第47卷 第4期 计 算 机 工 程 2021年4月
Vol.47 No.4 Computer Engineering April 2021

·人工智能与模式识别· 文章编号：1000-3428(2021)04-0068-09 文献标志码：A 中图分类号：TP311

一种基于分布式编码的同步梯度下降算法

李博文，谢在鹏，毛莺池，徐媛媛，朱晓瑞，张 基

（河海大学 计算机与信息学院，南京 211100）

摘 要：基于数据并行化的异步随机梯度下降(ASGD)算法由于需要在分布式计算节点之间频繁交换梯度数据，从而影响算法执行效率。提出基于分布式编码的同步随机梯度下降(SSGD)算法，利用计算任务的冗余分发策略对每个节点的中间结果传输时间进行量化以减少单一批次训练时间，并通过数据传输编码策略的分组数据交换模式降低节点间的数据通信总量。实验结果表明，当配置合适的超参数时，与SSGD和ASGD算法相比，该算法在深度神经网络和卷积神经网络分布式训练中平均减少了53.97%、26.89%和39.11%、26.37%的训练时间，从而证明其能有效降低分布式集群的通信负载并保证神经网络的训练精确度。

关键词：神经网络；深度学习；分布式编码；梯度下降；通信负载

开放科学(资源服务)标志码(OSID)：

中文引用格式：李博文，谢在鹏，毛莺池，等.一种基于分布式编码的同步梯度下降算法[J].计算机工程，2021，47(4)：68-76，83.

英文引用格式：LI Bowen，XIE Zaipeng，MAO Yingchi，et al. A synchronized gradient descent algorithm based on distributed coding[J].Computer Engineering，2021，47(4)：68-76，83.

A Synchronized Gradient Descent Algorithm Based on Distributed Coding

LI Bowen，XIE Zaipeng，MAO Yingchi，XU Yuanyuan，ZHU Xiaorui，ZHANG Ji

（School of Computer and Information，Hohai University，Nanjing 211100，China）

【Abstract】The Asynchronized Stochastic Gradient Descent（ASGD） algorithm based on data parallelization require frequent gradient data exchanges between distributed computing nodes，which affects the execution efficiency of the algorithm. This paper proposes a Synchronized Stochastic Gradient Descent（SSGD） algorithm based on distributed coding. The algorithm uses the redundancy allocation strategy of computation tasks to quantify the intermediate transmission time of each node，and thus reduces the consumed time for training of a single batch.Then the amount of data transmitted between nodes is reduced by using the grouped data exchange mode of the coding strategy for data communication.Experimental results show that with a suitable hyper parameter configuration，the proposed algorithm can reduce the average distributed training time of Deep Neural Network（DNN） and Convolutional Neural Network（CNN） by 53.97% and 26.89% compared with the SSGD algorithm，and by 39.11% and 26.37% compared with the ASGD algorithm.It can significantly reduce the communication loads of the distributed cluster and ensures the training accuracy of neural networks.

【Key words】neural network；deep learning；distributed coding；Gradient Descent（GD）；communication load

DOI：10. 19678/j. issn. 1000-3428. 0057340

0 概述

近年来，深度学习技术已在语音识别[1-2]、计算机视觉[3-4]和自然语言处理[5]等领域得到广泛应用并取得了众多研究成果。在深度学习任务中常使用梯度下降（Gradient Descent，GD）算法作为深度神经网络（Deep Neural Network，DNN）模型训练的主要算法[6-7]，然而由于数据量的爆炸式增长，现有单机系统已无法满足梯度下降算法对计算资源的庞大需求。分布式技术通过多机协同计算的方式，使用多个计算节点扩充计算性能，可提供梯度下降算法所需的大量计算资源，因此研究梯度下降算法在分布式集

基金项目：国家自然科学基金重点项目（61832005）；国家重点研发计划（2016YFC0402710）。

作者简介：李博文（1996—），男，硕士研究生，主研方向为边缘计算、分布式计算；谢在鹏，副教授、博士；毛莺池，教授、博士；徐媛媛、朱晓瑞，博士；张 基，硕士研究生。

收稿日期：2020-02-06 **修回日期：**2020-04-04 **E-mail：**bowenli@hhu.edu.cn

群上的并行化计算非常必要[8]。

目前,深度神经网络分布式训练中通常使用数据并行化方法,基于数据并行化的梯度下降算法包括同步随机梯度下降[9](Synchronized Stochastic Gradient Descent,SSGD)和异步随机梯度下降[10](Asynchronized Stochastic Gradient Descent,ASGD)两种算法。同步算法在每次迭代过程中通过同步各个节点的状态来保持状态一致,因此同步梯度下降能够保持和单机串行算法相同的收敛方向,从而导致其所需的通信量相对其他算法更多,容易受分布式集群中的通信瓶颈影响,同时计算节点之间需要相互等待传输梯度更新结果,从而造成了计算资源的浪费。异步算法则无需等待节点状态一致,每个节点在更新权值参数时不会检查其他节点的运行状态,独立地更新本地模型到全局模型上,并获取全局模型参数继续执行下一步计算,如果在该计算过程中又有其他节点更新了全局模型,那么该节点用于执行下一步计算的全局模型参数就不是最新的,该数据交互模式将导致梯度延迟问题[11-12],而梯度延迟问题又会导致每个节点用于执行下一步计算的全局模型不一致。在大规模分布式集群中,随着分布式集群中的节点数目增加,节点计算状态保持一致的概率越来越小,节点之间所需传输的信息量逐渐增多,因此无论同步或异步算法,都在大规模分布式集群中存在亟待解决的效率问题。本文提出基于分布式编码的同步随机梯度下降算法CSSGD,通过分布式编码策略[13-14]降低分布式集群上的通信负载。

1 相关工作

近年来,分布式并行梯度下降算法得到了广泛应用,研究人员针对分布式神经网络训练开展了大量研究工作,这些工作主要围绕分布式异构计算和通信负载两方面展开。分布式集群是典型的异构计算平台,在该异构计算环境下,每个执行进程的执行速度和任务完成时间均不确定,在同步梯度下降算法中由于系统需要同步每一批次中各个计算节点的状态,因此最慢节点的速度成为限制分布式集群整体任务运行速度的主要因素。文献[15-16]通过对每个节点的计算和更新过程进行限时,使每个节点都在已有样本上进行逐样本迭代计算,以保证每个节点都能在任意时间内给出一个可用但不完整的中间梯度运算结果,降低了节点异构对整体任务的影响。文献[17-18]针对大矩阵乘法的并行化问题,提出纠缠多项式码,使用纠缠多项式码对矩阵乘法运算进行拆解,并对原始矩阵计算任务添加一定量的冗余以加快运算速度。

此外,并行梯度下降算法还需在各个计算节点之间进行密集地数据交换,因此分布式集群的通信性能也是影响分布式神经网络训练性能的主要问题。文献[19]提出1-Bit数值量化方案,使用Adagrad算法进行学习率迭代并结合误差累积回溯,在深度神经网络中得到一种低通信负载的分布式神经网络训练方案。文献[20]提出批次内并行化方法,使用较小的Block划分和基于动量的参数迭代方案提高了分布式神经网络的训练速度。文献[21]提出改进的并行梯度下降算法,在改进算法中每个节点不再将本地计算结果推送给分布式集群中的所有节点,而是随机选取一部分节点进行更新以降低通信负载。文献[22]结合MDS编码和分布式通信编码对分布式机器学习中的大矩阵乘法进行优化,有效解决了分布式系统慢节点和通信瓶颈问题,但其在编码过程中的冗余量较大。

上述文献分别从不同角度对分布式神经网络的训练过程进行分析与改进,然而在针对分布式通信瓶颈和分布式计算异构方面,使用量化或随机策略不可避免地会影响网络训练精确度,而利用MDS码则会引入更多的通信冗余。本文设计一种用于同步梯度下降算法的分布式编码策略,能够降低分布式集群上的通信负载,并保证分布式神经网络的训练精确度。

2 本文同步随机梯度下降算法

梯度下降算法是用于求解目标函数最小值的常用优化算法,当网络模型参数为w、数据集为x和y、目标方程为$F(w;x,y)$时,计算公式如式(1)所示,迭代过程的计算公式如式(2)所示,其中,$\nabla F(w;x,y)$为目标函数的一阶导数,η为迭代步长,又称为学习率,t为迭代轮次。在分布式环境下,并行梯度下降算法在每个节点i上求其部分样本的梯度,计算公式如式(3)所示,最终汇总至全局模型参数$w(t)$上的计算公式如式(4)所示。

$$\min_w F(w;x,y) \tag{1}$$

$$w(t)=w(t-1)-\eta\cdot\nabla F(w(t-1);x,y) \tag{2}$$

$$\Delta w_i(t)=\nabla F\left(w_i(t-1);x_i,y_i\right) \tag{3}$$

$$w(t)=w(t-1)-\eta\cdot\frac{1}{|x|}\sum_i\Delta w_i(t) \tag{4}$$

本文算法是同步梯度下降算法的一种改进形式,算法执行步骤为:1)将目标样本集合以预设批次大小进行划分,在添加冗余后将其平均分配到各个节点上;2)每个节点都在已有样本上执行反向传播算法,并对中间梯度结果进行编码,同时将中间梯度结果进行整合,实现数据分组交换;3)每个节点根据接收到的中间结果数据,解码出该节点所需的内容,继续执行训练过程。本文算法在任务分配时使用冗余分发策略,因此每个节点所需的中间结果数据可以从r个包含该数据的节点处获取,因此每个节点可以仅发送必要数据的$1/r$,由接收节点自行拼装完整

结果，如图1所示。同时，由于多个节点间存在冗余数据，因此一份数据可以被多个节点所共有，使用编码策略将需要发送的多个数据分片累加，将中间结果数据多播给多个节点，每个节点接收到累加结果后减去已知部分，即可得到所需部分的结果，如图2所示。本文结合上述两种策略并将其扩展至梯度矩阵信息交换过程中，提出能够有效降低梯度下降算法通信负载的方案。

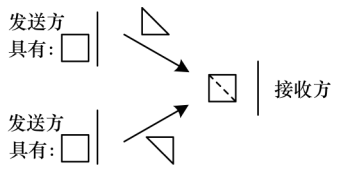

图1 冗余分发策略

Fig.1 Redundant distribution strategy

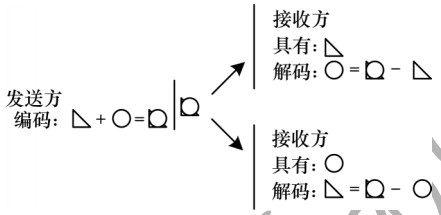

图2 编码策略

Fig.2 Coding strategy

本文同步随机梯度下降算法的伪代码如算法1所示，具体过程为在给定的训练数据集 F 和迭代次数 T 的情况下获取当前最终模型参数 w^f。

算法1 本文同步随机梯度下降算法

输入 dataset F, iteration T

输出 final model parameter w^f

1.initialize all w^f and $w_i(0)$ to the same value for all i;

2.each node i gets the dataset D_i using algorithm 2;

3.for t = 1, 2, ⋯, T do

4.for each node i in parallel, compute local update $\Delta w_i(t)$ using step 3;

5.for each node, encode and push $\Delta w_i(t)$ using algorithm 3;

6.for each node, receive and decode $\Delta w(t)$ using algorithm 4;

7. set $w_i(t) \leftarrow w(t)$ for all i, where $w(t)$ is defined in step 4;//全局结果合并

8.update $w^f \leftarrow \underset{w \in \{w^f, w(t)\}}{\operatorname{argmin}} F(w)$;

9.end for

10.return w^f;

2.1 任务分配过程

本文设计一种任务分配算法，在有 n 个计算节点的集群上，将训练数据集 F 冗余 r 倍并平均分配到每个节点上，且每个节点上的样本组合不重复。上述分配过程可以描述为一个简单的排列组合过程，首先要实现 r 倍冗余的分配，且每个节点上的数据组合互不重复，需要将每个样本的 r 份拷贝发送到任意 r 个不重复的计算节点上。其次要实现每个节点上的样本数量一致，将要分配的样本集合划分为 C_n^r 份，每一份样本子集都可以对应一个唯一的分发路径，依照组合结果进行分发，每个节点可以接收到 $(r \cdot C_n^r/n)$ 个样本集合，完成任务的平均分配。任务分配算法的伪代码如算法2所示，具体过程为在给定训练数据集 F、分布式节点集合 U 和训练所用的批次大小 b 的情况下获取每个计算节点 i 的本地样本 D_i，其中，combinations(U,r) 返回 U 的 r 倍所有可能的组合结果，split_into_batch(F,b) 返回按照批次大小划分后的数据集，split$\left(F_j, C.\text{size}\right)$ 返回 F_j 平均分为 C.size后的结果集合。

算法2 任务分配算法

输入 dataset F, node set U, batch size b

输出 data assignment D for each node i

1.C ← combinations(U, r);//获取组合结果

2.F ← split_into_batch(F, b);//获取样本 mini batch

3.D ← $\{D_i|D_i=\varnothing\}$;//为每个节点初始化私有样本集合

4.for each F_j in F do

5.P ← split$\left(F_j, C.\text{size}\right)$;//将 F_j 均分为 C_n^r 份

6.for i = 1, 2, ⋯, C.size do

7.for each N_i in C_i do

8.D_i.add$\left(P_i\right)$;

9.end for

10.end for

11.end for

12.return D;

为阐述算法2的执行过程，首先定义分布式集群中所有节点的集合为 U，如式(5)所示。取 U 的 r 倍组合结果记作 C，如式(6)所示，在 C 中的元素个数为 C_n^r。

$$U \triangleq \{N_i | i = 0, 1, \cdots, n-1\} \tag{5}$$

$$C \triangleq \text{combinations}(U, r) \tag{6}$$

同样地，将训练样本按照输入的批次大小 b 划分为多个 mini batch，并将每个 mini batch F_j 平均分为 C_n^r 份并记作 P。将 P_i 和 C_i 一对一配对，依次将每个 P_i 发送至对应 C_i 表示的节点集合上。此时，每个节点上存在的待处理样本块个数为 $\mu = r \cdot C_n^r/n$，且每个样本被拷贝并发送至 r 个不同节点上。依照上述算法流程可将数据集 F 平均分配到 N 中的每个计算节点上，每个节点上的样本数据数目可根据节点的性能进行动态调整，且在一定限度内不会影响其他节点的负载。

2.2 编码过程

本文通过对中间结果矩阵进行编码，可压缩每个节点发送的数据量，同时实现数据多播以减少发送时间。由于在任务分配时使用冗余分发策略，因此每个节点所需的中间结果数据可以从 r 个包含该数据的节点处获取，因此每个节点可以仅发送必要

数据的1/r，由接收节点自行拼装完整结果。同时，由于多个节点间存在冗余数据，因此一份数据可以被多个节点共有，使用编码策略将要发送的多个数据分片累加，将中间结果数据多播给多个节点，每个节点接收到累加结果后减去已知部分，即可得到所需部分的结果。上述编码过程可在不影响计算速度的情况下有效降低发送的数据量，一般情况下该过程可将数据量降低至朴素方法的1/n。

数据传输编码算法的伪代码如算法3所示，具体过程为在给定所有中间结果矩阵的列表B、当前节点N和冗余度r的情况下获取中间结果集合$\mathcal{C}$，其中，position_of($\boldsymbol{G}$,N)返回在编码过程中节点N获取梯度矩阵$\boldsymbol{G}$的拆分子矩阵编号，block_id_of($\boldsymbol{G}$)返回用于计算$\boldsymbol{G}$所使用的样本块编号，content_of($\boldsymbol{G}$,N)返回在编码过程中节点N应获取的对应$\boldsymbol{G}$的拆分子矩阵，adversary_of($\boldsymbol{G}$)返回不具有$\boldsymbol{G}$的所有节点，get_singularity(T)返回列表T中只出现一次的元素，pack(M,P,J,T)将内容M、P、J、T打包并等待发送。

算法3　数据传输编码算法

输入　list of all gradient matrix B, current node N, redundancy r

输出　intermediate result set $\mathcal{C}$

```
initialize C as empty set;
for each R in combinations(B, r) do
initialize M as 0;
initialize P, J as empty set;
initialize T as empty list;
for each G in R do
P ← P + position_of(G, N);
J ← J + block_id_of(G);
M ← M + content_of(G, N);
T ← T + adversary_of(G);
end for
T ← get_singularity(T);
C ← C + pack(M, P, J, T);
end for
return C;
```

按照算法3的执行过程，定义编码的运算过程发生在节点N_i、神经网络层L_l、批次样本F_j和批次样本中的块P_q上，四元组(i,l,j,q)表示运算过程发生的环境，例如，在节点N_i和层L_l上批次为F_j且由全局编号为q的样本块计算所得的梯度记作$\boldsymbol{G}_{(i,l,j,q)}$。当某个对象的存在与某个下标对应的维度无关时，将其标记为数学中的任意符号$\forall$，例如样本的存在与神经网络层数无关，因此在节点N_i上批次为F_j且全局编号为i的样本块记作$F_{(i,\forall l,j,q)}$。在样本块分发完毕后，每个节点在其接收到的样本块上执行前向传播和反向传播算法，计算以该样本块所得损失对网络参数$w_{(i,l,j,q)}$的一阶梯度，一个块内所有样本的对应损失的梯度加和记作$\boldsymbol{G}_{(i,l,j,q)}$。

将节点N_i在任务分配阶段获得的μ个第j批次样本块在层$\mathcal{L}_l$上计算得到的梯度矩阵记作$\boldsymbol{G}_{(i,l,j,\forall)}$，如式(7)所示。在$B_{(i,l,j,\forall q)}$中取$r$个元素进行组合，将包含$C_\mu^r$个组合结果的结果集记作$R_{(i,l,j,\forall q)}$，如式(8)所示。将$R_{(i,l,j,\forall q)}$中的组合结果记作$R^s_{(i,l,j,\forall q)}$，如式(9)所示。

$$B_{(i,l,j,\forall q)} \triangleq \left\{ \boldsymbol{G}_{(i,l,j,q_z)} : z = 1,2,\cdots,\mu \right\} \tag{7}$$

$$R_{(i,l,j,\forall q)} \triangleq \text{combinations}(B, r) \tag{8}$$

$$R^s_{(i,l,j,\forall q)} \triangleq \left\{ \boldsymbol{G}_{(i,l,j,d_1)}, \boldsymbol{G}_{(i,l,j,d_2)}, \cdots, \boldsymbol{G}_{(q,l,j,d_r)} : d_1 < d_2 < \cdots < d_r \right\} \tag{9}$$

本文依次获取$R_{(i,l,j,\forall q)}$中的组合结果$R^s_{(i,l,j,\forall q)}$，对于每一个组合结果$R^s_{(i,l,j,\forall q)}$，将其编码成一个要发送的数据包，数据包中包含组合结果的所有块编号$P^s_{(i,l,j,\forall q)}$、组合结果拆分位置$J^s_{(i,l,j,\forall q)}$和组合结果内容$M^s_{(i,l,j,\forall q)}$。初始化$P^s_{(i,l,j,\forall q)}$、$J^s_{(i,l,j,\forall q)}$、$M^s_{(i,l,j,\forall q)}$及发送目的节点集合$T^s_{(i,l,j,\forall q)}$为空。将集合$R^s_{(i,l,j,\forall q)}$中的元素$\boldsymbol{G}_{(i,l,j,d)}$进行拆分，通过按行向量或者按列向量进行拆分，梯度矩阵按列拆分的示例如图3所示。

$$\left(\begin{array}{cc|cc|cc} 1.4 & 0.8 & 0.8 & 0.3 & 0.5 & -0.5 \\ 0.2 & 0.9 & 0.1 & 0.6 & 0.1 & 0.7 \\ 1.1 & -0.9 & 0.2 & 1.1 & 0.4 & 0.9 \\ -0.9 & 0.7 & -1.9 & 0.8 & 0.3 & 0.8 \end{array}\right)$$

图3　梯度矩阵按列拆分的示例

Fig.3　Example for splitting a gradient matrix by column

将梯度矩阵$\boldsymbol{G}_{(i,l,j,d)}$拆分成$r$份，每个拆分子矩阵记作$B^{s,k}_{(i,l,j,d)}$，$k=0,1,\cdots,r-1$。将拆分所得的结果集合记作$\mathcal{G}^s_{(i,l,j,\forall i)}$，如式(10)所示。对每一个梯度矩阵$\boldsymbol{G}_{(i,l,j,d)}$，根据其计算来源样本块$F_{(i,l,j,d)}$，得到其在数据分发时对应的分发节点集合$C_i$，如式(11)所示。

$$\mathcal{G}^s_{(i,l,j,\forall i)} \triangleq \left\{ B^{s,k}_{(i,l,j,d)} : k=0,1,5\cdots,r-1,\ d=d_1,d_2,\cdots,d_r \right\} \tag{10}$$

$$C_i \triangleq \left\{ N_{a_1}, N_{a_2}, \cdots, N_{a_r} : a_1 < a_2 < \cdots < a_r \right\} \tag{11}$$

将C_i中的节点下标编号a_q进行升序排序，获取当前下标节点N_q在升序排列后的节点列表中的位置记作α，例如在节点组合$\{N_1,N_2,N_3\}$中，节点N_1对应的位置$\alpha=0$，节点N_3对应的位置$\alpha=2$。获取位置信息α_i并将其添加至组合结果拆分位置信息$P^s_{(i,l,j,\forall q)}$，同时将该块编号信息d添加至块编号信息$J^s_{(i,l,j,\forall q)}$。对应每个样本块都可获取一个该节点在该样本块分发列表中的位置α_i，那么可以由上述数据

求得 $M^{s}_{(i,l,j,\forall q)}$，如式(12)所示。遍历编码 $M^{s}_{(i,l,j,\forall q)}$ 所用的编码梯度矩阵 $\boldsymbol{G}_{(i,l,j,\forall q)}$ 求解 A_i，如式(13)所示。叠加所有的 A_i 并获取其中仅出现一次的节点，将其添加至编码包的发送目标列表 $T^{s}_{(i,l,j,\forall q)}$。

$$M^{s}_{(i,l,j,\forall q)}=\sum_{d}B^{s,\alpha_d}_{(i,l,j,d)} \tag{12}$$

$$A_i \triangleq U-C_i \tag{13}$$

至此，就能获取编码包所需的 $P^{s}_{(i,l,j,\forall q)}$、$J^{s}_{(i,l,j,\forall q)}$、$M^{s}_{(i,l,j,\forall q)}$ 和 $T^{s}_{(i,l,j,\forall q)}$ 信息并将发送信息记作 $C^{s}_{(i,l,j,\forall q)}$，如式(14)所示。当 $C^{s}_{(i,l,j,\forall q)}$ 缓存至空闲网络时，多播上述信息至 $T^{s}_{(i,l,j,\forall q)}$ 所有节点上。

$$C^{s}_{(i,l,j,\forall q)}\triangleq\text{pack}\left(P^{s}_{(i,l,j,\forall q)},J^{s}_{(i,l,j,\forall q)},M^{s}_{(i,l,j,\forall q)},T^{s}_{(i,l,j,\forall q)}\right) \tag{14}$$

2.3 解码过程

根据编码时的累加结果和分片规则，解码时应按照编码时的运算过程反向求取节点所需的内容。首先通过减法减去节点已知的数据内容，求取未知的数据内容，然后按照分片规则，还原中间结果矩阵，以获取完整的中间结果。数据传输解码算法的伪代码如算法4所示，主要功能为在给定所有中间结果矩阵列表 B 和接收到的编码数据包 H 的情况下输出矩阵切片(content)、切片编号(parts_id)和切片位置(parts_ position)，其中，can_get(B, id)判断节点是否包含样本块编号为id所计算出的梯度矩阵，part_of(B, id, pos)获取对应编号id的样本块计算出的梯度矩阵的第pos个拆分子矩阵。

算法4 数据传输解码算法

输入 list of all gradient matrix B, coded package H

输出 content, parts_id, parts_position

```
1.ids ← H.block_ids; //获取H的块编号
2.poss ← H.positions; //获取H中内容的位置编号
3.content ← H.content; //获取H中包含的矩阵内容
4.for each (id, pos) in (ids, poss) do
5.if can_get(B, id) then
6.content ← content − part_of(B, id, pos);
7.else
8.parts_id ← id;
9.parts_position ← pos;
10.end if
11.end for
12.return content, parts_id, parts_position;
```

在编码包中已包含编码所用样本块编号和编码拆分矩阵的位置信息，依据发送规则可从编码包中提取节点 N_i 缺失的信息，将发送编码包给节点 N_i 的节点记作 $N_{i'}$，将节点 $N_{i'}$ 发送返回的信息记作 $H^{s}_{(i',l,j,\forall q)}$。由于编码包中已包含样本块和拆分子矩阵位置信息，因此可利用节点 N_i 已有的信息解码未知信息，如式(15)所示。根据任务分配算法，将节点 N_i 缺失的样本块记作 $F_{(i,l,j,q_v)}$，$v=1,2,\cdots,C^{r}_{n}-\mu$，对应这些缺失样本所需的梯度中间结果记作 $\boldsymbol{G}_{(i,l,j,q_v)}$，$v=1,2,\cdots,C^{r}_{n}-\mu$。当节点 N_q 获取到所有缺失信息 $M^{k}_{(\forall i,l,j,q_v)}$ 后，将其依照拆分规则进行合并，如式(16)所示。当节点 N_q 获取到所有样本块对应的缺失梯度时，使用式(16)和式(17)计算全局梯度，该全局梯度即进一步迭代所需的模型参数增量，如式(18)所示。

$$M^{k}_{(\forall i,l,j,q_v)}=H^{s}_{(i',l,j,\forall q)}-\sum_{q_z,q_z\neq q_v}B_{(i,l,j,q_z)} \tag{15}$$

$$\boldsymbol{M}_{(\forall i,l,j,q_v)}=\begin{pmatrix}M^{0}_{(\forall i,l,j,q_v)}\\M^{1}_{(\forall i,l,j,q_v)}\\\vdots\\M^{r}_{(\forall i,l,j,q_v)}\end{pmatrix} \tag{16}$$

$$\boldsymbol{G}_{(i,l,j,\forall)}=\frac{1}{\left|F_j\right|}\left[\sum_{i_v}\boldsymbol{M}_{(i,l,j,q_v)}+\sum_{i_z}\boldsymbol{G}_{(i,l,j,q_z)}\right] \tag{17}$$

$$\Delta w_i(t)\leftarrow\boldsymbol{G}_{(i,l,j,\forall)} \tag{18}$$

2.4 通信负载计算

若假设本文同步随机梯度下降算法的通信负载为 L_{coding}，传统不使用多播技术的梯度下降算法的通信负载为 L_{normal}，则两者存在式(19)所述的关系：

$$\frac{L_{\text{coding}}}{L_{\text{normal}}}=\frac{C^{r+1}_{n}\cdot(r+1)\cdot\dfrac{1}{r}}{(n-1)\cdot C^{r}_{n}}=\frac{n-r}{n-1}\cdot\frac{1}{r} \tag{19}$$

对上述关系进行证明，具体证明过程如下：使用编码策略进行任务分发和交换，节点数目为 n，数据分配冗余度为 r，每个划分占总批次大小为 $a_1,a_2,\cdots,a_p$，其中 $p=C^{r}_{n}$，每个节点获取 $a_1,a_2,\cdots,a_p$ 中的 D_{have} 个样本块(如式(20)所示)，并缺少其中的 D_{left} 个样本块(如式(21)所示)。将节点 i 包含的所有样本块集合记作 $\mathcal{D}_i$，如式(22)所示。

$$D_{\text{have}}\triangleq r\cdot C^{r}_{n}/n=C^{r-1}_{n-1} \tag{20}$$

$$D_{\text{left}}\triangleq C^{r}_{n}-r\cdot C^{r}_{n}/n \tag{21}$$

$$\mathcal{D}_i\triangleq\{P_i|P_i\text{ 是发送给节点 }i\text{ 的样本块}\} \tag{22}$$

对于 n 个节点中任意 r 个节点组成的节点子集，其中一定存在1个样本块被这 r 个节点所共有，因为样本块分发标签包含了所有节点编号可以形成的组合，所以任意 r 个节点必然属于 C^{r}_{n} 种选择中的一个节点，如式(23)所示。同理，对于任意 $r+1$ 个节点组成的节点组至少存在 C^{r}_{r+1} 个这样的样本块，如式(24)所示。在 n 个节点组成的集群中共有 C^{r+1}_{n} 个节点组，每个节点组选取其中 C^{r}_{r+1} 个样本块 $\mathcal{D}_{\text{exchange}}$ 进行数据交换，每个节点仅发送由样本块 $\mathcal{D}_{\text{send}}$ 计算

所得的数据，如式(25)、式(26)所示。

$$\left|\mathcal{D}_{i_1}\cap\mathcal{D}_{i_2}\cap\cdots\cap\mathcal{D}_{i_r}\right|=1 \tag{23}$$

$$\left|\mathcal{D}_{i_1}\cap\mathcal{D}_{i_2}\cap\cdots\cap\mathcal{D}_{i_{r+1}}\right|=r+1 \tag{24}$$

$$\mathcal{D}_{\text{exchange}}\triangleq\mathcal{D}_{i_1}\cap\mathcal{D}_{i_2}\cap\cdots\cap\mathcal{D}_{i_{r+1}} \tag{25}$$

$$\mathcal{D}_{\text{send}}\triangleq\mathcal{D}_{i_1}\cap\mathcal{D}_{i_2}\cap\cdots\cap\mathcal{D}_{i_r} \tag{26}$$

每个包含$\mathcal{D}_{\text{exchange}}$的节点都互相不重复地将$\mathcal{D}_{\text{send}}$的$1/r$信息多播给其他所有节点。因为任意一个样本块只可能被一个节点组完全包含，所以每个节点组内部沟通的信息是互不重复的。每个节点发送$1/r$的信息，在一次交互完成后，每个节点总能从其他节点发送的$(r\cdot 1/r)$数据中获取一个完整的中间结果。取任意节点N_i，节点N_i能够参与到C_n^{r+1}中的C_{n-1}^r个节点组中进行数据交换，那么在C_{n-1}^r次数据交换后，节点N_i能够获得$(r\cdot 1/r\cdot C_{n-1}^r)$份中间计算结果，其中：

$$r\cdot\frac{1}{r}\cdot C_{n-1}^r=C_{n-1}^r=C_n^r\cdot\left(\frac{n-r}{n}\right)=C_n^r-\frac{r\cdot C_n^r}{n}=D_{\text{left}} \tag{27}$$

由于节点N_i完成所有分组交换后数据都不重复，因此其能够获得所有的D_{left}中间信息，使用C_n^{r+1}个节点组进行完全分组信息交换后，总的多播通信量如式(28)所示。对于具有同样冗余度的传统通信方式，每个节点所需通信量如式(29)所示，本文中的冗余度是指在同一样本块上进行计算的节点数量。对于不使用冗余策略的普通通信方式，每个节点缺少的数据是相互独立的，所需的通信量如式(30)所示。综上，式(19)所述的编码策略通信负载与传统方法通信负载之间的关系得以证明。

$$L_{\text{coding}}=C_n^{r+1}\cdot(r+1)\cdot\frac{1}{r} \tag{28}$$

$$L_{\text{uncoding}}=n\cdot D_{\text{left}}=(n-r)\cdot C_n^r \tag{29}$$

$$L_{\text{normal}}=(n-1)\cdot C_n^r \tag{30}$$

3　实验与结果分析

3.1　实验方法

本文通过神经网络对SSGD、ASGD和本文CSSGD算法进行分布式训练实验，对比在不同节点数目的情况下3种算法到达指定训练精确度所需物理时钟时间和数据传输量，并利用集群模拟对上述指标进行实验验证，实验中共使用10个计算节点，每个计算节点的配置信息如表1所示。本文使用MNIST和CIFAR-10数据集在卷积神经网络(Convolutional Neural Network，CNN)和DNN上进行实验。表2给出CNN和DNN在不限时单机执行环境下所得的测试精确度，以此测试精确度作为最优测试精确度，并在实验中对比3种算法的测试精确度。

表1　计算节点配置信息

Table 1　Configuration information of computing node

配置项目	配置信息
CPU	Intel Xeon 系列
内存容量/GB	8
网络类型	交换式以太网
网络速度/(Gb·s^{-1})	10

表2　实验数据集上的测试精确度

Table 2　Test accuracy on the experimental dataset

网络	数据集	测试精确度/%
DNN	MNIST	88.02
CNN	CIFAR-10	80.37

为保证浮点数执行编码运算后的精确度，本文对计算产生的中间结果进行量化，使用32位整型保存运算所需的浮点数据，设置中间结果区间为[−10，10]，取0x7FFFFFFF为10，取0x80000001为−10，等区间划分取值，并使用该量化结果进行运算和编码。本文在实验中对不同节点的网络使用不同的样本划分，在2个节点上只执行二分类，在3个节点上只执行三分类任务，以此类推，保证每次测试时每个节点都能够执行相同数据量的任务，分析由通信瓶颈导致的性能损失问题，同时以串行算法测试所得结果作为目标精确度。

3.2　实验结果

本文首先对CSSGD算法在DNN与CNN上的训练结果进行评估，然后使用SSGD、ASGD和CSSGD算法在MNIST数据集上训练DNN、在CIFAR-10数据集上训练CNN。在训练过程中，使用10个训练节点(ASGD算法额外使用一个参数服务器)并记录测试精确度的变化情况。由于CSSGD算法的目的是减少训练执行时间，因此实验着重讨论3种算法到达相同测试精确度的时间。

3.2.1　训练性能分析

DNN和CNN的训练结果如图4和图5所示，可以看出CSSGD、SSGD和ASGD算法在充足的训练时间下，均可达到最终的测试精确度。

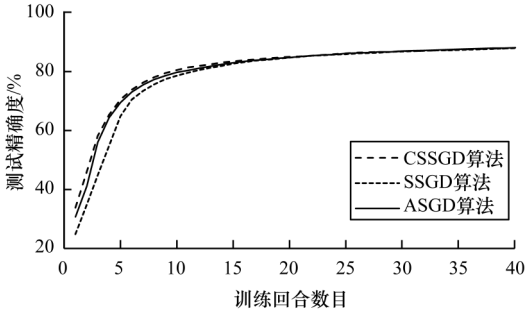

图4　利用CSSGD、SSGD和ASGD算法训练DNN的测试精确度比较

Fig.4　Comparison of test accuracy of training DNN with CSSGD，SSGD and ASGD algorithms

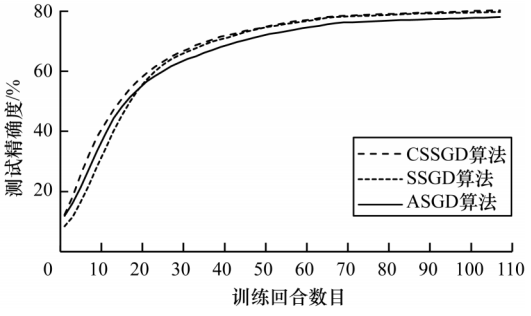

图5 利用CSSGD、SSGD和ASGD算法训练CNN的测试精确度比较

Fig.5 Comparison of test accuracy of training CNN with CSSGD, SSGD and ASGD algorithms

本节实验着重分析3种算法训练所需的时间，在限定的测试精确度下比较算法执行所需的时间以评估算法执行效率。实验在CNN和DNN的原始训练结果中，在接近目标精确度附近选取一个相对3种算法差异最大的基准测试精确度进行后续实验。如图6、图7所示，对于使用MNIST数据集的DNN，其选取的基准测试精确度为82%，对于使用CIFAR-10数据集的CNN，其选取的基准测试精确度为70%。下文将针对这组基准测试精确度，比较3种算法的执行效率。

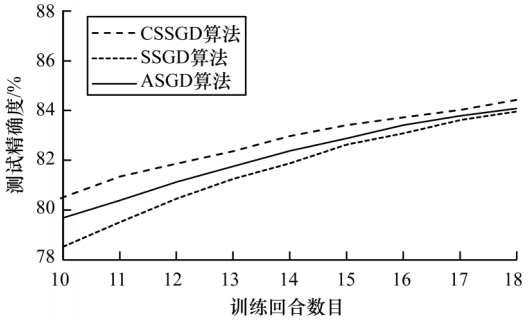

图6 DNN训练中的基准测试精确度

Fig.6 Benchmark test accuracy of DNN training

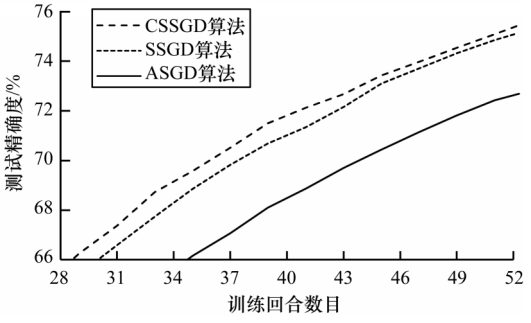

图7 CNN训练中的基准测试精确度

Fig.7 Benchmark test accuracy of CNN training

后续实验从3个方面详细分析了CSSGD算法的执行效率：1)由于ASGD算法无独占通信时间，因此在DNN和CNN训练过程中，对比使用CSSGD与SSGD算法在训练时间和通信总量上的差别；2)针对理论推导结果，通过实验验证不同冗余度设置下CSSGD算法的每批次训练时间；3)使用相同的样本数据集和神经网络在上述集群环境下，对比ASGD、SSGD和CSSGD算法训练至基准测试精确度所需的训练时间。

3.2.2 DNN实验结果分析

图8展示了DNN训练中SSGD和CSSGD算法在MNIST数据集上达到82%测试精确度所需的时间，其中，SSGD Multicast和SSGD Unicast分别表示基于多播技术的SSGD算法和基于单播技术的SSGD算法，CSSGD r=2和CSSGD r=3分别表示冗余度为2的CSSGD算法和冗余度为3的CSSGD算法，可以看出CSSGD算法能够有效降低SSGD算法执行所需时间。图9展示了DNN训练中SSGD Multicast和CSSGD算法占用的网络数据通信总量，可以看出CSSGD算法占用的网络数据通信总量相对SSGD算法更少。

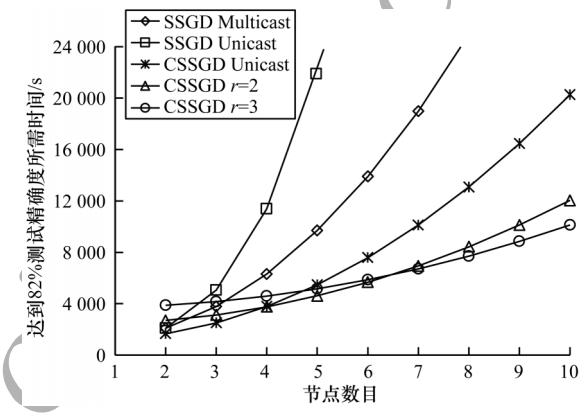

图8 在DNN训练中SSGD和CSSGD算法达到基准测试精确度所需时间

Fig.8 The time required for SSGD and CSSGD algorithms to reach benchmark test accuracy in DNN training

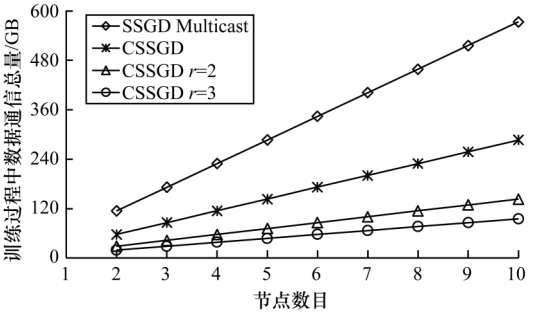

图9 在DNN训练中SSGD和CSSGD算法达到基准测试精确度所需通信总量

Fig.9 The total amount of communication required for SSGD and CSSGD algorithms to reach benchmark test accuracy in DNN training

3.2.3　CNN实验结果分析

图10展示了CNN训练中SSGD与CSSGD算法在CIFAR-10数据集上达到70%测试精确度所需的时间。在DNN训练过程中产生的权值矩阵相对CNN更大，容易由于网络速度限制造成计算瓶颈，如果对该类别网络使用CSSGD算法，则可以有效缓解通信瓶颈问题。在CNN中执行计算所需时间比DNN更多，因此其对通信瓶颈的影响相对DNN更小，在使用CSSGD算法进行编码梯度下降时，其所能获得的测试精确度相对DNN更低。图11展示了CNN训练中SSGD Multicast和CSSGD算法占用的网络数据通信总量，可以看出CSSGD算法在CNN和DNN训练中都能够有效降低中间结果的通信总量，但是CNN计算中数据通信所占时间相较DNN更少，因此数据通信优化对总体执行性能的影响较小。

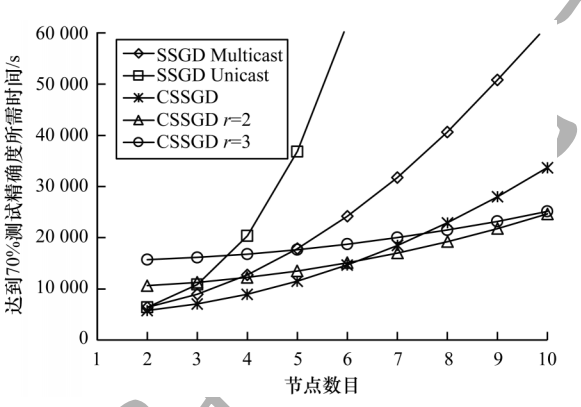

图10　在CNN训练中SSGD和CSSGD算法达到基准测试精确度所需时间

Fig.10　The time required for SSGD and CSSGD algorithms to reach benchmark test accuracy in CNN training

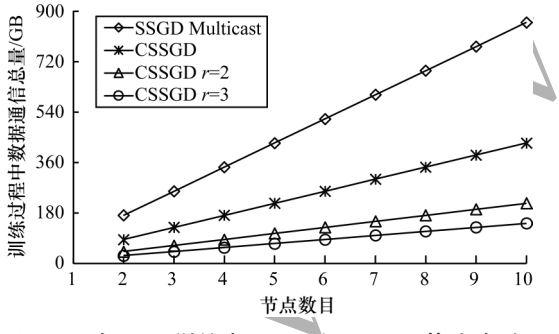

图11　在CNN训练中SSGD和CSSGD算法达到基准测试精确度所需通信总量

Fig.11　The total amount of communication required for SSGD and CSSGD algorithms to reach benchmark test accuracy in CNN training

3.2.4　冗余度分析

根据上文对编码过程的理论推导可以得出，当所需通信时间与数据通信总量呈线性相关时，CSSGD算法单一批次训练所需时间T_{CSSGD}可表示为每批次计算时间和每批次通信时间之和，假设SSGD多播算法中一个批次计算所需时间为$T_{\text{calculate}}$、中间结果传输所需时间为$T_{\text{communicate}}$，则T_{CSSGD}可表示为$T_{\text{CSSGD}}=r\cdot T_{\text{calculate}}+1/r\cdot T_{\text{communicate}}$，可以看出，当$T_{\text{calculate}}$和$T_{\text{communicate}}$为定值时存在一个确定的冗余度使得$T_{\text{CSSGD}}$最小。本文在8个计算节点的情况下对不同冗余度的CSSGD算法单一批次迭代所需时间进行实验，结果如图12、图13所示，可以看出在DNN和CNN训练中的最佳冗余度分别为3和2。由实验结果可以得出和理论推导相同的结论，即在固定的计算时间和传输时间下，每个节点都存在确定的冗余度，当冗余度为r时整个分布式集群单一批次的训练时间最短。

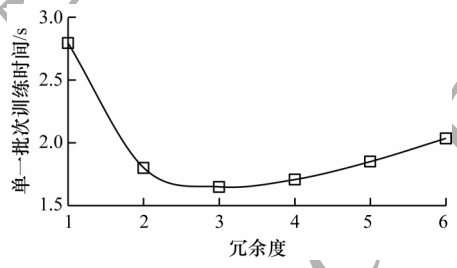

图12　在DNN训练中CSSGD算法冗余度与单一批次训练时间的关系

Fig.12　The relationship between the redundancy and the training time of single batch of CSSGD algorithm in DNN training

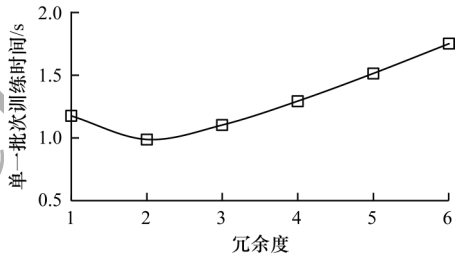

图13　在CNN训练中CSSGD算法冗余度与单一批次训练时间的关系

Fig.13　The relationship between the redundancy and the training time of single batch of CSSGD algorithm in CNN training

3.2.5　训练时间分析

本节对比ASGD、SSGD Multicast和最优冗余度下的CSSGD（CSSGD Optimal）算法的实际训练时间，如图14和图15所示，由于ASGD算法异步执行过程导致其执行状态难以确定，最终收敛所需次数也存在一定的随机性，因此对ASGD算法进行10次实验取训练时间的平均值。可以看出，在通信瓶颈问题较严重的DNN训练过程中，CSSGD算法相对于SSGD和ASGD算法可取得明显的效率提升，在DNN的分布式训练中，相对SSGD和ASGD平均能减少53.97%和26.89%的训练时间。在CNN训练过程中由于计算执行过程占总执

行时间的比例较高，因此通过添加冗余编码来降低通信负载的CSSGD算法效率提升较少，但是ASGD算法需要更多的计算开销来消除梯度延迟问题带来的干扰，因此CSSGD算法相对SSGD和ASGD算法效率依然有一定的提升，在CNN分布式训练中，相对SSGD和ASGD平均能减少39.11%和26.37%的训练时间。实验中给出10个计算节点以内的训练时间结果，可以看出，ASGD和CSSGD算法在分布式系统上的性能不能随节点数目线性提升，随着节点数目的增多，ASGD算法的理论同步更新概率会越来越小，迭代所需时间会继续增加，CSSGD算法也会受到额外通信负载和多播带宽拥塞的影响，但是其在更多节点加入后可更新其最优冗余度参数配置，能够达到更好的训练效果。

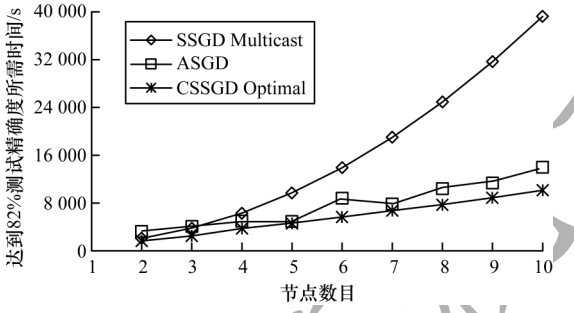

图14 在DNN训练中SSGD、ASGD和CSSGD算法达到基准测试精确度所需时间

Fig.14 The time required for SSGD, ASGD and CSSGD algorithms to reach benchmark test accuracy in DNN training

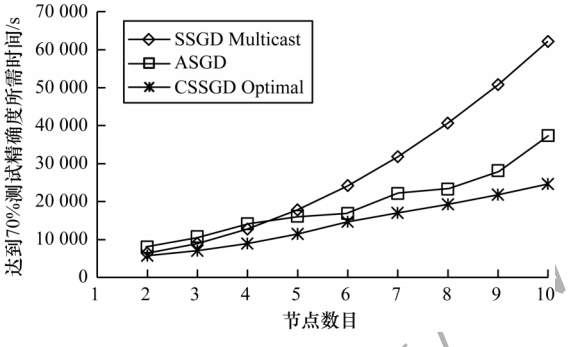

图15 在CNN训练中SSGD、ASGD和CSSGD算法达到基准测试精确度所需时间

Fig.15 The time required for SSGD, ASGD and CSSGD algorithms to reach benchmark test accuracy in CNN training

4 结束语

本文针对异步随机梯度下降算法的高通信负载问题，提出一种基于分布式编码计算的同步梯度下降算法，通过冗余分发策略降低通信负载并消除通信瓶颈对分布式集群的影响。实验结果表明，当配置合适的超参数时，与SSGD和ASGD算法相比，该算法在DNN分布式训练中能平均减少53.97%和26.89%的训练时间，在CNN分布式训练中能平均减少39.11%和26.37%的训练时间，降低了分布式集群上的通信负载。下一步将研究并行梯度下降算法在分布式集群上的应用，并分析数值量化误差对最终损失函数收敛性能的影响。

参考文献

[1] SAK H, SENIOR A, BEAUFAYS F. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition[EB/OL]. [2020-01-04]. https://arxiv. org/abs/1402. 1128.

[2] SERCU T, PUHRSCH C, KINGSBURY B, et al. Very deep multilingual convolutional neural networks for LVCSR [C]//Proceedings of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing. Washington D. C. , USA: IEEE Press, 2016: 4955-4959.

[3] KRIZHEVSKY A, SUTSKEVER I, GEOFFREY E H. ImageNet classification with deep convolutional neural networks [EB/OL]. [2020-01-04]. https://blog. csdn. net/yuanchheneducn/article/details/50161047.

[4] HE Kaiming, ZHANG Xiangyu, REN Shaoqing, et al. Deep residual learning for image recognition[C]//Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition. Washington D. C. , USA: IEEE Press, 2016: 770-778.

[5] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed representations of words and phrases and their compositionality[C]// Proceedings of International Conference on Neural Information Processing Systems. Washington D. C. , USA: IEEE Press, 2013: 3111-3119.

[6] KINGMA D P. ADAM: a method for stochastic optimization[C]//Proceedings of International Conference on Learning Representations. Washington D. C. , USA: IEEE Press, 2015: 1-7.

[7] LUO Liangchen, XIONG Yuanhao, LIU Yan, et al. Adaptive gradient method with dynamic bound of learning rate [C]//Proceedings of International Conference on Learning Representations. Washington D. C. , USA: IEEE Press, 2019: 15-25.

[8] DEAN J, CORRADO G S, MONGA R, et al. Large scale distributed deep networks[C]//Proceedings of International Conference on Neural Information Processing Systems. New York, USA: ACM Press, 2013: 1223-1231.

[9] POVEY D, ZHANG X H, KHUDANPUR S. Parallel training of DNNs with natural gradient and parameter averaging[C]//Proceedings of International Conference on Learning Representations. New York, USA: ACM Press, 2015: 7-18.

[10] NIU F, ECHT B. HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent[C]//Proceedings of the 25th Conference on Neural Information Processing Systems. New York, USA: ACM Press, 2011: 693-701.

[11] DAI Wei, ZHOU Yi, DONG Nanqing, et al. Toward understanding the impact of staleness in distributed machine learning [C]//Proceedings of International Conference on Learning Representations. New York, USA: ACM Press, 2019: 1-8.

（下转第83页）

（上接第76页）

[12] ZHENG Shuxin, MENG Qi, WANG Taifeng, et al. Asynchronous stochastic gradient descent with delay compensation[C]//Proceedings of International Conference on Machine Learning. New York, USA: ACM Press, 2017: 28-45.

[13] LI S Z, MADDAH-ALI M A, YU Q, et al. A fundamental tradeoff between computation and communication in distributed computing[J]. IEEE Transactions on Information Theory, 2018, 64(1): 109-128.

[14] LI S Z, MADDAH-ALI M A. Compressed coded distributed computing [C]//Proceedings of 2018 IEEE International Symposium on Information Theory. Washington D. C., USA: IEEE Press, 2018: 2032-2036.

[15] FERDINAND N, AL-LAWATI H, DRAPER S, et al. Anytime minibatch: exploiting stragglers in online distributed optimization[EB/OL]. [2020-01-04]. https://arxiv. org/abs/2006. 05752.

[16] FERDINAND N, GHARACHORLOO B, DRAPER S C. Anytime exploitation of stragglers in synchronous stochastic gradient descent[C]//Proceedings of the 16th IEEE International Conference on Machine Learning and Applications. Washington D. C., USA: IEEE Press, 2017: 141-146.

[17] YU Q, MADDAH-ALI M A. Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding [C]//Proceedings of IEEE International Symposium on Information Theory. Washington D. C., USA: IEEE Press, 2018: 2157-2162.

[18] YU Q, MADDAH-ALI M A. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication [C]//Proceedings of the 31st Conference on Neural Information Processing Systems. New York, USA: ACM Press, 2017: 4406-4416.

[19] SEIDE F, FU H, DROPPO J, et al. 1-Bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs[EB/OL]. [2020-01-04]. https://www. cnblogs. com/littleorange/p/12674552. html

[20] CHEN Kai, HUO Qiang. Scalable train of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering [C]//Proceedings of International Conference on Acoustics, Speech and Signal Processing. Washington D. C., USA: IEEE Press, 2016: 2379-2384.

[21] ASSRAN M, LOIZOU N, BALLAS N, et al. Stochastic gradient push for distributed deep learning [EB/OL]. [2020-01-04]. https://arxiv. org/abs/1811. 10792.

[22] LEE K, LAM M, PEDARSANI R, et al. Speeding up distributed machine learning using codes [J]. IEEE Transactions on Information Theory, 2018, 64(3): 1514-1529.

编辑　陆燕菲