面向图片识别的深度学习模型并行优化方法

巨涛，赵宇阳，刘帅，杨阳，杨文杰
（兰州交通大学电子与信息工程学院，730070，兰州）

摘要：针对机器学习中的图片识别问题，结合已有的图片识别方法，在集群并行系统上对图片识别的并行优化方法进行研究。通过引入参数服务器机制，对分布式随机梯度下降算法中的参数更新机制进行了改进。一方面对Worker节点计算出的梯度进行稀疏化处理，以减少Worker节点和参数服务器节点之间的通信量；另一方面将参数服务器节点向Worker节点发送更新后的模型参数转换为参数服务器节点向Worker节点发送累积的梯度，然后对累积的梯度进行稀疏化处理，以进一步减少Worker节点和参数服务器节点之间的通信量。此外，为了解决由于稀疏化而引起的训练精度损失问题，引入了一种应对动量损失的动量修正方法，以提升图片识别模型的精度。实验结果表明，与基本的异步随机梯度下降算法ASGD相比，本文并行优化方法在3种不同的压缩率下，对深度学习图片识别模型的训练速度平均可提高2.95倍，测试准确率平均提高了4.6%。

关键词：深度学习；并行优化；参数服务器；图片识别
中图分类号：TP311 **文献标志码**：A
DOI：10.7652/xjtuxb202301014 **文章编号**：0253-987X(2023)01-0141-11

A Parallel Optimization Method of Deep Learning Model for Image Recognition

JU Tao, ZHAO Yuyang, LIU Shuai, YANG Yang, YANG Wenjie
(School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China)

Abstract: Aiming at the problem of image recognition in machine learning, this paper studies the parallel optimization method of image recognition on the cluster parallel system by combining the existing image recognition methods. The parameter updating mechanism of the distributed stochastic gradient descent algorithm is improved by introducing a parameter server mechanism. On the one hand, the gradient calculated by the Worker node is sparsely processed to reduce the communication load between the Worker node and the parameter server node. On the other hand, the updated model parameters sent by the parameter server node to the Worker node are converted into the accumulated gradient sent by the parameter server node to the Worker node, and then the accumulated gradient is sparsely processed to further reduce the communication load between the Worker node and the parameter server node. In addition, in order to solve the problem of training accuracy loss caused by sparsification, a momentum correction method is introduced to improve the accuracy of image recognition model. Experimental results show that, compared with the basic asyn-

收稿日期：2022-05-17。 作者简介：巨涛(1980—)，男，副教授，硕士生导师。 基金项目：国家自然科学基金资助项目(61862037)；兰州交通大学天佑创新团队资助项目(TY202002)；兰州市人才创新创业资助项目(2021-RC-40)。
网络出版时间：2022-09-09 网络出版地址：https://kns.cnki.net/kcms/detail/61.1069.T.20220909.1137.002.html

chronous stochastic gradient descent algorithm ASGD, the proposed parallel optimization method in this paper can improve the training speed of deep learning image recognition model by 2.95 times on average and the test accuracy by 4.6% on average under three different compression rates.

Keywords: deep learning; parallel optimization; parameter server; image recognition

随着智能手机以及电脑设备的普及,当今世界已产生了大量的图像信息,其中图片信息占据了很大的比重。通过深度学习训练对图片进行识别,进而获取有价值的信息是当下人工智能领域的研究热点。目前用于图片识别模型训练的数据集的大小通常为十几吉甚至几十吉,传统的面向图片识别模型的训练几乎都是在单台计算机上进行的,面对如此庞大的数据量,其训练至模型收敛往往要花费很长的时间。而且随着大数据时代的到来,深度学习训练同时面临着大数据和大模型的双重挑战,从而对设备的存储量和计算能力提出了新的要求,只依托单台计算机进行模型训练在时间上已经不能满足人们的要求[1-4]。

为了应对上述挑战,目前较普遍的解决方案是利用多台计算机进行分布式深度学习训练,依托多个计算节点的并行处理来加速深度学习的训练过程,大幅缩短训练至模型收敛所要花费的时间。其中,使用并行度较高的计算机集群完成深度学习训练成为了比较有效的处理方式[5-7]。在近几年的分布式深度学习发展中,形成了数据流式和参数服务器架构两大主流的分布式深度学习架构。在这两个主流的分布式深度学习架构中,参数服务器架构是目前使用最多的架构[8]。同时,为了加速大规模深度学习的训练,学术界已经提出了多种分布式随机梯度下降算法,其中包括:同步随机梯度下降(synchronous stochastic gradient descent,SSGD)[9-11]和异步随机梯度下降(asynchronous stochastic gradient descent,ASGD)[12-15]算法。

本文在异步随机梯度下降算法的基础上,一方面对 Worker 节点计算出的梯度进行稀疏化处理,以此减少 Worker 节点和参数服务器节点之间的通信量;另一方面是将参数服务器节点向 Worker 节点发送更新后的模型参数转换为参数服务器节点向 Worker 节点发送累积的梯度,然后对累积的梯度进行稀疏化处理,以此进一步减少 Worker 节点和参数服务器节点之间的通信量。此外,引入了一种应对动量损失的方法,以提升图片识别模型的精度。

1 相关工作

1.1 参数服务器

在深度学习训练任务中,随着训练数据集和神经网络模型的大规模增长,单台计算机的存储空间可能无法满足要求且神经网络模型的训练周期较长,这时就要考虑利用分布式深度学习将训练数据分配到多个计算机上来解决这一问题。然而,现有的一些开源分布式框架对深度学习的支持都有一定的局限性,例如 Hadoop 不适合迭代计算,Spark 只提供了浅层深度学习支持[16]。与之相比,参数服务器架构因其易于部署且具有一定的容错性和可伸缩性,已成为分布式深度学习中常用的框架[17]。此外,参数服务器架构中训练数据被分配到各个工作节点,然后各个工作节点从参数服务器节点中获取模型参数并以并行的方式训练神经网络模型。和单台计算机相比,利用参数服务器架构可以提升神经网络模型的训练速度,缩短其整体训练时间。

依据功能,参数服务器架构中的计算节点被划分为参数服务器节点和工作节点。其中,参数服务器节点负责维护神经网络模型参数,工作节点负责执行训练任务。参数服务器的架构如图 1 所示。

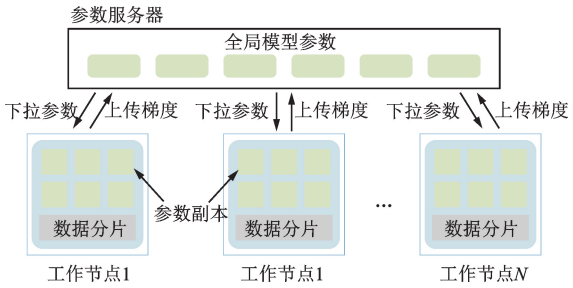

图 1 参数服务器架构

Fig. 1 Parameter server Architecture

从逻辑上把参数服务器节点和工作节点区分开,可以更加灵活地支持各种通信模式,并且便于在功能上进行扩展[18]。参数服务器架构中,参数服务器节点的个数可以是一个或多个。如果参数服务器节点的个数为 1,那么该参数服务器节点负责维护所有的神经网络模型参数。否则,各个参数服务器节点分别负责维护神经网络模型参数的一个分片。

同样地，参数服务器架构中，工作节点的数量也可以为 1 个到多个。模型并行中，如果只有 1 个工作节点，那么该工作节点在进行神经网络模型训练时，负责计算全局神经网络模型参数对应的梯度。否则，各个工作节点只需要计算各自负责的神经网络模型参数对应的梯度。数据并行中，不管有几个 Worker 节点，Worker 节点的本地有一份相同的模型副本，此种情况下，Worker 节点负责计算全局神经网络模型参数对应的梯度。神经网络模型训练过程中，各工作节点将自身计算出的梯度上传到相应的参数服务器节点，参数服务器节点根据先前设定的更新方式对模型参数进行更新。模型参数更新完毕后，当参数服务器节点接收到各个工作节点发出的获取最新的模型参数请求时，会将最新的模型参数发送给相应的工作节点。工作节点接收到最新的模型参数后，会继续进行迭代训练，直到达到了预定的训练轮数。

1.2　分布式随机梯度下降算法

1.2.1　同步随机梯度下降

同步随机梯度下降（SSGD）是最流行的分布式优化器之一，它将工作负载分配给各个工作节点，并在每轮迭代开始前，各个工作节点从参数服务器中获取相同的模型参数。该模式下，当一个工作节点完成一次迭代计算后，会通过 Push 操作将当前计算出的梯度发送至参数服务器节点。参数服务器节点在接收到所有工作节点上传的梯度后，会对梯度进行聚合平均，并将聚合平均的结果用于参数服务器内模型参数的更新，相当于使用单个工作节点进行批量更大的训练。在有 N 个工作节点的情况下，同步随机梯度下降执行以下更新

$$f(W)=\frac{1}{|D|}\sum_{x\in D}f(x,W) \tag{1}$$

$$W_{t+1}=W_t-R\frac{1}{NM}\sum_{i=1}^{N}\sum_{x\in S_{k,t}}\nabla f(x,W_t) \tag{2}$$

式中：$f(W)$ 表示损失函数；D 表示训练数据集；x 表示数据集中的一个样本；W 表示神经网络的权重参数；$f(x,W)$ 表示样本 x 对应的损失；R 表示学习率；N 表示工作节点总数；M 表示一个 mini-batch 的大小；$S_{k,t}$ 表示第 t 次迭代时，从 D 中采样的 N 个 mini-batch 序列。

上述参数更新机制使得 SSGD 能够保证模型的收敛性，但是也存在一定的缺陷，即参数服务器必须等待训练速度最慢的工作节点上传完所计算出的梯度后，才能进行模型参数更新。在参数服务器等待的过程中，其他已完成梯度上传的工作节点也在等待，这造成了一定的计算资源的浪费。同步随机梯度下降法各工作节点计算和通信过程如图 2 所示。

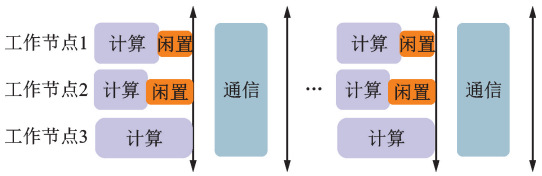

图 2　同步随机梯度下降法计算和通信过程

Fig. 2　The calculation and communication process of synchronous stochastic gradient descent method

1.2.2　异步随机梯度下降

基于同步随机梯度下降的分布式训练可能会出现某些工作节点滞后，从而可能会降低执行效率和并行可扩展性。为了解决这一问题，异步随机梯度下降（ASGD）被提出用以消除工作节点之间的同步障碍。和同步随机梯度下降相比，异步随机梯度下降中的单个工作节点在计算出本次迭代的全部梯度后，会立即将梯度上传到参数服务器节点。此时参数服务器节点不用等待其他工作节点完成梯度上传操作，直接利用该工作节点上传的梯度对模型参数进行更新，并将更新后的模型参数发送给该工作节点。由于工作节点之间不再存在同步障碍，异步随机梯度下降可以显著加快分布式训练的进程。此外，和同步随机梯度下降相比，异步随机梯度下降法可以更加充分地利用计算资源。异步随机梯度下降法中的各节点计算和通信过程如图 3 所示。异步随机梯度下降中的参数更新方式如下式

$$W_{t+1}=W_t-R\frac{1}{M}\sum_{x\in M}\nabla f(x,W_t) \tag{3}$$

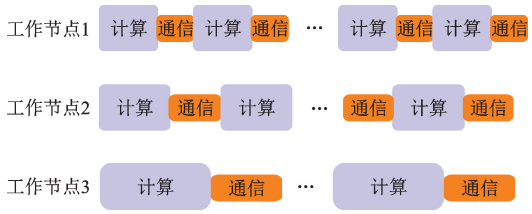

图 3　异步随机梯度下降法计算和通信过程

Fig. 3　The calculation and communication process of asynchronous stochastic gradient descent method

2　基于 ASGD 的梯度稀疏化并行优化方法

异步随机梯度下降中，虽然通过分布式方式增

加训练节点的数量并利用数据并行可以显著减少在相同大小训练数据上的前向传播和后向传播的总计算时间,但是将梯度从 Worker 节点上传至参数服务器节点的时间成本是很高的。相比之下,通过数据并行节省的计算时间就显得微不足道[19-20]。因此,数据通信所产生的时间开销就成为扩展分布式训练的重要瓶颈。同时,分布式随机梯度下降中的大多数梯度交换都是冗余的,如果将迭代计算出的所有梯度全部上传,则增加了很多不必要的通信量。已提出的用于优化 Worker 节点和参数服务器节点之间的通信的算法,例如 DGC 等优化算法[21]往往是针对同步随机梯度下降的,具有一定的局限性。为了改善这一问题,本文提出了一种基于 ASGD 的梯度稀疏化并行优化方法(GS-ASGD)。通过对 Worker 节点计算出的梯度进行稀疏化处理,和将参数服务器节点向 Worker 节点发送模型参数转换为参数服务器节点向 Worker 节点发送稀疏化后的累积梯度两方面,减少 Worker 节点和参数服务器节点之间的通信量[22];同时为了解决由于稀疏化而引起的训练精度损失问题,引入应对动量损失的动量修正方法,以提升图片识别模型的精度。最终将该并行优化方法用于图片识别模型的训练中,以减少图片识别模型训练过程中各个 Worker 节点和参数服务器节点间的通信量,提升图片识别模型的训练速度和精度。

2.1 梯度稀疏化

对于单个 Worker 节点,异步随机梯度下降算法在完成一次迭代训练后会将本次迭代计算出的梯度全部上传到参数服务器。分布式随机梯度下降中的大多数梯度交换都是冗余的,即梯度中的部分元素对模型参数的更新并没有产生较大的影响。因此,可以通过梯度稀疏化的方法,仅将那些对模型参数更新能产生较大影响的梯度元素,上传至参数服务器节点,参与模型参数的更新,从而减少 Worker 节点和服务器节点之间的数据通信量,加速模型训练速度。

要实现梯度稀疏化,首先要确定哪些梯度元素能够对模型参数的更新产生较大的影响。梯度是由多个变量的偏导数汇总而成的张量,梯度越大,模型参数变化的越快。因此,那些绝对值较大的梯度元素会对模型参数的更新产生较大的影响。所以通过设定梯度压缩率,找出 Worker 节点中能对模型参数更新产生较大影响的梯度元素,上传至参数服务器节点,参与模型参数的更新,就可以实现梯度稀疏化。基于以上分析,梯度稀疏化算法具体设计如下。

(1)基于分布式随机梯度下降中的大多数梯度交换都是冗余的特点,通过设定梯度压缩率来减少 Worker 节点需要发送到参数服务器节点的梯度个数。梯度压缩率是指梯度压缩后的元素个数和梯度压缩之前的元素个数之比。梯度压缩率决定了 Worker 节点向参数服务器发送的梯度个数,梯度压缩率值越大说明向参数服务器发送的梯度个数越多,则对应的传输开销就越大,模型的训练时间就越长;相反,梯度压缩率值越小,则向参数服务器发送的梯度个数越少,则对应的传输开销就越小,从而可以加速模型的训练速度,但同时梯度压缩率值越小,则参与模型参数更新的梯度数越少,会由于动量损失而降低模型求解的精度,所以本文梯度压缩率在1%~20%范围内,模型的求解速度和求解精度都达到了一个较好的平衡,如果小于1%则就会因动量损失引起模型求解精度的降低,同时若高于20%则会因传输开销过大而降低模型的求解速度。为了更明显的对比分析 Worker 节点将不同数目的梯度发送到参数服务器节点,对图片识别模型训练速度的影响,先后设定了3种梯度压缩率,分别为1%、10%、20%。

(2)遍历卷积神经网络的每一层,并通过梯度压缩率确定每一层需要发送至参数服务器节点的梯度个数 n。卷积神经网络中每一层的梯度个数与梯度压缩率的乘积即是 n。

(3)首先对每一层中的梯度值取绝对值,然后将各个梯度值的绝对值作为 torch 库中的 topk()函数的输入张量,并将最大值(largest)参数设置为 True,通过 topk()函数得到前 n 大的绝对值元素,选择将前 n 大的绝对值元素对应的梯度发送到参数服务器节点。torch 库是 PyTorch 深度学习框架的重要组成部分,torch 库包含了用于多维张量的数据结构,并定义了对这些张量的数学运算。同时,它还提供了许多用于高效序列化张量和任意类型的实用程序。torch 库中的 topk()函数的功能是沿给定维度返回输入张量中前 n 大的值或者后 n 小的值,当 topk()的 largest 参数值为 True 时,topk()函数返回输入张量中前 n 大的值,当 topk()的 largest 参数值为 False 时,topk()函数返回输入张量中后 n 小的值。

(4)对神经网络的每一层,将第(3)步中得到的前 n 大的绝对值元素中的最小值作为阈值,将绝对值小于阈值的梯度元素存储在本地,并将其分别加

到下一次迭代计算出的相应的梯度元素上，待其累积值的绝对值大于相应的阈值时再将其发送到参数服务器节点，这样做的目的是充分利用 Worker 节点计算出的梯度。这里的阈值其实就是临界值，设定阈值的目的是便于将每一层中绝对值小于阈值的梯度筛选出来。

(5)通过上述操作，可以设定不同的梯度压缩率来控制 Worker 节点在一次迭代训练结束后需要上传到参数服务器节点的梯度的个数，以此来减少 Worker 节点和参数服务器节点之间的通信量，提升图片识别模型的训练速度。

2.2　模型参数稀疏化

参数服务器架构中的 Worker 节点和参数服务器节点之间的通信是双向的，即 Worker 节点需要将通过迭代训练计算出的梯度上传到参数服务器节点，参数服务器节点需要利用 Worker 节点上传的梯度更新本地的模型参数，并将更新后的模型参数发送给该 Worker 节点。所以从参数服务器节点将更新后的模型参数发送到 Worker 节点这一层面对其二者之间的通信作进一步的优化，缩短其通信时间，从而进一步加速图片识别模型的训练过程。

基于参数服务器架构的 ASGD 中，单个 Worker 节点在计算出本次迭代的全部梯度后，会立即将计算出的梯度上传到参数服务器节点。参数服务器节点会直接利用该 Worker 节点上传的梯度，对模型参数进行更新，并将更新后的模型参数发送给该 Worker 节点。然而，在该 Worker 节点最近一次从参数服务器节点获取模型参数开始，至其利用获取的模型参数完成一次迭代训练，并将自身计算出的梯度上传至参数服务器节点这段时间内，参数服务器节点中的模型参数可能被更新数次。其中，每次被更新时参数的变化量是某一 Worker 节点上传的梯度与学习率的乘积。因此，对于参数服务器架构下的同一 Worker 节点，在其最近一次从参数服务器节点获取模型参数开始，至其利用获取的模型参数完成一次迭代训练并将自身计算出的梯度上传至参数服务器节点这段时间内，参数服务器节点对接收到的梯度进行累积，梯度累积值与学习率的乘积就是该 Worker 节点本地模型参数的变化量。因此只需要用该 Worker 节点本地的模型参数减去上述参数服务器节点获取到的梯度累积值与学习率的乘积，就可得到该 Worker 节点本地更新后的模型参数。所以参数服务器只需要向 Worker 节点发送累积梯度，而不需要发送所有的模型参数，从而可以进一步减少从参数服务器节点到 Worker 节点之间的数据通信量。

2.3　动量修正

上述梯度稀疏化过程中，取前 n 大的绝对值元素中的最小值作为阈值，将小于阈值的梯度元素存储在本地并将其分别加到下一次迭代计算出的相应的梯度元素上，待其累积梯度大于相应的阈值时再将其发送到参数服务器节点。由于将小于阈值的梯度元素在本地进行累积，这导致损失了部分动量，从而会导致模型训练精度降低。所以在异步随机梯度下降算法的基础上进一步提出了一种动量修正策略，以此消除动量损失对模型训练精度的干扰。

2.3.1　带动量的 SGD 工作原理分析

在普通的 SGD 中，模型参数每一次的更新量是 $r\nabla f(x,W_t)$，其中 $\nabla f(x,W_t)$ 为损失函数对 x 的一阶导数，即梯度。普通的 SGD 的缺点是其更新方向完全依赖于当前 batch 计算出的梯度，因此十分不稳定。此外，通过对 Worker 节点计算出的梯度进行压缩，虽然可以在一定程度上加速图片识别模型的训练进程，但由于只是将原本梯度中的一部分上传至参数服务器节点，因此会对图片识别模型的训练精度造成干扰。

对于普通的 SGD，梯度计算与模型参数更新如下式

$$g=\frac{1}{M}\sum_{x\in M}\nabla f(x,W_t) \tag{4}$$

$$W_{t+1}=W_t-Rg \tag{5}$$

式中 g 代表梯度。

普通的 SGD 中，每次进行梯度更新时，都会更新 $-Rg$。对于普通的 SGD，其更新值的方差较大，会导致收敛过程产生波动，可能陷入极小值。动量[16]常被用于深度学习训练，它可以提供一个显著的优化提升。相对于普通的 SGD，带动量的 SGD 采用了不同的学习率更新方式。与普通的 SGD 不同的是在计算梯度的地方，当前时刻的梯度是从开始时刻到当前时刻的梯度指数加权平均，并将这个梯度的指数加权值称为速度 μ。μ 既有大小也有方向，且在每次更新参数时会有一定的衰减。

对于带动量的 SGD，梯度计算与模型参数更新如下式

$$g=\frac{1}{M}\sum_{x\in M}\nabla f(x,W_t) \tag{6}$$

$$\mu_t=m\mu_{t-1}+Rg \tag{7}$$

$$W_{t+1}=W_t-\mu_t \tag{8}$$

式中：m 为动量系数，表示要在多大程度上保留原来的更新方向；R 表示当前 batch 的梯度多大程度上影响最终的更新方向。

第 1 次迭代后，参数更新如下式

$$\mu_1 = Rg_1 \tag{9}$$

$$W_2 = W_1 - \mu_1 = W_1 - Rg_1 \tag{10}$$

第 2 次迭代后，参数更新如下式

$$W_3 = W_2 - \mu_2 = W_2 - m\mu_1 - Rg_2 = W_2 - mRg_1 - Rg_2 = W_2 - R(mg_1 + g_2) \tag{11}$$

由上述式(9)～(11)推导可知，如果本次的梯度方向和上次的梯度方向相同，那么这次下降的幅度就会增大，进而使得模型的收敛速度加快；如果本次和上次的梯度方向是相反的，那么这次就和上次相互抑制，减缓振荡，从而解决了 SGD 波动大的问题。同时，由于有动量的作用，在局部最优点时，其可以借助动量跳出来，不易陷入局部最优。

综上所述，带动量的 SGD 的思想是引入一个积攒历史梯度信息的动量来加速 SGD。进行模型参数更新时，在一定程度上保留之前更新的方向，同时利用当前 batch 的梯度进行微调，以得到最终的更新方向，能在一定程度上摆脱局部最优，从而提升模型的求解精度。

2.3.2 动量修正策略

在基于参数服务器架构的动量 ASGD 中，参数服务器节点收集梯度并更新动量和模型参数，如下式

$$\mu_t = m\mu_{t-1} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_t) \tag{12}$$

$$W_{t+1} = W_t - \mu_t \tag{13}$$

式中：μ_t 是速度；$\frac{1}{M}\sum_{x\in M} f(x, W_t)$ 为单个 Worker 节点通过一次迭代训练计算出的梯度。然而，通过梯度稀疏化操作，原来的基于参数服务器架构的动量 ASGD 将会被进一步改变为基于参数服务器架构的稀疏化动量 ASGD，其更新公式如下

$$v_{k,t} = v_{k,t-1} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_t) \tag{14}$$

$$\mu_t = m\mu_{t-1} + \mathrm{sparse}(v_{k,t}) \tag{15}$$

$$v_{k,t} = \mathrm{unsparse}(v_{k,t}) \tag{16}$$

$$W_{t+1} = W_t - \mu_t \tag{17}$$

式(14)表示 Worker 节点 k 将第 t 次迭代计算出的梯度与学习率相乘，并将二者的乘积分别加到之前存储在本地的相应的梯度与学习率乘积的累积值上。式(15)中，对 $v_{k,t}$ 进行 sparse(vk,t) 稀疏化处理，将绝对值大于阈值 d 的 $v_{k,t}$ 发送到参数服务器节点，以参与 μ_t 的更新。unsparse($v_{k,t}$) 表示绝对值小于阈值的 $v_{k,t}$，式(16)表示将绝对值小于阈值的 $v_{k,t}$ 存储在 Worker 节点 k 的本地。式(17)表示对模型参数进行更新。

由于式(16)中的 $v_{k,t}$ 绝对值小于阈值而被存储在 Worker 节点 k 的本地，未参与式(15)中的动量更新，这导致了部分动量的消失，从而对图片识别模型的收敛性及精度造成了一定的干扰。为了消除上述干扰，本文引入了一种稀疏化感知动量进行动量修正，以实现和式(12)与(13)等价。

稀疏化感知动量修正策略算法思想如下。

(1)基于 GS-ASGD 中的梯度压缩策略，在 Worker 节点本地，不对计算出的梯度进行累积，而是将计算出的梯度直接用于动量更新，把动量更新的结果 μ 视为梯度，并将绝对值大于阈值的 μ 值发送到参数服务器节点，以参与模型参数的更新。此外，在 Worker 节点的本地，并不将绝对值大于阈值的那部分 μ 清零。

(2)对于绝对值小于阈值的 μ，则在 Worker 节点本地，将其放大为原来的 $1/m$ 倍。因为在每次迭代之后，剩余的动量值比被发送到参数服务器节点的动量值重要，因此放大未发送的动量值是合理的。稀疏化感知动量修正过程中动量的表达式如下

$$\mu_{k,t} = \begin{cases} m\mu_{k,t-1} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_t), & \mu_{k,t} > d \\ \left(m\mu_{k,t-1} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_t)\right)\frac{1}{m}, & \mu_{k,t} < d \end{cases} \tag{18}$$

下面，解释将小于阈值的动量放大 $1/m$ 倍的原因。用 L 表示两次迭代之间稀疏更新间隔的长度，如下式

$$\mu_{t+L} = m\mu_{t+L-1} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+L}) = m\left(\left(m\mu_{t+L-2} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+L-1})\right)\frac{1}{m}\right) + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+L}) = m\mu_{t+L-2} + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+L-1}) + R\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+L}) = m\mu_t + R\sum_{j=1}^{L}\frac{1}{M}\sum_{x\in M}\nabla f(x, W_{t+j}) \tag{19}$$

基于参数服务器架构的动量 ASGD 中，如果忽略其他节点的干扰，则经过 L 次迭代训练，μ_t 的表达式如下

$$\mu_{t+L}=m\mu_{t+L-1}+R\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L})=$$
$$m\Big(m\mu_{t+L-2}+R\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L-1})\Big)+$$
$$R\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L})=$$
$$m^{L}\mu_{t}+R[m^{L-1}\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+1})+$$
$$m^{L-2}\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+2})+\cdots+$$
$$\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L})] \tag{20}$$

基于参数服务器架构的稀疏化动量 ASGD 中，在稀疏更新间隔 L 内，μ_t 会被累积，其表达式如下

$$\mu_{t+L}=m^{L}\mu_{t}+R[\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+1})+$$
$$\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+2})+\cdots+\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L})] \tag{21}$$

随着 L 的增大，m^L 会趋近于 0，进而导致 $m^L\mu_t$ 这一项趋近于 0。因此，μ_{t+L} 的表达式如下

$$\mu_{t+L}=R[\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+1})+$$
$$\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+2})+\cdots+\frac{1}{M}\sum_{x\in M}\nabla f(x,W_{t+L})] \tag{22}$$

当 $L=1$ 时，式(19)等价于式(20)，这表明，在稀疏化场景下引入稀疏化感知动量，对于基于参数服务器架构的 ASGD 来说，其等价于基于参数服务器架构的动量 ASGD。如果 $L>1$，式(19)可以被看作是一个大的动量更新。由式(19)可知，通过将小于阈值的动量放大 $1/m$ 倍，稀疏化感知动量在稀疏更新间隔 L 内只用 $\mu_{k,t}$ 乘以 m 一次。当 L 增大时，和式(21)中的 $m^L\mu_t$ 相比，$m\mu_t$ 不会趋近于 0，即式(19)中的动量项不会消失。因此，在稀疏化场景下，通过引入稀疏化感知动量，可以保证基于参数服务器架构的 ASGD 中的动量不会消失。

2.4　图片识别并行优化方法处理流程

2.4.1　系统初始化模块处理流程

系统初始化模块负责 Worker 节点和参数服务器节点上的图片识别数据集、神经网络模型参数、梯度压缩率 r 及相关配置信息的初始化。具体处理流程如下。

(1)本算法采用数据并行方式进行分布式训练[23]，参数服务器节点根据 Worker 节点的个数对数据集进行平均划分，使得各个 Worker 节点分别获得其中一份。若事先已经将图片识别数据集存放于 Worker 节点和参数服务器中的相应目录下，则参数服务器节点只需根据 Worker 节点的个数对数据集进行平均划分。

(2)参数服务器节点依据所用的神经网络模型初始化模型参数。

(3)各 Worker 节点异步的从参数服务器节点获取步骤(2)中已经被初始化的模型参数作为本地的模型参数。

2.4.2　分布式并行训练模块处理流程

分布式并行训练模块中，各 Worker 节点利用获取到的模型参数及训练子集进行图片识别模型的训练。具体处理流程如下。

(1)各 Worker 节点首先检查本地是否有待训练的训练子集，如果有则从中取出一个批次的训练数据，接着转到步骤(2)；否则，说明本地所有的训练子集均被训练完，结束训练。

(2)Worker 节点利用一个批次的训练数据和本地模型参数进行图片识别模型训练，并通过反向传播算法计算各个模型参数对应的梯度；依据每一层计算出的梯度个数及设定的 r，确定需要发送到参数服务器节点的梯度个数 n；将每一层中计算出的梯度分别用于动量更新，并把通过动量更新得到的各个 μ 视为梯度，对各个 μ 取绝对值，并把各个 μ 的绝对值作为 topk()函数的输入张量，通过 topk()函数得到前 n 大的绝对值元素，选择前 n 大的绝对值元素中的最小值作为阈值；将绝对值大于阈值的 μ 发送到参数服务器节点，将绝对值小于阈值的 μ 放大为原来的 $1/m$ 倍；最终，将绝对值大于阈值的 μ 和上述放大为原来的 $1/m$ 倍的 μ 存储在 Worker 节点的本地，以供后续的迭代训练进行动量更新使用。

(3)参数服务器节点对接收到的用于更新模型参数的 μ 进行累积；对于每一层，依据 r 及对应的不同 μ 累积值的个数，确定需要发送给 Worker 节点的不同 μ 累积值的个数 n；将不同 μ 累积值的绝对值作为 topk()函数的输入张量，通过 topk()函数得到前 n 大的绝对值元素，选择将前 n 大的绝对值元素中的最小值作为阈值；将绝对值大于阈值的 μ 累积值发送给 Worker 节点，将绝对值小于阈值的 μ 累积值在参数服务器节点的本地进行存储，以供后续的 μ 累积使用。

(4)Worker 节点利用接收到的 μ 累积值，对本地的模型参数进行更新，然后转到步骤(1)。

3 图片识别实验

3.1 实验环境

3.1.1 硬件环境

本实验的分布式环境是通过 3 台 Linux 主机搭建的，其中的 1 台作为参数服务器节点，另外两台作为 Worker 节点，Worker 节点和参数服务器节点之间通过 2 Gb/s 的以太网进行通信。其中，每台机器配备了 1 个 CPU 及 1 块 GPU 卡，硬件的具体配置：CPU 为 Intel Xeon Platinum 82 系列(80vCPU) v6 @2.5 GHz，内存 8 GB，显卡为 NVIDIA Tesla V100-SXM2，显存为 32 GB。

3.1.2 软件环境

本文算法通过 PyTorch 1.2.0 实现，参数服务器架构通过 PyTorch 1.2.0 分布式 API 和 gloo 后端实现。操作系统采用 CentOS Linux release 7.8.2003 (Core)版本，PyTorch 版本为 1.2.0，Python 版本为 3.7，CUDA 版本为 9.0，cuDNN 版本为 7.6.5。

3.1.3 实验数据集及网络模型

本文所用的数据集是通常被用于识别普适物体的 CIFAR-10[24] 图片识别数据集，是机器学习和深度学习研究中被用到的最广泛的数据集之一。该数据集由 10 个类别的 60 000 张 32×32 像素的彩色图片组成，其中每个类别包含 6 000 张图片。这 10 个类别分别是卡车、飞机、鸟类、鹿、蛙类、汽车、马、猫、狗和船。总体上包含 50 000 张训练图片和 10 000张测试图片。该数据集共分为 5 个训练批次和 1 个测试批次，其中每个批次包含 10 000 张图片。测试批次包含来自不同类别的 1 000 张随机选择的图片。训练批次包含随机顺序的剩余图片，但有些训练批次中不同类别的图片的张数可能会不同。在它们之间，一个训练批次包含来自不同类别的 5 000 张图片。基于通用性考虑，实验用到的神经网络模型为基础的 AlexNet[25] 网络模型。

3.1.4 实验参数

本文将图片识别模型训练的初始学习率设置为 0.1，再利用 PyTorch 中动态调整学习率的函数 torch.optim.lr_scheduler.MultiStepLR()，在第 30 个 Epoch 时，将学习率调整为 0.01，在第 40 个 Epoch 时，将学习率调整为 0.001。此外，batch_size 的大小为 64，用于测试的 test_batch_size 的大小为 10 000，Epoch 大小为 50。

3.2 实验结果与分析

3.2.1 实验结果

实验依托参数服务器架构及基于 ASGD 的梯度稀疏化并行优化方法 GS-ASGD，对图片识别网络模型进行分布式并行训练。

3 种梯度压缩率下，AlexNet 在 CIFAR-10 训练集上的训练损失值随时间的变化如图 4 所示。可以看出，随着训练时间的增加，3 种梯度压缩率下的训练损失值逐渐下降，在下降过程中都明显出现了拐点并迅速下降，然后趋于稳定，这是由于在第 30 和第 40 个 Epoch 时，将学习率调整为先前的 0.1 倍的原因。通过学习率调整策略，训练损失值迅速下降，取得了不错的效果。从 AlexNet 神经网络模型整体训练时间上看，当 r 为 0.01 时，共花费了 918 s 完成训练；当 r 为 0.10 时，共花费了 1 328 s；当 r 为 0.20 时，共花费了 1 553 s。

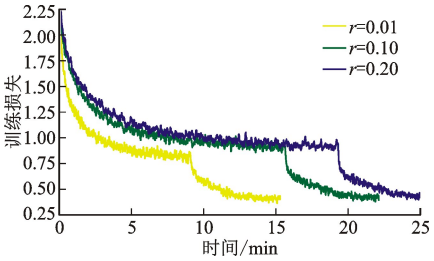

图 4 不同梯度压缩率下 GS-ASGD 的训练损失值随时间的变化

Fig. 4 The curve of training loss of GS-ASGD under different compression ratio

3 种梯度压缩率下，AlexNet 在 CIFAR-10 测试集上的图片识别准确率随时间变化的曲线如图 5 所示。

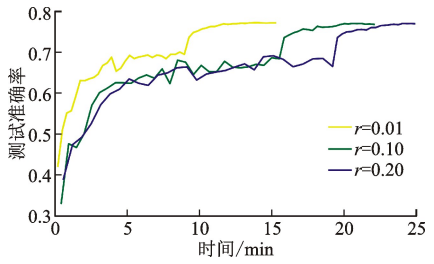

图 5 不同梯度压缩率下 GS-ASGD 的识别准确率随时间的变化

Fig. 5 The recognition accuracy curve of GS-ASGD under different compression ratio

从图 5 可看出，在训练开始后，测试集上的图片识别准确率逐渐增大，同样受到学习率调整策略

的影响，分别在第 9、16、19 min 附近出现拐点，然后稳步增大并最终趋于稳定。r 为 0.01 时，测试准确率最终达到了 77.12%，r 为 0.10 和 0.20 时，分别达到了 76.76%和 76.89%。

3.2.2　实验对比分析

分别从训练损失、测试准确率和训练耗时 3 个方面，将本文提出的基于 ASGD 的梯度稀疏化并行优化训练算法和 ASGD 算法两者的实验结果进行对比分析。其中，训练损失随时间的变化如图 6 所示，测试集的识别准确率随时间的变化如图 7 所示。

ASGD 算法和 GS-ASGD 算法的性能对比实验结果如表 3 所示。从表 3 可知，GS-ASGD 算法在训练损失、测试准确率和训练耗时这 3 个评价指标上的结果均优于 ASGD 算法，尤其是在测试准确率和训练耗时方面表现突出。当梯度压缩率为 1%时，GS-ASGD 算法仅花费了 15.3 min 就完成了图片识别模型的训练，ASGD 算法花费了 59.4 min，其图片识别模型的训练速度是 ASGD 算法的 3.88 倍。当梯度压缩率为 10%、20%时，图片识别模型的训练速度分别达到了 ASGD 算法的 2.68 倍、2.29倍。3 种压缩率下，图片模型的训练速度平均提高了 2.95 倍。此外，和 ASGD 算法相比，GS-ASGD 算法的测试准确率没有下降且有一定的提升，3 种压缩率下测试准确率平均提高了 4.6%。这说明引入的稀疏化动量修正策略可有效解决由于稀疏化而引起的训练精度损失问题，同时对图片识别模型的识别精度有一定的提升作用。

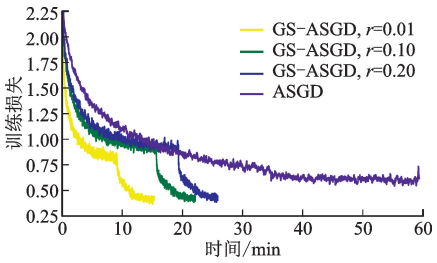

图 6　GS-ASGD 和 ASGD 的训练损失值变化对比

Fig. 6　Comparison of training loss between GS-ASGD and ASGD

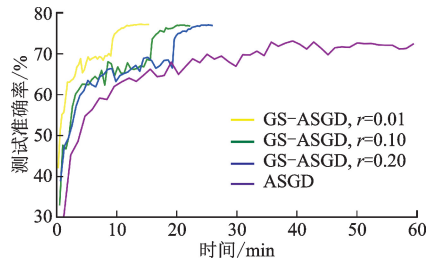

图 7　GS-ASGD 和 ASGD 的测试准确率变化对比

Fig. 7　Comparison of recognition accuracy between GS-ASGD and ASGD

表 3　ASGD 和 GS-ASGD 算法性能对比

Table 3　Performance comparison of ASGD and GS-ASGD algorithms

算法	梯度压缩率/%	训练损失	测试准确率/%	训练耗时/min
ASGD		0.474 8	72.32	59.40
GS-ASGD	1	0.354 8	77.12	15.30
	10	0.386 7	76.76	22.13
	20	0.427 9	76.89	25.88

4　结　论

本文针对机器学习中图片识别的问题，在集群并行系统上对图片识别进行了优化，主要基于参数服务器机制，一方面是对 Worker 节点计算出的梯度进行稀疏化处理，以此减少 Worker 节点和参数服务器节点之间的通信量；另一方面是将参数服务器节点向 Worker 节点发送更新后的模型参数转换为参数服务器节点向 Worker 节点发送累积的梯度，然后对累积的梯度进行稀疏化处理，进一步减少 Worker 节点和参数服务器节点之间的通信量；同时引入动量修正策略，在进行模型参数更新时，一定程度上保留之前更新的方向，同时利用当前 batch 的梯度进行微调以得到最终的更新方向，摆脱局部最优，从而在一定程度上提高模型的求解精度。通过对 Worker 节点和参数服务器节点之间的通信及动量传递进行优化，最终提高了图片识别模型的训练速度及精度。对比实验结果表明，和基本的 ASGD 算法相比，本文所提方法在 AlexNet 模型和 CIFAR-10 数据集上对深度学习图片识别模型的训练速度平均提高 2.95 倍，可大幅度缩短模型的训练时间，同时测试准确率平均提高了 4.6%。

参考文献：

[1] MOGHADAM H A, DIMITRIJEV S, HAN Jisheng, et al. Transient-current method for measurement of active near-interface oxide traps in 4H-SiC MOS capacitors and MOSFETs [J]. IEEE Transactions on Electron Devices, 2015, 62(8): 2670-2674.

[2] HUANG S J, SHIH K R. Short-term load forecasting via ARMA model identification including non-Gaussian process considerations [J]. IEEE Transactions on Power Systems, 2003, 18(2): 673-679.

[3] 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)：98-115.
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): 98-115.

[4] 朱虎明，李佩，焦李成，等. 深度神经网络并行化研究综述 [J]. 计算机学报，2018，41(8)：1861-1881.
ZHU Huming, LI Pei, JIAO Licheng, et al. Review of parallel deep neural network [J]. Chinese Journal of Computers, 2018, 41(8): 1861-1881.

[5] BAO Fenye, CHEN I R. Trust management for the internet of things and its application to service composition [C]//2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). Piscataway, NJ, USA: IEEE, 2012: 1-6.

[6] KIM Y, CHOI H, LEE J, et al. Efficient large-scale deep learning framework for heterogeneous multi-GPU cluster [C]//2019 IEEE 4th International Workshops on Foundations and Applications of Self * Systems (FAS * W). Piscataway, NJ, USA: IEEE, 2019: 176-181.

[7] SHEN Jian, ZHOU Tianqi, WANG Kun, et al. Artificial intelligence inspired multi-dimensional traffic control for heterogeneous networks [J]. IEEE Network, 2018, 32(6): 84-91.

[8] 和新树. 分布式机器学习集群系统性能优化[D]. 成都：电子科技大学，2020.

[9] COATES A, HUVAL B, WANG Tao, et al. Deep learning with COTS HPC systems [C]//Proceedings of the 30th International Conference on International Conference on Machine Learning. New York, NY, USA: ACM, 2013: III-1337-III-1345.

[10] JIA Zhihao, ZAHARIA M, AIKEN A. Beyond data and model parallelism for deep neural networks [J/OL]. arXiv, 2018 [2021-06-10]. DOI: 10.48550/arXiv.1807.05358.

[11] STROM N. Scalable distributed DNN training using commodity GPU cloud computing [C]//Proceedings of the Interspeech 2015. Baixas, France: ISCA, 2015: 1488-1492.

[12] KEUPER J, PFREUNDT F J. Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms [C]//Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments. New York, NY, USA: Association for Computing Machinery, 2015: 1.

[13] NIU Feng, RECHT B, RE C, et al. HOGWILD! A lock-free approach to parallelizing stochastic gradient descent [C]//Proceedings of the 24th International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc., 2011: 693-701.

[14] TSITSIKLIS J, BERTSEKAS D, ATHANS M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms [J]. IEEE Transactions on Automatic Control, 1986, 31(9): 803-812.

[15] ZHANG Shanshan, ZHANG Ce, YOU Zhao, et al. Asynchronous stochastic gradient descent for DNN training [C]//2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Piscataway, NJ, USA: IEEE, 2013: 6660-6663.

[16] 王思远. 分布式深度学习中参数交换优化机制研究 [D]. 武汉：华中科技大学，2015.

[17] LI Mu, ANDERSEN D G, PARK J W, et al. Scaling distributed machine learning with the parameter server [C]//Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation. New York, USA: USENIX Association, 2014: 583-598.

[18] KONG Weicong, DONG Zhaoyang, HILL D J, et al. Short-term residential load forecasting based on resident behaviour learning [J]. IEEE Transactions on Power Systems, 2018, 33(1): 1087-1088.

[19] WEN Wei, XU Cong, YAN Feng, et al. TernGrad: ternary gradients to reduce communication in distributed deep learning [C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc., 2017: 1508-1518.

[20] 巨涛，张兴军，陈衡，等. 面向众核系统的线程分组映射方法 [J]. 西安交通大学学报，2016，50(10)：57-63.
JU Tao, ZHANG Xingjun, CHEN Heng, et al. A grouping mapping mechanism of threads on many-core

systems [J]. Journal of Xi'an Jiaotong University, 2016, 50(10): 57-63.

[21] LIN Yujun, HAN Song, MAO Huizi, et al. Deep gradient compression: reducing the communication bandwidth for distributed training [J/OL]. arXiv, 2017 [2020-08-01]. DOI: 10.48550/arXiv.1712.01887.

[22] 王龙翔，张兴军，朱国峰，等. 重复数据删除中的无向图遍历分组预测方法 [J]. 西安交通大学学报，2013，47(10)：51-56.
WANG Longxiang, ZHANG Xingjun, ZHU Guofeng, et al. A grouping prediction method based on undirected graph traversal in de-duplication system [J]. Journal of Xi'an Jiaotong University, 2013, 47(10): 51-56.

[23] YANG Bowen, ZHANG Jian, LI J, et al. PipeMare: asynchronous pipeline parallel DNN training [J/OL]. arXiv, 2020 [2021-08-01]. DOI: 10.48550/arXiv.1910.05124.

[24] KRIZHEVSKY A. Learning multiple layers of features from tiny images [EB/OL]. (2009-04-08)[2021-09-01]. https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=2AE323C0FD8D6B85D5BA8229315A2F2F?doi=10.1.1.222.9220&rep=rep1&type=pdf.

[25] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks [C]//Proceedings of the 25th International Conference on Neural Information Processing Systems: Volume 1. Red Hook, NY, USA: Curran Associates Inc., 2012: 1097-1105.

（编辑　武红江　亢列梅）