DOI:10.11918/202407052

一种深度神经网络多步延迟参数更新并行优化方法

巨 涛[1],康贺廷[1],刘 帅[2],丁肖健[1],王龙翔[2]

(1.兰州交通大学 电子与信息工程学院,兰州 730070;
2.西安交通大学 计算机科学与技术学院,西安 710049)

摘 要: 为解决深度神经网络(deep neural network,DNN)分布式数据并行训练中因聚合节点梯度进行全局梯度参数更新而导致的高通信开销问题,提出一种DNN多步延迟参数更新并行优化方法。首先,设计了一种自适应多步更新间隔选择策略,通过多次本地迭代,再聚合节点梯度,降低频繁通信造成的额外开销;同时,提出了一种参数修正策略,防止本地模型在多步本地更新后偏离全局模型,从而保证训练精度;其次,在聚合梯度时,将梯度张量切分为子张量,在梯度聚合过程中实现通信与计算的最大化重叠,进一步加速模型训练;最后,在CIFAR-100和ImageNet-mini数据集上,将本文方法与SSGD、Local SGD训练方法进行对比。实验结果表明,本文方法可以在保证模型训练精度的基础上,显著减少因参数更新引入的通信开销,可以实现通信与计算的最大化重叠,充分利用计算资源提升并行训练速度。研究结果可为降低DNN分布式训练过程中的通信开销提供新的方案。

关键词: 深度神经网络;数据并行;通信调度;参数更新;计算与通信重叠

中图分类号: TP391 **文献标志码:** A **文章编号:** 0367-6234(2025)09-0095-14

A multi-step delay parameter update parallel optimization method for deep neural network

JU Tao[1], KANG Heting[1], LIU Shuai[2], DING Xiaojian[1], WANG Longxiang[2]

(1. School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China;
2. School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: To address the high communication overhead caused by global gradient parameter updates at aggregation nodes in distributed data parallel training of deep neural network (DNN), a parallel optimization method of multi-step delay parameter updates for deep neural network is proposed. Firstly, an adaptive multi-step update interval selection strategy was designed. After completing multiple local iterative parameter updates, node gradients are aggregated to update the global model parameters, reducing the excessive communication overhead caused by frequent gradient aggregation. At the same time, to prevent the local model from deviating from the global model after several local updates, a parameter correction strategy is proposed to ensure the accuracy of model training. Secondly, during gradient aggregation, the gradient tensor is split into several sub-tensors. By combining sub-tensor priority scheduling, communication and computation during gradient aggregation are maximally overlapped, further accelerating the model training process. Finally, on the CIFAR-100 and ImageNet-mini datasets, the proposed method is compared with SSGD, Local SGD training methods. Results show that the proposed method can significantly reduce communication overhead due to parameter updating on the basis of ensuring model training accuracy. It can maximize the overlap of communication and computing, and make full use of computing resources to improve the speed of parallel training. The results of this study can provide a new resolution to reduce communication costs in the distributed training process of deep neural network.

Keywords: deep neural network; data parallelism; communication scheduling; parameter updating; computation and communication overlapping

深度神经网络(deep neural network,DNN)已广泛应用于计算机视觉、自然语言处理和语音识别等领域。随着DNN模型复杂度的增加和数据量的激增,训练DNN模型变得越来越耗时,采用多个计算

收稿日期: 2024-07-17;**录用日期:** 2024-10-09;**网络首发日期:** 2025-01-02
网络首发地址: https://link.cnki.net/urlid/23.1235.t.20241231.1630.002
基金项目: 国家自然科学基金(61862037);甘肃省科技计划项目(23CXGA0028)
作者简介: 巨 涛(1980—),男,教授,硕士生导师
通信作者: 巨 涛,jutao@mail.lzjtu.cn

节点分布式训练DNN已成为大型机器学习系统的常态[1-2]。分布式训练方式下,将训练数据样本切分为多份,分别存储在不同的计算节点,每个计算节点根据本地模型参数和部分训练数据迭代地执行本地计算;每次迭代结束后,将不同计算节点计算出的梯度进行聚合,以更新全局模型参数;之后再利用最新的模型参数开始下一轮训练[3]。由于计算节点梯度聚合过程中的密集通信占据了总训练时间的主要部分,计算设备的计算资源得不到充分利用,导致通过多个分布式计算设备进行数据并行训练的加速性能通常远低于线性加速。如何提高通信效率、充分发挥计算设备的计算能力、提高模型训练速度,是分布式机器学习系统必须要解决的关键问题[4-6]。

DNN的分层结构特点,可使计算任务和通信任务并行执行,通信调度在不影响计算结果的前提下,通过改变不同层梯度在通信过程中的传输顺序来重叠计算任务和通信任务,从而加速分布式DNN训练[7]。Zhang等[8]提出了无等待反向传播(wait-free backpropagation,WFBP)方法,该方法通过独立推送和拉取每层的梯度张量,将部分通信开销隐藏于计算开销中,但其仅实现了通信在反向传播过程中的重叠,计算和通信重叠的程度较低。Hashemi等[9]基于WFBP,根据反向传播过程中梯度传输的顺序性,实现了计算和通信的最优重叠。由于是非抢占式的调度方式,完成计算较早的低优先级张量的通信会阻塞完成计算较晚的高优先级张量传输,不能很好地利用带宽。为解决此问题,Jayarajan等[10]通过将梯度张量切分成若干子张量,并根据模型的分层结构赋予每层不同的优先级,再按照优先级方式调度,使得通信和下一次迭代的前向传播也可以并行执行,实现了更细粒度的张量调度方式,但如何设置张量切分的基准尺寸大小是一个超参数,在不同网络环境中需要手动调整。Peng等[11]基于贝叶斯优化理论确定了张量切分所需最优基准尺寸这一超参数,但求得的最优超参数在整个训练过程中保持不变,难以适应动态变化的训练环境。针对该问题,Ma等[12]提出了一种根据训练环境变化动态搜索最优超参数的方法。当训练环境变化时,动态地调整超参数并保持较好的训练性能。但该方法在动态网络环境下自动调整参数需要数百次迭代,参数调整成本较高。Bao等[13]将张量切分为多个子张量,通过实时监控计算和通信的状态,动态调整子张量的调度顺序。该方法可适用于不同的分布式训练环境,但在进行动态调度过程中会引入额外的系统开销,不适用于低带宽或高延迟网络。

DNN分布式训练过程中的参数更新方式主要有同步更新(synchronous stochastic gradient descent,SSGD)[14]和异步更新(asynchronous stochastic gradient descent,ASGD)[15]两种。同步更新指所有计算节点使用一致的模型参数进行训练,确保了分布式算法的收敛性和模型的训练精度,但在该更新方式下,系统性能取决于迭代最慢的计算节点,不同计算节点之间的等待时间开销会导致同步更新训练过程中训练性能降低[16]。异步更新解决了参数同步过程中训练效率较低的问题,但由于迭代训练速度的不同,计算节点之间产生的新旧梯度会导致模型收敛效率低下[17]。为减少梯度同步过程中的通信开销,先在各节点本地累积大批量训练的多次迭代梯度后,再聚合各节点累积的梯度进行整个模型参数的更新,从而大幅度降低通信频次[18]。但该方法相当于线性增大了批量大小,而批量过大会因缺乏梯度噪声而影响模型的泛化性[19]。与之不同,Local SGD[20]允许节点在本地独立更新本地模型参数若干次后,再进行梯度聚合以更新全局模型参数,但其没有对梯度进行累积,而是在每次迭代中使用小批量数据计算梯度进行本地更新,保持了较小的批量,从而可同时兼顾模型的泛化性和较低的通信频次。其中,更新间隔H是一个超参数,会影响模型训练的性能。Moritz等[21]统计了不同更新间隔下模型训练到20%测试准确率时所需的时间,然后从中选择训练时间最短的更新间隔,但会引入额外的统计时间开销。Coquelin等[22]预先设置一个损失阈值,当训练损失超过该阈值后,再进行节点梯度聚合,并清空累积的损失值。但随着训练的进行,损失值越来越小,损失值的累积也越来越慢,更新间隔也将越来越大,不利于训练后期模型参数的微调,进而影响模型的收敛性。Wang等[23]设置了一个初始的通信间隔,再结合学习率和损失函数的变化,动态调整更新间隔,但频繁的调整需要额外的计算资源以实时监控学习率和训练损失,增加了额外的系统资源开销。

上述DNN数据并行优化存在以下问题:1)计算与通信不能较好重叠,计算资源得不到充分利用;2)没有充分考虑子张量频繁通信所引入的额外通信开销,在一定程度上削减了张量切分所带来的性能收益,影响了模型的训练效率;3)尽管多次本地更新有效减少了通信量,但更新间隔的选择较为复杂,影响模型的收敛性。

针对以上问题,本文提出了一种DNN多步延迟参数更新并行优化方法。先在本地执行多次迭代更

新后，再聚合各节点的梯度进行全局参数更新，从而减少频繁通信引入的额外通信开销。同时，为确定不同训练环境下本地更新间隔，设计了一种自适应的多步更新间隔选择策略，以充分利用训练过程中计算和通信资源。在梯度聚合过程中，设计了张量切分和优先级调度策略，实现计算和通信最大重叠，充分利用计算资源，加速梯度聚合过程。

1　多步延迟参数更新整体训练框架

本文提出的 DNN 多步延迟参数更新整体训练框架如图 1 所示。在每轮训练开始时，根据该轮的训练损失和学习率的变化，结合计算与通信时间，求得该轮的更新间隔 H_s；每个计算节点分别读取小批量数据以计算本次迭代的梯度和修正，并根据计算得到的梯度和修正值更新本地模型参数；每经过 H_s 次本地更新后，将每个节点的梯度聚合，进行全局参数更新；在聚合过程中，对层张量进行切分，结合调整后的优先级进行子张量的传输，实现聚合过程中计算和通信的重叠最大化；完成该层所有子张量聚合后，更新该层的模型参数，开始该层下一次迭代的前向计算。

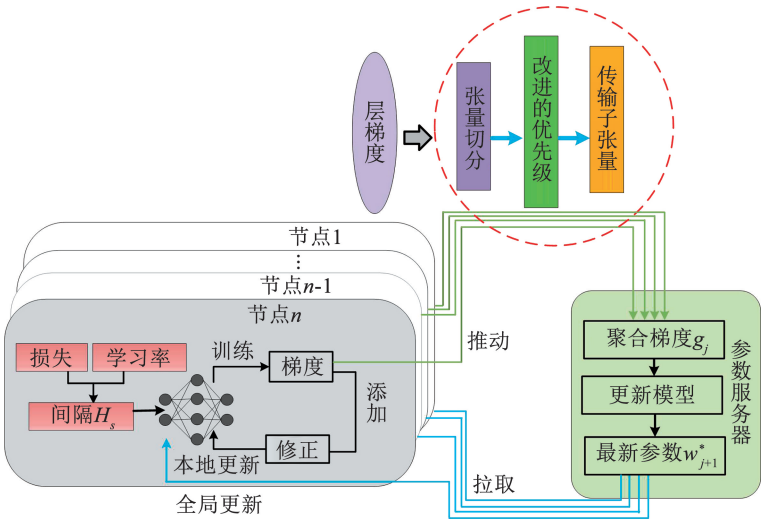


图 1　多步延迟参数更新整体训练框架

Fig. 1　Overall training framework of multi-step delayed parameter update

2　多步延迟参数更新策略

数据并行训练过程中，需聚合所有计算节点的梯度以更新全局参数，但频繁的聚合会引起严重的通信开销。为此，本文设计了一种多步延迟参数更新策略，在保证模型训练精度的同时，加快训练速度。

2.1　自适应间隔选择

更新间隔的选择直接影响模型的收敛性和训练速度，如果更新间隔太小，节点频繁地聚合梯度进行全局参数更新，导致通信时间远大于计算时间，计算资源处于闲置，从而使通信开销成为瓶颈；如果更新间隔太大，训练过程中聚合梯度的次数大幅度减少，虽然降低了通信开销，但各节点的模型参数差异较大，进而影响模型的收敛性。

在模型训练早期阶段，较大的更新间隔可以快速降低训练损失值；在训练后期阶段，较小的更新间隔有助于对模型进行更精细地调节[22]。为了在保证模型训练精度的同时提升模型的训练速度，需要设计一种动态调整更新间隔的策略，使得在训练早期阶段通过大的更新间隔快速收敛模型，而在训练后期阶段通过减少更新间隔提高模型性能。此外，由于训练过程中学习率的衰减和损失的变化，在调整更新间隔时要同时考虑学习率和训练损失的影响。基于以上分析，本文根据每轮训练后的本地节点训练损失，并结合学习率的变化，得出自适应更新间隔调整表达式为

$$H_s = \left\lceil \sqrt{\frac{\alpha_0}{\alpha_s}\frac{F(D_{H_s})}{F(D_{H_0})}} H_0 \right\rceil \tag{1}$$

式中：H_s 为第 s 轮迭代的更新间隔；H_0 为初始更新间隔，是一个需要设置的超参数；α_0 为初始学习率；α_s 为第 s 轮训练的学习率；$F(\cdot)$ 为损失函数；D_{H_0} 为初始部分训练数据子集；D_{H_s} 为第 s 轮训练的训练数据子集。

2.2　参数修正

多次本地更新后，各节点本地模型参数之间会出现发散，影响模型收敛。为了保证模型收敛，本文

提出了一种参数修正策略。当本地模型偏离全局模型时,对本地模型进行惩罚修正,以限制本地模型在执行多步本地更新后偏离全局模型的程度,具体修正过程如图 2(以第 s 轮迭代的更新间隔 $H_s = 3$ 为例)所示。其中,$g_1^{(1)}$、$g_2^{(1)}$、$g_3^{(1)}$、$g_1^{(2)}$、$g_2^{(2)}$、$g_3^{(2)}$ 分别为节点 1、2 在第 1、2、3 次迭代计算的梯度,w_1、$w_1^{(1)}$、$w_1^{(2)}$ 为初始模型参数,$w_2^{(1)}$、$w_3^{(1)}$、$w_2^{(2)}$、$w_3^{(2)}$ 分别为节点 1、2 在完成 1、2 次迭代更新后的本地模型参数,w_4 为第 3 次迭代聚合两个节点的梯度 $g_3^{(1)}$ 和 $g_3^{(2)}$ 更新后的全局模型参数。例如,在第 1 次迭代时,节点 1 利用梯度 $g_1^{(1)}$ 和修正 $\lambda(w_1^{(1)} - w_1)$(λ 为修正系数),将本地模型参数 $w_1^{(1)}$ 更新为 $w_2^{(1)}$;节点 2利用梯度 $g_1^{(2)}$ 和修正 $\lambda(w_1^{(2)} - w_1)$,将本地模型参数 $w_1^{(2)}$ 更新为 $w_2^{(2)}$;在第 3 次迭代时,将节点 1、2 的梯度 $g_3^{(1)}$ 和 $g_3^{(2)}$ 聚合,将全局模型参数更新为 w_4。

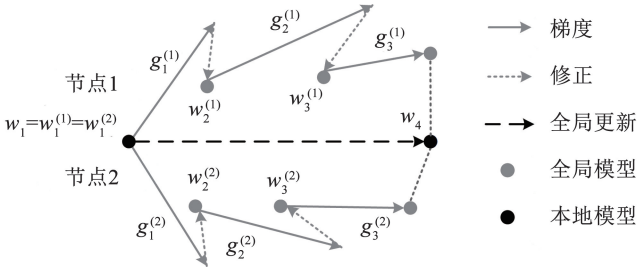


图 2 参数修正示意

Fig. 2 Illustration of parameter correction

2.3 参数更新过程

在每轮迭代开始求得本轮更新间隔 H_s 后,每个节点读取小批量数据计算梯度,并添加修正以更新本地模型,每执行 H_s 次本地更新后,聚合所有节点的梯度,更新全局模型参数。具体参数更新式为

$$w_{j+1}^{(n)} = \begin{cases} w_j - \alpha_j \dfrac{1}{N}\sum_{n=1}^{N} g_j^n, j\%H_s == 0 \\ w_j^n + \alpha_j g_j^n + \lambda c_j^n, \text{其他} \end{cases} \tag{2}$$

式中:w_j^n 为计算节点 n 在第 j 次迭代的模型参数,α_j 为学习率,N 为计算节点的个数,g_j^n 为计算节点 n 在第 j 次迭代计算的梯度,λ 为修正系数,c_j^n 为节点 n 第 j 次迭代计算的修正,w_j 为全局模型参数。

整个参数更新过程主要由两部分组成:

1)本地更新。在每轮迭代开始时,根据本轮训练开始时的训练损失,结合一次迭代的计算时间与通信时间,确定该轮的更新间隔 H_s;然后,读取小批量数据计算得到梯度 g_j^n,并根据节点 n 的本地模型参数 w_j^n 和全局模型参数 w_j 计算得到修正项 c_j^n;最后,节点 n 根据计算所得到的梯度和修正项,将本地模型参数 w_j^n 更新为 w_{j+1}^n。

2)全局更新。在每轮训练过程中,每隔 H_s 次本地更新后,聚合所有节点的梯度,然后根据聚合后的梯度更新全局模型参数,并将该次迭代更新的全局模型参数应用至下一次本地更新中,用于对本地模型参数更新的修正。多步延迟参数更新实现如算法 1所示。

算法 1 多步延迟参数更新

输入:初始模型参数 w,学习率 α_j,修正系数 λ,总迭代次数 T,总训练轮数 R

输出:全局模型参数

```
while j < T do
    计算第 s 轮的更新间隔 H_s
    c_j^n = w_j^n − w_j        //计算修正
    w_{j+1}^{(n)} ← w_j^n + α_j g_j^n + λ c_j^n        //更新本地模型
    if j%H_s == 0 then
        w_{j+1}^{(n)} = w_j − α_j (1/N) Σ_{n=1}^{N} g_j^n        //更新全局模型
    else
        w_{j+1}^{(n)} ← w_j^n + α_j g_j^n + λ c_j^n        //更新本地模型
        w_{j+1}^n ← w_j^n        //计算下一次本地更新的修正
    end if
    j = j + 1
end while
```

3 梯度聚合通信优化分析

通过在本地更新参数 H_s 次后再聚合梯度,减少了由于频繁的梯度聚合引起的过高额外通信开销。但在全局模型参数更新聚合过程中,计算资源需要等待全局参数更新完成后才能开始下一次迭代计算,在整个聚合通信过程中计算资源处于闲置等待状态。图 3 示意了不同训练方法下的参数更新方式。

图 3(a)中假设第 1、2 次迭代在本地执行,在第 3 次迭代时进行梯度聚合,在梯度聚合的通信过程

中，节点计算资源处于闲置状态。为了减少等待时间，最大化利用计算节点的计算资源，本文在梯度聚合通信过程中将层梯度张量切分为若干子张量，并对子张量的传输做了调整和改进，使计算和通信在聚合通信过程中实现重叠。如图 3（b）所示，在第 3 次迭代梯度聚合过程中，使得通信和第 3 次迭代的反向传播和第 4 次迭代的前向传播重叠，从而充分利用计算资源加速梯度聚合过程。

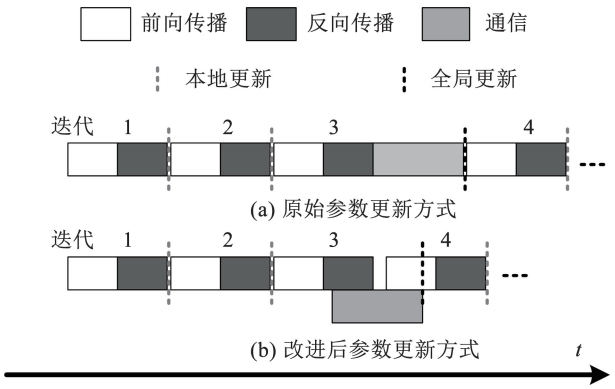


图 3　不同训练方法下的参数更新方式

Fig. 3　Parameter update methods under different training approaches

3.1　计算与通信依赖关系分析

DNN 训练是在大规模数据集上对优化目标进行迭代计算的过程，其中，模型参数在每次迭代完成后更新。每次计算迭代通常包括 3 个步骤：1）输入小批量进行前向传播得到损失；2）利用求得的损失反向传播得到梯度；3）根据所计算的梯度更新模型参数。重复迭代上述步骤，直至模型收敛为止。其中，梯度张量的计算和通信操作形成了一个有向无环图（directed acyclic graph，DAG），现有深度学习框架底层通过执行 DAG 训练 DNN。图 4 为相邻两次迭代之间 DAG 依赖性的示例。其中，F_l、B_l 和 C_l 分别为层 l 的（$1 \leqslant l \leqslant L$）前向传播、反向传播和梯度通信。由于前向传播是从第一层开始直到最后一层结束，而反向传播以相反的方向执行，因此 F_l 依赖于 C_l，B_l 依赖于 B_{l+1}，C_l 依赖于 B_l。DNN 的反向传播确定了靠近输出层的张量被更早计算，但是前向传播的顺序是从输入层到输出层，根据图 4 中计算 – 通信的依赖关系，如果 C_l 完成的更早，则 F_l 将更早的执行。如果将张量切分为若干子张量，并按照优先级调度方式，优先传输靠近输入层的子张量，可以有效地重叠通信和前向传播，使得下次迭代的前向传播尽早开始。

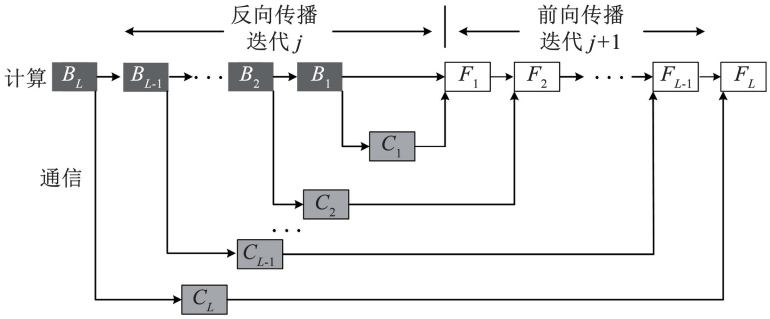


图 4　计算 – 通信依赖有向无环图

Fig. 4　Computation-communication dependency directed acyclic graph

3.2　张量切分

依据基准分区大小将每层的梯度张量切分为多个较小的子张量，并优先传输靠近输入层的子张量，从而可以在计算的同时传输部分子张量，实现计算和通信重叠，达到将部分通信开销隐藏于计算中以有效减少训练时间的目的。设层 l 计算得到的梯度张量为 $\boldsymbol{G}_l$，其大小为 S_l，根据基准分区 S 进行切分，切分后得到 m_l 个子张量 $\{\boldsymbol{G}_{l,1}, \boldsymbol{G}_{l,1}, \cdots, \boldsymbol{G}_{l,m_l}\}$，表达式为

$$m_l = \left\lceil \frac{S_l}{S} \right\rceil \tag{3}$$

除最后一个子张量的大小可能小于 S 外，其余每个子张量的大小均为 S[11]。具体切分过程见图 5。

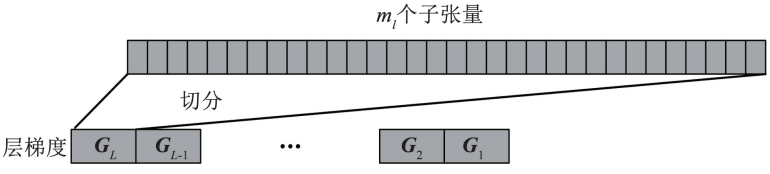


图 5　张量切分示意

Fig. 5　Tensor partitioning diagram

3.3 优先级调度过程分析

在优先级调度过程中,根据层 l 前向传播的顺序,赋予层 l 不同的优先级,越在前的层赋予更高的优先级,从而使较前的层张量优先传输完成,开始下一次迭代的前向传播。通过优先级调度传输切分后的子张量,可以实现通信和下次迭代前向传播的重叠。图 6 以一个 5 层的网络模型为例,分析不同的通信调度方式。图 6(a)为未重叠的通信方式,图 6(b)为基于张量切分的优先调度通信方式。由图 6(b)可以看出,通过优先级调度方式使得计算和通信重叠,将部分通信开销隐藏于计算开销中。但在每次传输一个子张量时会引入一次通信延迟,多次子张量的逐个传输会累积很大的延迟开销,增加总的通信时间。尽管优先级调度可以使得计算和通信重叠,但多个子张量通信引入额外的通信延迟开销可能会削弱计算与通信重叠所带来的收益。为了减少通信延迟开销,进一步加速模型训练过程,如图 6(c)所示,通过对基于张量切分的优先级调度时子张量传输过程中的通信延迟进行优化以减少通信延迟开销,加速模型训练。

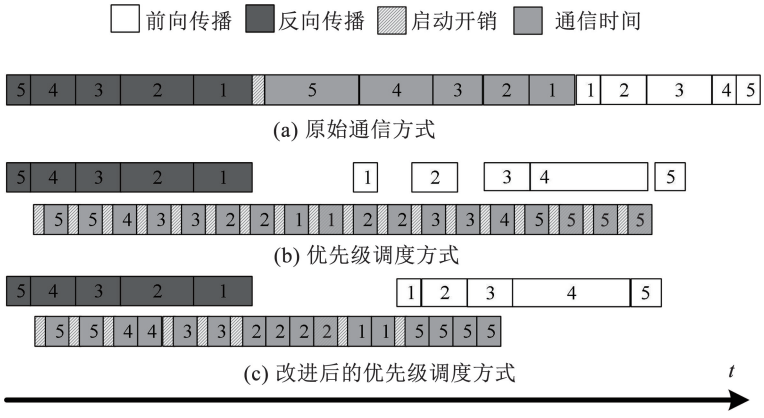


图 6 不同调度方法下的一次迭代时间

Fig. 6 Iteration time under different scheduling methods

4 子张量传输通信优化

4.1 不同阶段计算与通信的重叠

本文将计算和通信的重叠分为两个阶段,具体如图 7 所示。第 1 阶段为梯度通信和第 j 次迭代的反向传播的重叠,第 2 阶段为梯度通信和第 $j+1$ 次迭代的前向传播的重叠。分别定义两个阶段层 l 的优先级为 p_1^l 和 p_2^l,其决定每个阶段中子张量的传输顺序。

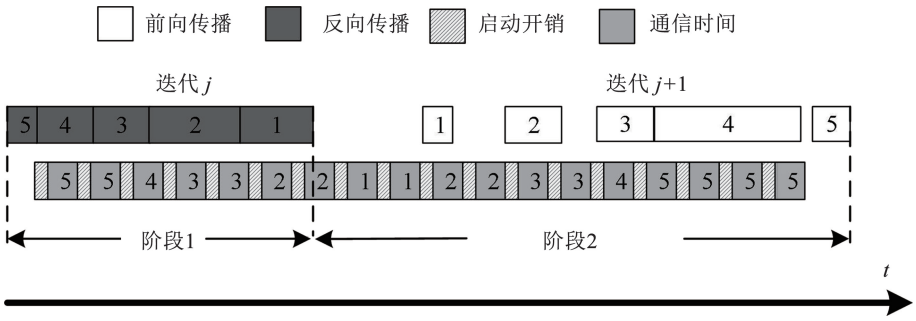


图 7 两个阶段计算与通信重叠示意

Fig. 7 Two-phase illustration of computation and communication overlapping

4.1.1 梯度通信和反向传播重叠阶段

在梯度通信和反向传播重叠阶段,由于反向传播是从后向前逐层进行的,靠后的层会先完成反向传播。此时,尽管这些后层的优先级较低,但由于前面的层还没有完成反向传播,所以这些已完成反向传播层的子张量会被优先传输,随着反向传播的进行,前面优先级更高的层也会陆续完成反向传播,此时具有更高优先级的层梯度被计算出来,优先级最高的子张量也在不断变化。

设第 k 次通信开始的时间戳为 τ_c^k,若要传输层 l 的第 i 个子张量 $\boldsymbol{G}_{l,i}$,则需满足以下条件:1)需要层 l 的张量已经完成计算,即该层已经完成反向传播;2)上一个$(k-1)$通信操作已经完成。

满足上述条件后,在第 k 次通信开始,传输层 l 的第 i 个子张量 $\boldsymbol{G}_{l,i}$ 的开始时间戳 τ_c^k 的公式为

$$\tau_c^k=\begin{cases}\tau_b^l+t_b^l, k=1\\ \max\{\tau_b^l+t_b^l,\tau_c^{k-1}+T_{ar}(\boldsymbol{G}_{l,i})\}, k>1\end{cases} \tag{4}$$

式中：τ_{b}^{l} 为层 l 开始反向传播的时间戳，t_{b}^{l} 为层 l 反向传播经过的时间，τ_{c}^{k-1} 为第 $k-1$ 个通信操作开始的时间戳，$T_{\mathrm{ar}}(\boldsymbol{G}_{l,i})$ 为传输第 i 个子张量 $\boldsymbol{G}_{l,i}$ 需要的时间。

根据式(4)，可以确定第 k 次开始通信的时间戳，其由该层反向传播结束的时间戳和上一个通信操作结束的时间戳决定。在传输层 l 子张量过程中，如果其前一层 $l-1$ 的反向传播完成，则需停止层 l 子张量的传输，优先传输层 $l-1$ 的子张量，层 l 剩余的子张量将会推迟到梯度通信和下次迭代前向传播重叠阶段。

4.1.2　梯度通信和前向传播重叠阶段

上一阶段中，层 l 没有传输完成的子张量将在本阶段进行传输。在梯度通信和前向传播重叠阶段，所有层梯度已经计算完成，因此，当第 k 次通信完成后，可以立即开始下一($k+1$)次的通信。当层 l 所有子张量通信结束后，更新该层参数，要开始下一次迭代的前向传播，需满足以下两个条件：1）其前一层($l-1$)的前向传播已经完成；2）本次迭代层 l 的所有子张量已传输完成。

满足上述条件后，层 l 开始下一次迭代的前向传播开始时间戳 τ_{f}^{l}，可用如下公式表示：

$$\tau_{\mathrm{f}}^{l}=\begin{cases}\tau_{\mathrm{c}}^{k}+T_{\mathrm{ar}}(R_l\times\boldsymbol{G}_{l,i}), & l=1\\ \max\{\tau_{\mathrm{f}}^{l-1}+t_{\mathrm{f}}^{l-1},\tau_{\mathrm{c}}^{k}+T_{\mathrm{ar}}(R_l\times\boldsymbol{G}_{l,i})\}, & l>1\end{cases}\tag{5}$$

式中：τ_{f}^{l} 为层 l 开始前向传播的时间戳，t_{f}^{l-1} 为层 $l-1$前向传播需要的时间，R_l 为层 l 的剩余子张量个数，$T_{\mathrm{ar}}(R_l\times\boldsymbol{G}_{l,i})$ 为层 l 剩余的 R_l 个子张量通信需要的时间。

4.1.3　优化目标

在以上两个阶段通信与计算重叠分析的基础之上，通过最小化子张量传输延迟来最小化一次迭代时间。设最小化一次迭代时间为 t_{iter}，其定义为第 j 次迭代反向传播开始时间戳与第 $j+1$ 次迭代前向传播结束时间戳的差值，则最终梯度聚合优化目标的表达式为

$$\min\ t_{\mathrm{iter}}=\tau_{\mathrm{f}}^{L}+t_{\mathrm{f}}^{L}-\tau_{\mathrm{b}}^{L}\tag{6}$$

式中：τ_{f}^{L} 为最后一层 L 前向传播开始的时间戳，t_{f}^{L} 为最后一层 L 反向传播经过的时间，τ_{b}^{L} 为最后一层 L 反向传播结束的时间戳。

4.2　子张量优先级调度

4.2.1　梯度通信和反向传播阶段优先级调整

在此阶段中，利用优先级调度方式逐个传输子张量会累积巨大的通信延迟开销。为了减少通信开销，本文通过比较一次传输当前层所有子张量和采用优先级调度方式逐个传输子张量确定是否能减少一次迭代时间。如果不能减少迭代时间，则继续按照初始优先级调度方式进行通信；如果能减少迭代时间，则启用一次通信来传输当前层所有子张量，但由于当前层优先级较低，所以需要调整提高该层优先级，以确保该层所有子张量完成传输。图 8 示意了层优先级调整过程，其中带有序号的箭头表示执行过程。先按照初始优先级调度方式给每层张量初始化优先级 p_i，然后从最后一层 L 到第 2 层，通过判断优先传输该层整个张量是否可以减少一次迭代时间 t_{iter} 来依次提高每一层的优先级。假设优先传输层 l 的张量后，迭代时间变为 t_{iter}^{*}，如果 $t_{\mathrm{iter}}^{*}<t_{\mathrm{iter}}$，则提高该层的优先级，否则保持原来的优先级。

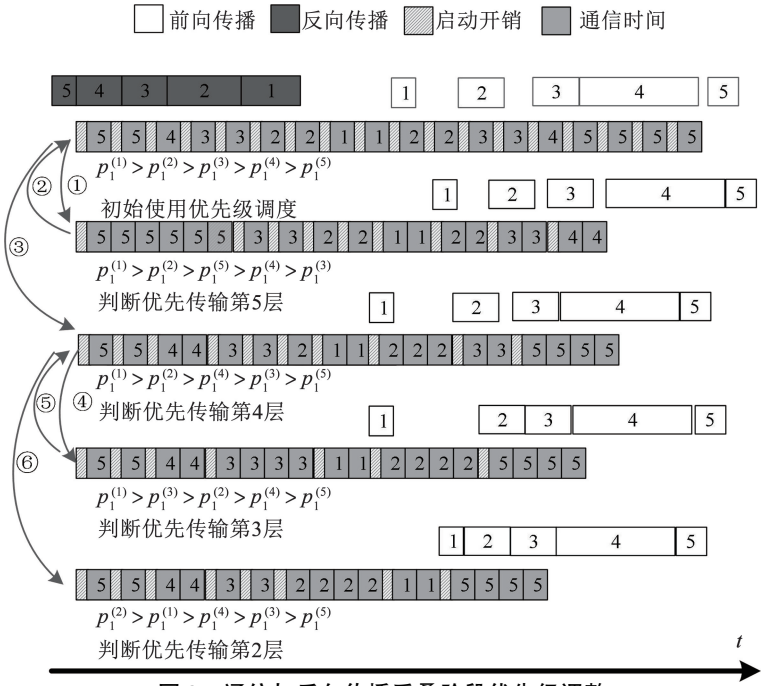


图 8　通信与反向传播重叠阶段优先级调整

Fig. 8　Priority adjustment for overlapping phases of communication and backpropagation

如图 8 所示,当优先传输第 5 层的子张量使迭代时间延长,则采用原始优先级调度方式传输该层张量;在此基础上继续判断优先传输第 4 层的子张量,发现可以减少迭代时间,则优先传输第 4 层的所有子张量;之后继续判断优先传输第 3 层的子张量,发现不能减少迭代时间,则采用原始优先级调度方式传输第 3 层的子张量,持续这一判断过程直至第 2 层为止。

在优先级调整过程中,如果需要提高某层的优先级,需要判断将该层优先级提高到哪一层之前。假设在第 k 次通信时,优先传输第 l 层的张量可以减少迭代时间,同时第 r 层反向传播已经完成,正在进行第 $r-1$ 层的反向传播,则第 l 层张量优先传输完成的时间戳与每层反向传播开始时间戳的关系可以表示为

$$\tau_{\mathrm{b}}^{r}+t_{\mathrm{b}}^{r}<\tau_{\mathrm{c}}^{k}+T_{\mathrm{ar}}(m_l\times \boldsymbol{G}_{l,i})\leqslant\tau_{\mathrm{b}}^{r-1}+t_{\mathrm{b}}^{r-1} \tag{7}$$

式中:τ_{b}^{r} 和 t_{b}^{r} 分别为第 r 层反向传播开始的时间戳和反向传播的时间,$T_{\mathrm{ar}}(m_l\times \boldsymbol{G}_{l,i})$ 为传输层 l 所有张量需要的时间。

根据式(7),如果优先传输层 l 的张量可以减少迭代时间,那么层 l 的优先级应该被提到第 r 层之前。整个优先级调整过程如算法 2 所示。

算法 2 第 1 阶段优先级调整算法

输入:网络层数 L,基准分区大小 S_p,每层张量大小 $S[1,\cdots,L]$

输出:调整后的优先级 p_1^*

```
l = L;
while l ≥ 1 do
    //记录当前通信的层和反向传播结束的层
    //根据式(7)调整优先级
    for r = L → 1 do
        if 式(7) is true then
            p_1^* = Adjust(p_1, l, k)
            break
        end if
    end for
    //更新一次的迭代时间
    if t_iter^* < t_iter then
        t_iter = t_iter^*; p_1 = p_1^*; l = r;
    end if
    l = l - 1;
end while
return p_1^*
```

算法 2 基于优先级调度机制初始化优先级 p_1 和 p_2,并依据两个阶段的优先级计算单次迭代时间。首先,定位当前通信所在层,判断优先传输该层张量是否能够减少迭代时间。如果可以,则调用 Adjust(·)函数提高该层张量的优先级,并一次完成该层所有子张量的通信。在此过程中,可能有多个层完成了反向计算,会导致这些层的张量未传输,需等待所有层反向计算结束后再进行传输。该算法首先遍历每一层,以确定是否需要调整优先级,时间复杂度为 $O(L)$,每次优先级调整的时间复杂度为 $O(L^2)$,因此整体复杂度为 $O(L^3)$。算法在训练之前只需执行一次,在模型整个训练过程中的时间开销可以忽略不计,但算法依赖于稳定的节点计算能力和通信带宽,如果训练环境发生变化,则确定的调度顺序不再适用。另外,该算法在低带宽环境下,加速效果显著;但在高带宽、低延迟环境下,调度顺序对整体训练性能的影响变小,本文调度算法的优势难以充分体现。

4.2.2 梯度通信和前向传播重叠阶段子张量合并

在此阶段中,由于所有层的梯度张量已经完成计算,优先级调度方式传输的是所有层剩余子张量中优先级最高的子张量,但逐个传输剩余的子张量会累积很大的传输延迟开销。为了进一步降低传输开销,本文将层 l 未传输的剩余 R_l 个子张量合并为一个较大的张量,调用一次通信以传输合并后的剩余子张量。由于合并的是同一层中剩余的子张量,所以合并操作并不会改变层 l 的优先级顺序。具体合并过程如算法 3 所示。

算法 3 剩余子张量合并算法

初始化:每层剩余子张量 $R[l]$,层优先级 $p_2[l]$

```
for l = 1 → L do
    if R[l] > 0 then
        for i = 0 → R[l] do
            Merge(G_{l,i})
            t_c[G_{l,i}] = T_ar(G_{l,i})
        end for
    end if
    if l == 1 then
        τ_f[l] = τ_c[k] + T_ar(G_{l,i})
    else
        τ_f[l] = max{τ_f^{l-1} + t_f^{l-1}, τ_c^k + T_ar(G_{l,i})}
    end if
end for
```

算法 3 首先遍历每一层,判断该层是否有未传输的剩余子张量。如果有,则获取该层未传输的剩余子张量个数,然后调用 Merge(·)函数将剩余的子

张量合并为一个较大的张量，并传输合并后的子张量，同时记录传输结束的时间戳；如果该层没有剩余子张量，则继续下一层判断，直到最后一层为止。

5　实验验证

将本文方法与同步更新 SSGD[13]、本地训练 Local SGD[19]方法就训练损失、训练精度、训练加速比进行对比。

5.1　实验设置

5.1.1　软硬件环境

实验采用北京超算中心的 N40 分区，采用 4 个计算节点，每个节点一张 GPU 卡，具体软硬件配置见表 1。

表 1　软硬件环境配置信息

Tab. 1　Software and hardware environment configuration

CPU	GPU	内存/GB	显存/GB	PyTorch	CUDA	Python	NCCL
AMD EPYC 7402 (48C)@2.8 GHz	NVIDIA ® GeForce ® RTX 409	512	24	2.0.1	11.7	3.8	2.18.1

5.1.2　数据集及模型

实验基于 PyTorch 深度学习框架，模型选择具有相同架构但不同深度的 ResNet-50 和 ResNet-20，以及不同架构的 VGG19。数据集采用 CIFAR-100 和 ImageNet-mini。CIFAR-100 数据集由 100 个类别的 6×10^4 张 32×32 像素大小的彩色图片组成，具体包含 5×10^4 张训练图片和 1×10^4 张测试图片。ImageNet-mini 数据集是 ImageNet 数据集的子集，由 100 个不同类别的 13.5×10^4 张图片组成，训练时将每张图片按照 224×224 像素大小进行裁剪，其中训练类包含 13×10^4 张图片，测试类包含 5×10^3 张图片。

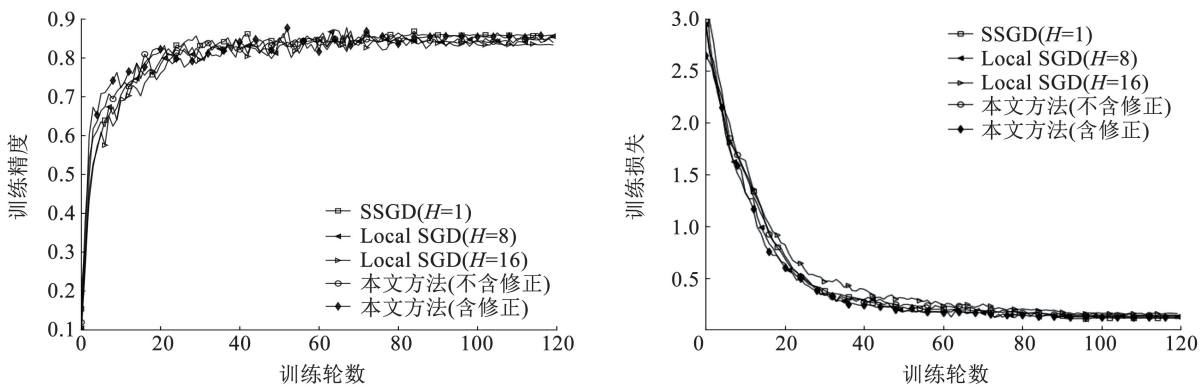
5.1.3　参数设置

采用分布式随机梯度下降优化算法，在 CIFAR-100 数据集上训练 VGG19 和 ResNet-20，在 ImageNet-mini 数据集上训练 ResNet-50。初始学习率设置为 0.1，每经历 30 轮迭代衰减 0.1，修正系数设置为 0.1，批量大小均设置为 64，共计训练 120 个 epoch。

5.2　实验结果对比及分析

5.2.1　训练精度、训练损失对比

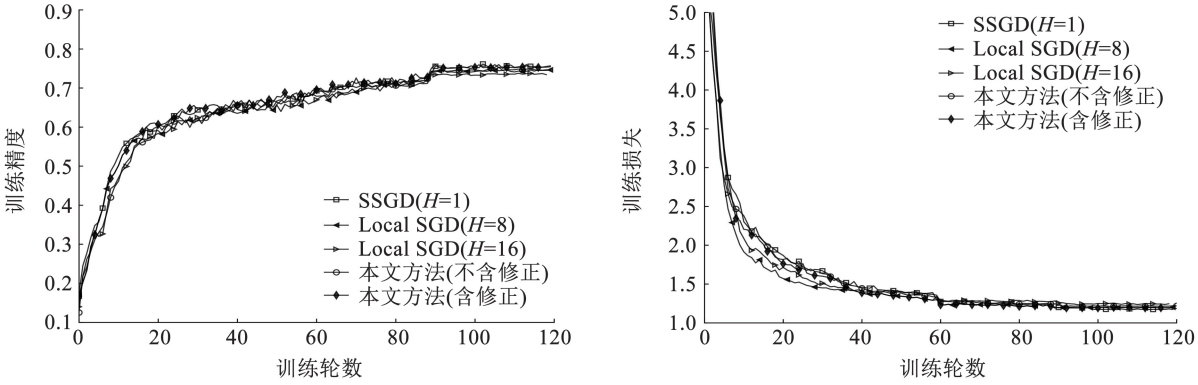
为验证本文方法在训练精度、训练损失方面的整体性能，将本文方法与 SSGD、Local SGD 方法进行了对比。其中对 Local SGD 方法分别取更新间隔为 8 和 16 时的数据进行比较。VGG19、ResNet-50 和 ResNet-20 的训练结果分别如图 9 ~ 11 所示。由图 9(a)、10(a)和 11(a)可以看出，在 3 个模型上，训练初期 3 种方法的训练精度均迅速提升，达到一定训练轮数之后逐渐趋于稳定，最终均能收敛。同样地，由图 9(b)、10(b)和 11(b)可以看出，在训练初期，3 种方法的训练损失均快速下降，到达一定值之后逐渐趋于稳定。



(a) 训练精度　　(b) 训练损失

图 9　VGG19 训练结果

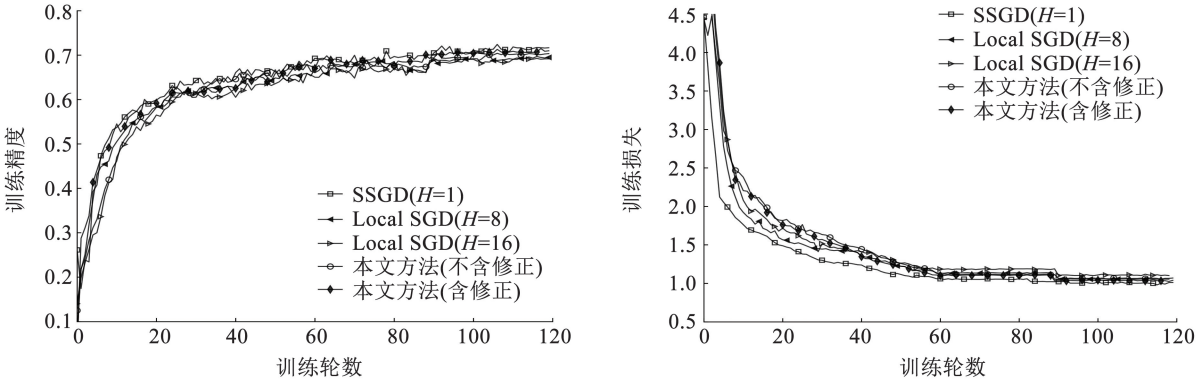
Fig. 9　Training results on VGG19



(a) 训练精度 (b) 训练损失

图 10 ResNet-50 训练结果

Fig. 10 Training results on ResNet-50



(a) 训练精度 (b) 训练损失

图 11 ResNet-20 训练结果

Fig. 11 Training results on ResNet-20

3 个模型的最终训练精度、训练损失及训练耗时比较结果见表 2。由表 2 可以看出:当更新间隔从 1 增加至 8,再增加至 16 时,SSGD 和 Local SGD 方法虽然训练时间在减少,但同时模型精度也在迅速下降;而本文方法可以在不损失训练精度的情况下,训练耗时达到最少。主要原因是本文方法根据训练过程中的实际情况自适应确定每轮训练的更新间隔,及时进行梯度聚合以更新全局模型参数,从而可以在保证训练精度的同时减少训练耗时。由表 2 还可以看出,采用参数修正策略能够保持较高的训练精度和较低的训练损失,原因是参数修正限制了本地模型的发散程度,从而保证了模型的训练精度。

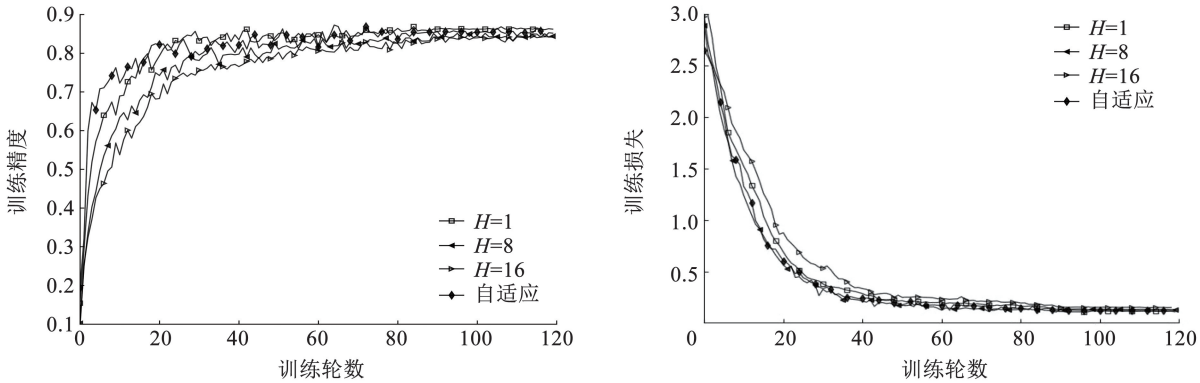
表 2 不同方法下最终训练精度、训练损失及训练耗时对比

Tab. 2 Comparison of final training accuracy, training loss and training time among different methods

模型	训练方法	最终训练精度	最终训练损失	训练耗时/s
VGG19	SSGD($H=1$)	0.868 8	0.114 1	5 231
	Local SGD($H=8$)	0.857 2	0.136 8	1 726
	Local SGD($H=16$)	0.846 1	0.188 4	1 522
	本文方法(不含修正)	0.856 4	0.162 2	1 304
	本文方法(含修正)	0.865 1	0.126 1	1 311
ResNet-50	SSGD($H=1$)	0.753 1	1.172 2	40 219
	Local SGD($H=8$)	0.741 7	1.214 4	17 015
	Local SGD($H=16$)	0.734 2	1.243 7	16 509
	本文方法(不含修正)	0.742 2	1.217 0	15 754
	本文方法(含修正)	0.749 5	1.191 0	15 761
ResNet-20	SSGD($H=1$)	0.712 2	1.021 6	2 237
	Local SGD($H=8$)	0.694 4	1.051 5	1 721
	Local SGD($H=16$)	0.686 3	1.118 3	1 409
	本文方法(不含修正)	0.701 8	1.064 6	1 334
	本文方法(含修正)	0.707 2	1.031 8	1 339

5.2.2　自适应更新间隔有效性分析

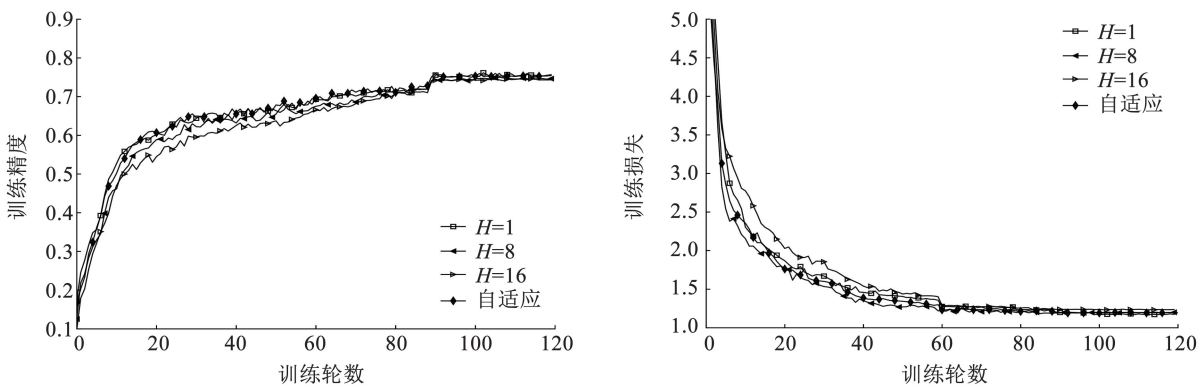
为了进一步验证本文自适应更新间隔策略的有效性，分别对比了本文方法在自适应更新间隔策略和不同固定更新间隔下的训练精度、训练损失及训练耗时。其中，当固定间隔设置大于 16 时，训练精度会严重降低，模型难以收敛，所以将对比的固定更新间隔设置为 8 和 16，并以 SSGD（$H=1$）训练方法为比较基线。具体训练结果如图 12 ~ 14 所示，不同更新间隔下的最终训练精度、训练损失及训练耗时见表 3。



(a) 训练精度　　(b) 训练损失

图 12　VGG19 训练结果

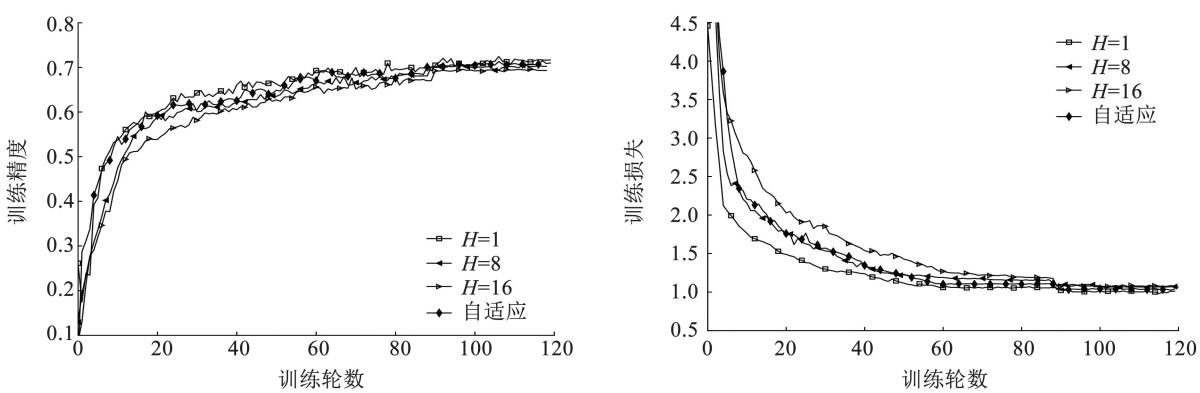
Fig. 12　Training results on VGG19



(a) 训练精度　　(b) 训练损失

图 13　ResNet-50 训练结果

Fig. 13　Training results on ResNet-50



(a) 训练精度　　(b) 训练损失

图 14　ResNet-20 训练结果

Fig. 14　Training results on ResNet-20

表 3 不同更新间隔下本文方法最终训练精度、最终训练损失和训练耗时对比

Tab. 3 Comparison of final training accuracy, final training loss and training time of our method under different update intervals

模型	更新间隔	最终训练精度	最终训练损失	训练耗时/s
VGG19	$H=1$	0.868 8	0.114 1	5 231
	$H=8$	0.860 6	0.133 7	1 674
	$H=16$	0.856 3	0.154 1	1 370
	自适应	0.865 1	0.126 1	1 316
ResNet-50	$H=1$	0.753 1	1.172 2	40 219
	$H=8$	0.745 5	1.198 3	16 581
	$H=16$	0.738 1	1.216 5	16 063
	自适应	0.749 5	1.191 0	15 766
ResNet-20	$H=1$	0.712 2	1.021 6	2 237
	$H=8$	0.694 4	1.051 5	1 721
	$H=16$	0.686 3	1.118 3	1 409
	自适应	0.707 2	1.031 8	1 339

由图 12 ~ 14 可以看出,不同更新间隔下,3 个模型均能收敛,但不同更新间隔对模型最终训练精度影响较大。由表 3 可以看出:在更新间隔为 8 时,3 个模型上一轮训练耗时相比于 SSGD 方法大幅度降低,但训练精度有所下降;更新间隔继续增大至 16 时,一轮训练耗时继续减少,但对精度影响较大,这是因为通信次数减少,导致模型全局参数更新不足,影响了训练精度。此外,3 个模型训练损失的变化趋势表明,更新间隔过大时,会减缓模型的损失收敛速度。本文自适应更新间隔策略在 3 种模型上的最终训练精度均接近于最优精度,并有效缩短了训练时间,说明本文自适应间隔策略可以在有效减少通信开销的同时,保持较高的训练精度。

5.2.3 模型训练耗时及加速比分析

为了验证本文方法在提升模型训练速度方面的有效性,将 SSGD、Local SGD 与本文方法完成一轮训练的计算时间和通信时间进行了对比,结果如图 15、16 所示。

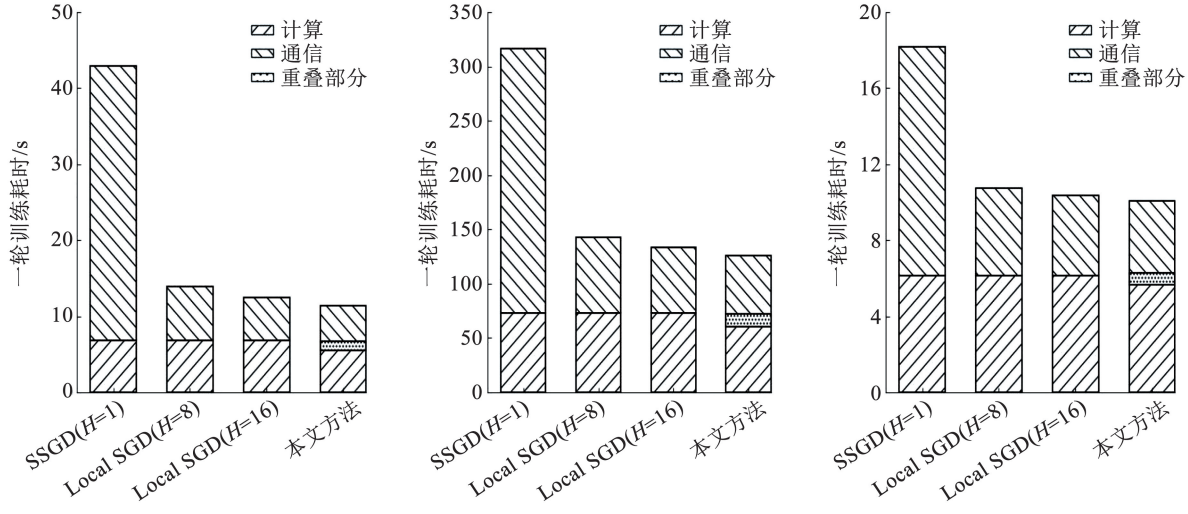


图 15 不同模型的训练时间

Fig. 15 Training time on different models

由图 15 可以看出:SSGD 训练方法在 3 个模型上的一轮训练耗时最长,主要是因为通信占据了大部分时间;Local SGD 和本文方法由于通过多次本地更新,减少了通信时间,进而缩短了整体训练时间。特别地,当 Local SGD 的更新间隔为 8 时,3 个模型的通信时间大幅度降低,间隔继续增大至 16 时,通信时间进一步减少,但相比于间隔为 8 时减少的幅度变小。主要是因为间隔为 8 时,通信次数大幅度降低,通信时间已经小于计算时间,如果继续增大间隔减少通信,会影响模型的收敛,导致整体收益变小。本文方法一轮训练的耗时最少,原因是通过自适应策略调整本地更新间隔,减少了通信次数。同时,在梯度聚合过程中使得计算与通信重叠,进一步缩短了整体的训练时间。

图 16 为不同训练方法相对于 SSGD 方法总训练耗时的加速比。由图 16 可以看出,本文方法在 VGG19、ResNet-50 和 ResNet-20 上的最大加速比分别达到了 3.81、2.51 和 1.49。在 VGG19 上的加速

效果明显优于 ResNet-50 和 ResNet-20，主要是相比于 ResNet-50 和 ResNet-20，VGG19 具有不同的架构和参数规模，其训练过程中的通信开销在总训练时间中占比更大，通过多次本地更新的加速效果更加明显。对于 ResNet-20 模型，当 Local SGD 的更新间隔为 8 和 16 时，性能表现相近。原因是 ResNet-20 模型相对较小，当更新间隔为 8 时，节点主要处于计算阶段，此时进一步增加更新间隔所带来的通信开销的减少不再增加，因此加速比的提升不太明显。

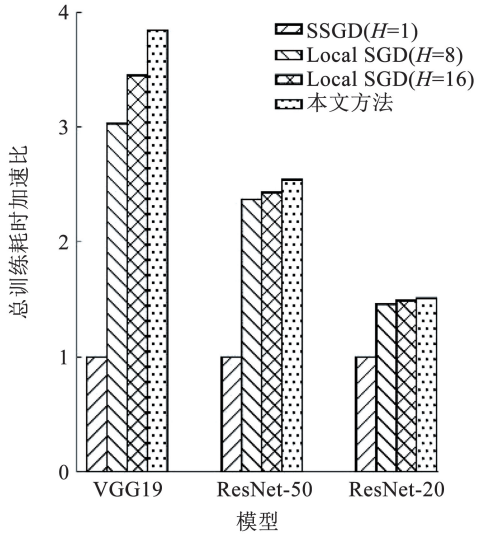


图 16　并行训练加速比

Fig. 16　Parallel training speedup

6　结　论

1）提出了一种多步延迟参数更新优化方法，通过设计自适应多步更新间隔选择策略，有效减少了节点间频繁的通信。同时提出参数修正策略，限制本地模型在执行多步本地更新后偏离全局模型的程度。在梯度聚合过程中，引入改进后的子张量优先级调度机制，实现计算和通信的高度重叠。

2）自适应多步更新间隔策略能根据具体训练环境动态确定本地更新间隔，以适应不同的训练环境，减少密集通信带来的开销。

3）参数修正策略可以在当本地模型偏离全局模型时，调整每个节点因多次本地更新导致节点模型偏离全局模型的程度，保证模型的训练精度。

4）改进后的优先级调度机制，通过优先调度切分后的子张量，实现聚合过程中的计算和通信最大化重叠，进一步加速通信过程，提高计算资源利用率。

5）所提方法在 ResNet-50、ResNet-20 和 VGG19 模型上均取得了良好的训练效果，且可以在保证模型训练精度的同时，有效提高训练速度。

参考文献

[1]朱泓睿，元国军，姚成吉，等. 分布式深度学习训练网络综述[J]. 计算机研究与发展，2021，58(1)：98
ZHU Hongrui，YUAN Guojun，YAO Chengji，et al. Survey on network of distributed deep learning training[J]. Journal of Computer Research and Development，2021，58(1)：98. DOI:10.7544/issn1000－1239.2021.20190881

[2]王恩东，闫瑞栋，郭振华，等. 分布式训练系统及其优化算法综述[J]. 计算机学报，2024，47(1)：1
WANG Endong，YAN Ruidong，GUO Zhenhua，et al. A survey of distributed training system and its optimization algorithms[J]. Chinese Journal of Computers，2024，47(1)：1. DOI:10.11897/SP.J.1016.2024.00001

[3]LIU Ling，ZHOU Pan，SUN Gang，et al. Topologies in distributed machine learning：comprehensive survey，recommendations and future directions[J]. Neurocomputing，2024，567：127009. DOI：10.1016/j.neucom.2023.127009

[4]CAO Xuanyu，BASAR T，DIGGAVI S，et al. Communication-efficient distributed learning：an overview[J]. IEEE Journal on Selected Areas in Communications，2023，41(4)：851. DOI:10.1109/JSAC.2023.3242710

[5]王帅，李丹. 分布式机器学习系统网络性能优化研究进展[J]. 计算机学报，2022，45(7)：1384
WANG Shuai，LI Dan. Research progress on network performance optimization of distributed machine learning system[J]. Chinese Journal of Computers，2022，45(7)：1384. DOI：10.11897/SP.J.1016.2022.01384

[6]徐欣，刘强，王少军. 一种高度并行的卷积神经网络加速器设计方法[J]. 哈尔滨工业大学学报，2020，52(4)：31
XU Xin，LIU Qiang，WANG Shaojun. A highly parallel design method for convolutional neural networks accelerator[J]. Journal of Harbin Institute of Technology，2020，52(4)：31. DOI:10.11918/201812159

[7]巨涛，康贺廷，刘帅，等. 深度神经网络动态分层梯度稀疏化及梯度合并优化方法[J]. 西安交通大学学报，2024，58(9)：105
JU Tao，KANG Heting，LIU Shuai，et al. A dynamic layer-wise gradient sparsity and gradient aggregation optimization method for deep neural networks[J]. Journal of Xi′an Jiaotong University，2024，58(9)：105. DOI：10.7652/xjtuxb202409011

[8]ZHANG Hao，ZHENG Zeyu，XU Shizhen，et al. Poseidon：an efficient communication architecture for distributed deep learning on GPU clusters[C]//Proceedings of the 2017 USENIX Annual Technical Conference. Santa Clara：USENIX Association，2017：181

[9]HASHEMI S H，ADBU J S，CAMPBELL R. TicTac：accelerating distributed deep learning with communication scheduling[EB/OL]. (2018-10-04)[2024-06-17]. https://arxiv.org/abs/1803.03288

[10] JAYARAJAN A, WEI J, GIBSON G, et al. Priority-based parameter propagation for distributed DNN training [EB/OL]. (2019-05-10)[2024-06-17]. https://arxiv.org/abs/1905.03960

[11] PENG Yanghua, ZHU Yibo, CHEN Yangrui, et al. A generic communication scheduler for distributed DNN training acceleration [C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville: ACM, 2019: 16. DOI: 10.1145/3341301.3359642

[12] MA Yiqing, WANG Hao, ZHANG Yiming, et al. AutoByte: automatic configuration for optimal communication scheduling in DNN training [C]//Proceedings of the 41st IEEE Conference on Computer Communications. London: IEEE, 2022: 760. DOI: 10.1109/INFOCOM48880.2022.9796752

[13] BAO Yixin, PENG Yanghua, CHEN Yangrui, et al. Preemptive all-reduce scheduling for expediting distributed DNN training[C]// Proceedings of the 38th IEEE Conference on Computer Communications. Toronto: IEEE, 2020: 626. DOI: 10.1109/INFOCOM41043.2020.9155446

[14] ZHAO Xing, AN Aijun, LIU Junfeng, et al. Dynamic stale synchronous parallel distributed training for deep learning [C]// 39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019. Richardson: IEEE, 2019: 1507. DOI: 10.1109/ICDCS.2019.00150

[15] LIU Shuai, JU Tao. APapo: an asynchronous parallel optimization method for DNN models[J]. Future Generation Computer Systems, 2024, 152: 317. DOI: 10.1016/j.future.2023.11.004

[16] YU Enda, DONG Dezun, XU Yemao, et al. CP-SGD: distributed stochastic gradient descent with compression and periodic compensation[J]. Journal of Parallel and Distributed Computing, 2022, 169: 42. DOI: 10.1016/j.jpdc.2022.05.014

[17] TU Jun, ZHOU Jia, REN Donglin. An asynchronous distributed training algorithm based on gossip communication and stochastic gradient descent[J]. Computer Communications, 2022, 195: 416. DOI: 10.1016/j.comcom.2022.09.010

[18] XU Hang, ZHANG Wenxuan, FEI Jiawei, et al. SLAMB: accelerated large batch training with sparse communication [C]// Proceedings of the 40th International Conference on Machine Learning. Honolulu: ML Research Press, 2023: 38801

[19] GU Xinran, LYU Kaifeng, ARORA S, et al. A quadratic synchronization rule for distributed deep learning [C]//12th International Conference on Learning Representations. Vienna: ICLR, 2024: 14423

[20] WOODWORTH B, PATEL K K, SREBRO N. Minibatch vs Local SGD for heterogeneous distributed learning [C]//34th Conference on Neural Information Processing Systems. Vancouver: Neural Information Processing Systems, 2020

[21] MORITZ P, NISHIHARA R, STOICA I, et al. SparkNet: training deep networks in spark [C]//Proceedings of the 4th International Conference on Learning Representations. San Juan: ICLR, 2016

[22] COQUELIN D, DEBUS C, GÖTZ M, et al. Accelerating neural network training with distributed asynchronous and selective optimization (DASO)[J]. Journal of Big Data, 2022, 9(1): 14. DOI: 10.1186/s40537-021-00556-1

[23] WANG Jianyu, JOSHI G. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD[EB/OL]. (2019-03-07)[2024-06-17]. https://arxiv.org/abs/1810.08313