 在经过稀疏化后, 特征尺寸降低至仅 20 MB. 特征收集步骤根据子图节点序号找到对应的特征数据并复制到内存中一块连续的缓冲区. 由于稀疏化后每个节点对应的特征数据尺寸减小, 复制的数据总量降低从而减少了特征收集开销. 从 CPU 到 GPU 的数据传输步骤的速度则受限于 PCIe 带宽, 大大减小的特征尺寸也使得这一步骤的开销变得微不足道.

我们也对不同稀疏化设置下的模型准确度和加速性能进行了研究, 表 4 中展示了在 MAG240M 数据集上训练 GraphSAGE 模型的结果. 可见在压缩比不超过 64 时, 稀疏化对模型准确度的影响很小, 差距在 0.3% 以内, 压缩比达到 128 时准确度才有明显下降. 考虑加速效果方面, 在压缩比较小时, 不同的稀疏化设置对训练耗时的影响较大, 但随着压缩比的增大, 不同稀疏化设置对整体训练速度的影响变小. 在训练过程中, 稀疏化设置影响着数据加载和解压缩步骤的耗时, 而这两

者主要和子图的特征尺寸相关. 在压缩比足够大时, 数据加载不再是主要的瓶颈, 且此时特征尺寸足够小, 数据加载和解压缩步骤中与数据量无关的调用开销也不可忽略, 因此再提升压缩比的效果不明显. 综上所述, 应用稀疏化时, 在保证模型准确度的前提下, 更高的压缩比可以带来更好的加速效果, 但只需一个不太低的压缩比就可以带来接近于最优配置的加速效果, 因此对稀疏化设置的调参较为容易.

表 4　不同稀疏化设置对模型准确度和迭代轮次耗时的影响

模型表现	稀疏化压缩比					
	1	8	16	32	64	128
准确度 (%)	68.4	68.5	68.3	68.2	68.2	67.2
耗时 (s)	564	204	150	130	120	114

最后, 我们在 MAG240M 数据集上对 GraphSAGE 模型进行多卡训练, 并在图 8 中展示训练吞吐量. 如果不进行稀疏化, 增加 GPU 带来的性能提升有限, 由于用于特征收集的 CPU 资源以及用于数据传输的 PCIe 带宽资源的抢占, 无法达到理想的多卡加速比, 4 卡加速比仅为 2.26. 而使用稀疏化后, 由于数据加载工作量的全面降低, 上述资源相对充裕, 4 卡加速比可以达到 3.74, 接近线性加速效果, 从而使得 4 卡时稀疏化数据加载的加速效果进一步达到基线的 8.7 倍. 最后, 利用稀疏化的数据加载方法, 我们成功在 40 min 内使用 4 个 GPU 在目前最大的 MAG240M 数据集上完成 GraphSAGE 的训练并达到理想准确度, 作为比较, 使用同等设定的基线需要训练超过 5.5 h.

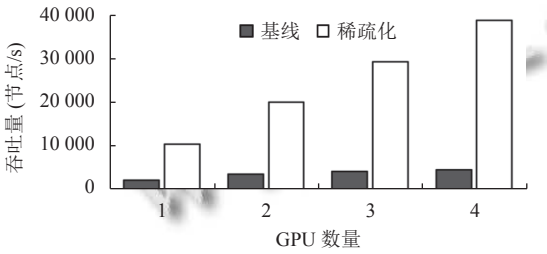

图 8　应用稀疏化前后多 GPU 的吞吐量对比

4.4　可训练的图规模提升

对图神经网络训练来说, 限制其扩展到大规模图的主要因素有两点, 一是内存容量限制, 图数据集往往需要被完全缓存在内存中, 因而无法训练过大的图, 二是成本问题, 由于缓慢的数据加载, 昂贵的 GPU 资源利用率往往不到 20%, 在训练大规模图时需要耗费巨大且不必要的时间和资金成本, 这使得大规模图上 GNN 的训练成本难以接受.

对于内存资源, GNN 训练过程中峰值内存占用略大于图数据集尺寸 (即图结构和图特征尺寸之和). 如图 2 所示, 特征占数据集尺寸的绝大部分, 而稀疏化的特征尺寸下降一个数量级以上, 甚至小于图结构尺寸, 而且如第 3.4 节所述, 稀疏化的过程中几乎没有额外内存开销, 因此特征稀疏化可以大大降低训练时的内存需求. 在 MAG240M 数据集上进行训练时, 基线的峰值内存消耗约 460 GB, 而使用稀疏化的特征时, 内存消耗的峰值降低到大约 80 GB (其中约 60 GB 为图结构). 在我们进行测试使用的拥有 512 GB 内存的机器上, 预计使用特征稀疏化后的单机可以处理的图规模可达 10 亿节点以上, 已经可以满足多数实际用途.

对于资源利用率, 如第 4.3 节所述, 稀疏化数据加载方法可以有效缓解数据加载瓶颈, 提高 GPU 资源利用率, 大大降低训练耗时, 节约时间成本. 同时, 由于云服务商的 GPU 实例通常将内存容量和 GPU 绑定销售 (比如需要 500 GB 以上的内存通常需要租用 8 卡的实例), 而 GNN 训练所需的 GPU 资源相对较少, 稀疏化对内存资源需求的降低也使得在租用云服务商的 GPU 实例时可以更加经济.

总之, 使用稀疏化的数据加载方法可以突破现有限制因素, 以更低的成本快速完成在更大规模图上的 GNN 训练, 有效提高 GNN 训练的可扩展性.

5　结论与展望

在图神经网络训练过程中, 数据加载是主要瓶颈. 我们率先采用稀疏化输入特征的数据加载方法来缓解这一瓶颈, 并提出了适用于图特征的稀疏化方法. 通过实验表明, 利用稀疏化减小特征尺寸, 可以在较好地保持模型准确度的前提下大幅降低内存占用和端到端训练耗时, 同时提升同等硬件可训练的图规模.

虽然我们的工作目前围绕单机训练展开, 但是稀疏化方法在分布式训练情况下也具有潜力. 分布式训练中每台机器分别保存一部分图数据, 数据加载过程中特征数据经由网络的交换是一个潜在的瓶颈, 如何利用稀疏化来加速分布式训练是我们后续的研究方向.

参考文献

1 Wu YJ, Lian DF, Xu YH, *et al*. Graph convolutional networks with Markov random field reasoning for social

spammer detection. Proceedings of the 37th AAAI Conference on Artificial Intelligence. Washington: AAAI Press, 2020: 1054–1061. [doi: 10.1609/aaai.v34i01.5455]

2 Fout A, Byrd J, Shariat B, *et al*. Protein interface prediction using graph convolutional networks. Proceedings of the 31st International Conference on neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6533–6542.

3 Wu SW, Sun F, Zhang WT, *et al*. Graph neural networks in recommender systems: A survey. ACM Computing Surveys, 2022, 55(5): 97. [doi: 10.1145/3535101]

4 Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. Proceedings of the 5th International Conference on Learning Representations. Toulon: OpenReview.net, 2017.

5 Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 1025–1035.

6 Lin ZQ, Li C, Miao YS, *et al*. PaGraph: Scaling GNN training on large graphs via computation-aware caching. Proceedings of the 11th ACM Symposium on Cloud Computing. ACM, 2020. 401–415. [doi: 10.1145/3419111.3421281]

7 Yang JB, Tang DH, Song XN, *et al*. GNNLab: A factored system for sample-based GNN training over GPUs. Proceedings of the 17th European Conference on Computer Systems. Rennes: ACM, 2022. 417–434. [doi: 10.1145/3492321.3519557]

8 Dong JL, Zheng D, Yang LF, *et al*. Global neighbor sampling for mixed CPU-GPU training on giant graphs. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. ACM, 2021. 289–299. [doi: 10.1145/3447548.3467437]

9 Fey M, Lenssen JE, Weichert F, *et al*. GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings. Proceedings of the 38th International Conference on Machine Learning. PMLR, 2021. 3294–3304.

10 Ding MC, Kong KZ, Li JL, *et al*. VQ-GNN: A universal framework to scale up graph neural networks using vector quantization. Proceedings of the 35th Conference on Neural Information Processing Systems. OpenReview.net, 2021. 6733–6746.

11 Hu WH, Fey M, Ren HY, *et al*. OGB-LSC: A large-scale challenge for machine learning on graphs. Proceedings of the 35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track. OpenReview.net, 2021.

12 Chen J, Ma TF, Xiao C. FastGCN: Fast learning with graph convolutional networks via importance sampling. Proceedings of the 6th International Conference on Learning Representations. Vancouver: OpenReview.net, 2018.

13 Chiang WL, Liu XQ, Si S, *et al*. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Anchorage: ACM, 2019. 257–266. [doi: 10.1145/3292500.3330925]

14 Hu WH, Fey M, Zitnik M, *et al*. Open graph benchmark: Datasets for machine learning on graphs. Proceedings of the 34th International Conference on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020, 33. 1855.

15 Wang MJ, Zheng D, Ye ZH, *et al*. Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv:1909.01315, 2019.

16 Fey M, Lenssen JE. Fast graph representation learning with PyTorch geometric. arXiv:1903.02428, 2019.

17 Shi SH, Chu XW, Cheung KC, *et al*. Understanding Top-*k* sparsification in distributed deep learning. Proceedings of the 2019 International Conference on Learning Representations. Addis Ababa: ICLR, 2019.

18 Batson J, Spielman DA, Srivastava N, *et al*. Spectral sparsification of graphs: Theory and algorithms. Communications of the ACM, 2013, 56(8): 87–94. [doi: 10.1145/2492007.2492029]

19 Jouppi NP, Young C, Patil N, *et al*. In-datacenter performance analysis of a tensor processing unit. Proceedings of the 44th ACM/IEEE Annual International Symposium on Computer Architecture. Toronto: IEEE, 2017. 1–12. [doi: 10.1145/3079856.3080246]

20 Abadal S, Jain A, Guirado R, *et al*. Computing graph neural networks: A survey from algorithms to accelerators. ACM Computing Surveys, 2022, 54(9): 191. [doi: 10.1145/3477141]

21 Veličković P, Cucurull G, Casanova A, *et al*. Graph attention networks. Proceedings of the 6th International Conference on Learning Representations. Vancouver: ICLR, 2017.

(校对责编: 孙君艳)