资源受限边缘计算系统中的自适应联合学习

王世强，蒂凡尼·托尔，西奥多罗斯·萨洛尼迪斯，梁建K，

马卡娅、何婷、陈凯文

*摘要*-新兴技术和应用，包括物联网（IoT）、社交网络和众包在网络边缘产生大量数据。机器学习模型通常是根据收集到的数据建立起来的，以便能够检测、分类和预测未来的事件。由于带宽、存储和隐私问题，将所有数据发送到集中位置通常是不切实际的。在本文中，我们考虑从分布在多个边缘节点的数据中学习模型参数的问题，而不需要将原始数据发送到一个集中的地方。我们的重点是使用基于梯度下降的方法训练的一类通用机器学习模型。从理论上分析了分布式梯度下降算法的收敛界，并在此基础上提出了一种在给定资源预算下，在局部更新和全局参数聚合之间进行最佳折衷以使损失函数最小化的控制算法。通过对实际数据集的大量实验，在网络化原型系统和更大规模的仿真环境中对该算法的性能进行了评估。实验结果表明，在不同的机器学习模型和不同的数据分布下，我们提出的方法性能接近最优。

*索引术语*-分布式机器学习、联合学习、移动边缘计算、无线网络

一、 简介

物联网（IoT）和社交网络应用的快速发展导致网络边缘产生的数据呈指数级增长。据预测，在不久的将来，数据生成速率将超过当今互联网的容量[2]。由于网络带宽和数据隐私问题，将所有数据发送到远程云是不切实际的，而且通常是不必要的。因此，研究机构估计超过90%的数据将在本地存储和处理[3]。全球协调的本地数据存储和处理成为可能

S、 Wang（通讯作者）、T.Salonidis和C.Makaya在美国纽约约克敦高地的IBM T.J.Watson研究中心工作。电子邮件：wangshiq@us.ibm.com, tsaloni@us.ibm.com, chrismak@ieee.org

T、 托尔和梁家辉在英国伦敦帝国理工学院工作。电子邮箱：tiffany.tuor14@imperial.ac.uk, 梁健@imperial.ac.uk

T、 他在宾夕法尼亚州立大学，宾夕法尼亚大学公园，美国。

电子邮箱：t。he@cse.psu.edu

K、 Chan在美国马里兰州阿德尔菲陆军研究实验室工作。电子邮件：

凯文·s。陈先生@邮件.mil

这项研究由美国陆军研究实验室和英国国防部根据W911NF-16-3-001协议号赞助。本文件中的观点和结论是作者的观点，不应被解释为代表美国军事研究实验室、美国政府、英国国防部或英国政府的官方政策。美国政府和英国政府有权复制和分发政府的转载，尽管此处有任何版权说明。

这项工作的初步版本名为“当边缘满足学习：资源受限分布式机器学习的自适应控制”在IEEE INFOCOM 2018上提出[1]。

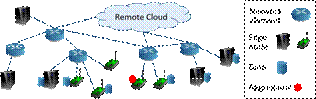


图1：系统架构。

移动边缘计算技术（MEC）[4]，[5]，其中边缘节点，如传感器、家庭网关、微型服务器和小型单元，具有存储和计算能力。多个边缘节点与远程云协同工作以执行大规模的分布式任务，这些任务涉及本地处理和远程协调/执行。

为了分析大量的数据并获得有用的信息，以便对未来事件进行检测、分类和预测，通常采用机器学习技术。机器学习的定义非常广泛，从线性回归的简单数据摘要到支持向量机（SVM）和深层神经网络的多类分类[6]，[7]。后者近年来在图像分类等复杂任务中表现出了很好的性能。机器学习的一个关键因素是使用大量数据学习（训练）模型的能力。随着新应用程序生成的数据量不断增加，以及越来越多的应用程序成为数据驱动的，可以预见机器学习任务将成为未来分布式MEC系统的主要工作负载。然而，在资源受限的MEC系统上进行分布式机器学习是一个挑战。

在本文中，我们探讨如何有效地利用有限的计算和通讯资源，以达到最佳的学习效果。我们考虑一个典型的边缘计算架构，其中边缘节点通过网络元素（如网关和路由器）与远程云互连，如图1所示。在多个边缘节点收集和存储原始数据，并根据分布式数据训练机器学习模型，而不需要将原始数据从节点发送到中心位置。这种来自边缘节点联合体的分布式机器学习（模型训练）被称为联合学习[8]–[10]。

我们主要研究基于梯度下降的联邦学习算法，这些算法普遍适用于各种机器学习模型。学习过程包括局部更新步骤，每个边缘节点通过梯度下降来调整（局部）模型参数，以最小化在其自身数据集上定义的损失函数。它还包括全局聚合步骤，其中在不同边缘节点获得的模型参数被发送到聚合器，聚合器是可以在远程云、网络元素或边缘节点上运行的逻辑组件。聚合器聚合这些参数（例如，通过取加权平均值），并将更新后的参数发送回边缘节点，以进行下一轮迭代。全局聚合的频率是可配置的；可以以一个或多个本地更新的间隔进行聚合。每个局部更新消耗边缘节点的计算资源，每个全局聚集消耗网络的通信资源。消耗的资源量可能会随时间而变化，并且全局聚集频率、模型训练精度和资源消耗之间存在复杂的关系。

我们提出了一种算法来确定全局聚集的频率，以便最有效地利用可用资源。这一点很重要，因为机器学习模型的训练通常是资源密集型的，学习任务的非优化操作可能会浪费大量的资源。本文的主要贡献如下：

1） 我们从理论上分析了基于梯度下降的联合学习的收敛界，得到了一个新的收敛界，它包含了节点间的非独立和同分布（noni.i.d.）数据分布和两个全局聚合之间任意数量的局部更新。

2） 利用上述理论收敛界，我们提出了一种学习数据分布、系统动力学和模型特性的控制算法，并在此基础上实时动态调整全局聚合的频率，以使学习损失最小化。

3） 我们通过在硬件原型和模拟环境中使用真实数据集的大量实验来评估所提控制算法的性能，这证实了我们提出的方法对于不同的数据分布、不同的机器学习模型和，以及具有不同数量边缘节点的系统配置。

二。相关工作

现有的关于MEC的工作主要集中在通用应用程序上，针对应用程序卸载[11]、[12]、工作负载调度[13]、[14]和由用户移动性触发的服务迁移[15]、[16]提出了解决方案。然而，它们并没有解决机器学习应用中通信、计算和训练精度之间的关系，这对于优化机器学习任务的性能非常重要。

联邦学习的概念最早是在[9]中提出的，它通过对各种数据集的实验证明了它的有效性。在对比了文献[17]中的同步和异步分布式梯度下降方法的基础上，文[9]提出联合学习应该采用同步方法，因为它比异步方法更有效。[9]中的方法使用固定的全局聚集频率。它不能提供理论上的收敛保证，实验也不是在网络环境下进行的。最近，对最初的联合学习提案进行了一些扩展。例如，在[18]中提出了一种安全的全局聚合机制。在[19]，[20]中提出了压缩在一个全局聚合步骤内交换的信息的方法。在[21]中研究了调整标准梯度下降程序以获得更好的联邦设置性能。[22]研究了联合学习的参与者（客户）选择。文献[23]提出了一种在非i.i.d.数据分布下与其他节点共享少量数据以提高学习性能的方法。这些研究没有考虑全局聚集频率的适应性，因此它们与本文的工作是正交的。据我们所知，在资源受限的情况下，全局聚合频率对联合学习的适应性在文献中还没有被研究过。

与联合学习相关的一个领域是通过使用工作机和参数服务器在数据中心进行分布式机器学习[24]。数据中心环境与边缘计算环境的主要区别在于，在数据中心中，通常使用共享存储。工作机本身不保存持久性数据存储，它们在学习过程开始时从共享存储中获取数据。因此，不同工作者获得的数据样本通常是独立的，且分布相同（i.i.d.）。在联邦学习中，数据直接在边缘采集并持久存储在边缘节点上，因此不同边缘节点上的数据分布通常是非i.i.d.在本文的工作中，在[25]中针对数据中心设置研究了考虑运行时间的同步频率优化问题。它没有考虑非i.i.d.数据分布的特征，这在联合学习中是必不可少的。

文献[26]研究了不同地理位置的多个数据中心的分布式机器学习，提出了一种基于阈值的方法来减少不同数据中心之间的通信。虽然[26]中的工作涉及到根据资源考虑调整同步频率，但它关注点对点连接的数据中心，这不同于非对等的联邦学习体系结构。它还允许数据中心节点之间的异步，这在联邦学习中不是这样。另外，文献[26]中的方法是根据经验设计的，没有考虑具体的理论目标，也没有考虑除通信资源受限外，计算资源约束在MEC系统中也很重要。

从理论上看，文献[27]-[29]给出了分布梯度下降算法收敛的界，在全局聚集前只允许局部更新一步。在[30]，[31]中，分散梯度下降法允许部分全局聚集，其中在每个局部更新步骤之后，参数聚集在非空的节点子集上执行，这不适用于我们的联邦学习设置，在某些局部更新步骤之后根本没有聚集。在[26]中导出的边界中，可以在聚合之前进行多个本地更新，但是本地更新的数量根据阈值过程而变化，不能指定为给定常量。与我们的工作同时，在[32]，[33]中导出了全局聚合步骤之间具有固定数量的本地更新的边界。然而，在[32]中的界限只适用于i.i.d.数据分布；在[33]中的界限独立于数据集的不同程度，这是低效的，因为它没有捕捉到这样一个事实：对i.i.d.数据的培训可能比对非i.i.d.数据的培训更快地收敛。适用于机器学习应用的分布式优化的相关研究还包括[34]–[36]，其中使用单独的解算器来解决局部问题。[34]-[36]的主要关注点是通信和优化之间的权衡，其中没有研究解决本地问题的复杂性（例如需要的本地更新的数量）。此外，现有的许多研究都是显式或隐式地假设i.i.d.数据分布在不同的节点上，这在联合学习中是不合适的。据我们所知，在联邦学习环境下，分布式梯度下降的收敛界既能捕获不同（可能是非i.i.d.分布）数据集的特征，又能在两个全局聚合步骤之间获得给定数量的局部更新步骤，这在文献中还没有被研究过。

与上述研究不同，本文正式讨论了MEC系统中，在给定资源预算的情况下，动态确定全局聚合频率以优化学习的问题。这是一个非常重要的问题，因为每个学习步骤和之前的学习步骤之间存在复杂的依赖关系，很难通过分析来捕捉。由于不同节点的非i.i.d.数据分布，数据分布事先未知，数据集之间可能具有不同程度的相似性，以及系统的实时动态性，这也是一个挑战。我们提出了一种适用于实时系统动力学的理论分析和算法。

在下一节中，我们将从总结联合学习的基础知识开始。在第四节中，我们描述了我们的问题公式。第五节和第六节分别介绍了收敛性分析和控制算法。实验结果见第七节，结论见第八节。

三、 序言和定义

*A、 损失函数*

机器学习模型包括一组基于训练数据学习的参数。训练数据样本通常由两部分组成。一个是作为机器学习模型输入的向量（例如图像的像素）；另一个是模型期望输出的标量（例如图像的标签）。为了便于学习，每个模型在每个数据样本的参数向量上都定义了一个损失函数。损失函数捕捉模型对训练数据的误差，模型学习过程是在一组训练数据样本上最小化损失函数。对于每个数据样本*j***xw公司***jyj公司j*

表一：常用机器学习模型的损失函数

|  |  |
| --- | --- |
| 模型 | 损失函数（w，xj，yj）（，（w））*f福建* |
| 平方支持向量机 | 是常数。） |
| 线性回归 | **wx公司**T*j*k2 |
| 聚类 | 式中，[w（1）T（2）T]T**栈单***,,...* |
| 卷积神经网络 | 线性交叉熵[7]和级联线性熵 |

*j*，我们将损耗函数定义为（w，xj，yj），简写为（w）[1]。*f福建*

表I[6]、[7]、[37]总结了常用机器学习模型的损失函数示例[2]。为了方便起见，本文假设所有向量都是列向量，并用以表示的转置。我们用“，”表示“定义为等于”，用k·k表示L2范数。假设我们有具有本地数据集D1，D2，…，Di，…，DN的边缘节点。对于节点上的每个数据集Di，此节点上收集数据样本的损失函数为**二十**T*N我*

*.* （一）

我们定义| Di |，其中|··|*Di公司*表示集合的大小，以及。假设Di∩Di0=∅对于6=i0，我们将所有分布式数据集的全局损失函数定义为*我*

*.* （二）

注意，（w）不需要在多个节点之间共享信息而直接计算。*F不能*

*B。学习问题*

学习问题是最小化（w），即找到*F*

**w**∗, 阿格明夫（w）。（三）

由于大多数机器学习模型固有的复杂性，通常不可能找到（3）的闭式解。因此，（3）常采用梯度下降法求解。

*C。分布梯度下降*

我们提出了一个求解（3）的典型分布式梯度下降算法，该算法在最先进的联邦学习系统（如[9]）中得到了广泛的应用。每个节点都有其局部模型参数（t），其中=0,1,2，。。。表示迭代索引。当=0时，所有节点的局部参数初始化为相同的值。对于0，（t）的新值根据局部损失函数的梯度下降更新规则，基于上一次迭代−1中的参数值计算。对每个节点的局部损失函数（在局部数据集上定义）的这种梯度下降步骤称为局部更新。在一个或多个之后*我***栈单***我tt我t>我t*

|  |  |  |
| --- | --- | --- |
| 操作： | **A**局部迭代 | **B**全局聚合 |



图2：节点处（t）和（t）值的图示。**栈单***我*e*我我*

算法1：分布式梯度下降（逻辑视图）

输入：，*τT*

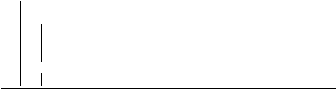
输出：最终模型参数**w**f

1                        将（0）和（0）初始化为相同的值；**万维网**f*我*e*我我*

2                        对于=1,2，…，T do*t*

3                        对于每个节点，使用（4）计算局部更新；*我并联*

4                        如果那样的话*t是的整数倍τ*

5                        设为ei（t）←w（t），其中（t）在（5）中定义；**栈单***我*

//全局聚合

6                        更新←∈{f（t）（w）；**WWW**f阿格明*,*}*F*

7                        其他的

8                        全部设为ei（t）←wi（t）；//无全局聚合**w***我*

本地更新，则通过

聚合器将每个节点的局部参数更新为所有节点参数的加权平均值。我们定义每个迭代都包含一个本地更新步骤，该步骤后面可能有一个全局聚合步骤。

全局聚合后，每个节点的局部参数t通常会发生变化。为了方便起见，我们使用ei（t）来表示可能的全局聚集后节点处的参数。如果在迭代中没有进行聚合，我们有ei（t）=wi（t）。如果在迭代时进行聚合，则通常ei（t）6=wi（t），我们设置（t）=w（t），其中（t）是下文（5）中定义的（t）的加权平均值。这些定义的示例如图2所示。**wwwwwww***我我我tt*e*我我*

每次迭代中的局部更新是在上一次迭代中可能的全局聚合之后对参数执行的。对于每个节点，更新规则如下：*我*

**w***我*（t） =w（t−1）−η∇Fi（w（t−1））（4），其中0是步长。对于任何迭代（可能包括也可能不包括全局聚合步骤），我们定义e*我*e*我η >t*

**w***.* （五）

如果在迭代过程中进行全局聚合，那么这个全局模型参数（t）只对系统中的节点可见，但是我们为所有节点定义了它以便于以后的分析。**w***tt*

我们定义系统在每两个全局聚合之间的每个节点执行局部更新步骤。我们定义为每个节点的局部迭代总数。为了便于表达，我们在理论分析中假设是的整数倍，当我们在第VI-B节讨论实际问题时，这一点将被放宽。分布式梯度下降的逻辑在算法1中给出，它忽略了与聚集器和边缘节点之间的通信有关的方面。这些方面将在后面的第VI-B节中讨论。*τTTτ*

从算法1得到的最终模型参数是在整个算法执行过程中，每次全局聚集后产生最小全局损失的参数。我们使用而不是（T），以符合将在中介绍的理论收敛边界**万维网**ff

表二：主要符号汇总



|  |  |
| --- | --- |
| *F*（w） | 全局损失函数 |
| *金融机构*（w） | 节点局部损失函数*我* |
| *t* | 迭代索引 |
| **w***我*（吨） | 迭代中节点处的局部模型参数*我t* |
| **w**（吨） | 迭代中的全局模型参数*t* |
| **w**f | 在学习过程结束时获得最终模型参数 |

最小化（w）梯度下降步长的真最优模型参数*F*

两个全局聚合之间的本地更新步骤数

每个节点上的本地更新步骤总数

全局聚合步骤总数，等于*T/τ*

)资源类型总数（第个资源类型）在一个本地更新步骤中类型资源的第个类型资源消耗的总预算*米米米*

在一个全局聚合步骤中消耗类型资源*米*

（w）（∀i）和（w）的Lipschitz参数*金融机构F*

（w）（∀i）和（w）的平滑度参数*金融机构F*

梯度散度

（11）中定义的函数，模型参数之间的间隙

|  |  |
| --- | --- |
|  | 从分布和集中的梯度下降中获得 |
| *ϕ* | 引理2中定义的常数，控制参数 |
| *G*(τ) | （18）控制目标中定义的功能 |
| *τ*∗ | 最小化（τ）得到的最优值*τG* |



在实践中，我们已经看到和（T）通常是相同的，但是使用在保证收敛性方面提供了理论上的严格性，因此我们在本文中使用。注意，算法1第6行中的（w）是根据（2）以分布式方式计算的；详细信息将在后面介绍。**WWW**fff*F*

算法1的基本原理是当=1时，即当我们在每个局部更新步骤之后进行全局聚合时，分布式梯度下降（忽略通信方面）相当于集中式梯度下降，后者假设所有的数据样本都在一个集中的位置可用，并且可以直接观察到全局损失函数及其梯度。这是由于梯度算子的线性。由于篇幅限制，请参阅我们的在线技术报告[38，附录A]和[39]，了解有关这方面的详细讨论。*τ*

表二总结了本文的主要符号。

四、 问题表述

当大量数据（通常是训练精确模型所需的）分布在大量节点上时，联邦学习过程会消耗大量的资源。这里“资源”的概念是通用的，可以包括与计算和通信相关的时间、能量、货币成本等。为了不积压系统并保持较低的运行成本，人们常常不得不限制用于学习每个模型的资源量。在计算和通信资源不如数据中心丰富的边缘计算环境中，这一点尤为重要。

因此，一个自然的问题就是如何有效地利用一定数量的资源，使模型训练的损失函数最小化。对于上面提出的基于梯度下降的分布式学习方法，问题的范围缩小到确定和的最优值，从而使全局损失函数在给定的资源约束下最小化。*Tτ*

我们用来表示迭代中全局聚合的总数。因为我们之前假设是*KTTτ*，我们有。我们定义

**w**f,                阿格明F（w）。（六）

**w**∈{w（kτ）：k=0,1,2，…，k}

很容易验证这个定义是否与算法1中的定义等价。**w**f

为了计算（6）中的（w），每个节点首先计算（w）并将结果发送给聚合器，然后聚合器根据（2）计算（w）。在第k个聚集节点（τ）之后（因为在第k个聚集节点上计算τ-or）将被发送回第k个聚集节点（τ-or）。为了计算最后一个损失值（w（Kτ））=F（w（T）），最后进行了一轮局部和全局更新。我们假设在每个节点，局部更新消耗相同的资源量，不管是只计算局部损失（在最后一轮中）还是同时计算局部损失和梯度（在所有其他轮中），因为损失和梯度计算通常基于相同的中间结果。例如，用于计算神经网络中梯度的反向传播方法需要一个前向传播过程，该过程本质上是将损失作为中间步骤[7]。*F我金融机构F***w***k金融机构我FF*

我们考虑不同类型的资源。例如，一种资源可以是时间，另一种资源可以是能量，第三种资源可以是通信带宽等等。对于每一个∈{1,2，…，M}，我们定义所有节点上的每个局部更新步骤消耗单位类型资源，每个全局聚合步骤消耗单位类型资源，其中≥0和≥0都是有限实数。对于给定和，消耗类型资源的总量为（T+1）cm+（K+1）bm，其中额外的“+1”用于计算（w（Kτ）），如上所述。*米米厘米米bm公司米厘米bm公司Tτ米F*

表示资源类型的总预算。我们寻求以下问题的解决方案：*林吉特米*

（7） s.t.（t+1）cm+（K+1）bm≤Rm，∀m∈{1，…，m}t=Kτ。

为了求解（7），我们需要找出和（从而）如何影响根据最终模型参数计算的损失函数（wf）。一般不可能找到一个与（wf）相关的精确解析表达式，因为它取决于梯度下降的收敛性（只有上下界已知[40]），以及全局聚集频率对收敛性的影响。此外，资源消耗和在实践中可能是时变的，这使得问题比（7）更具挑战性。*τKTF***w**f*τKF厘米bm公司*

在第五节中，我们分析了分布式梯度下降算法（算法1）的收敛界，然后利用该界近似求解（7），提出了一种自适应选择最优值并在第六节中达到接近最优资源利用率的控制算法。*τT*

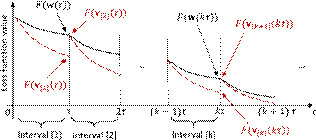


图3：不同间隔的定义说明。

五、 收敛性分析

在这一节中，我们分析了算法1的收敛性，得到了（wf）−F（w∗）的上界。为了便于分析，我们首先介绍一些符号。*F*

*A、 定义*

我们可以将迭代划分为不同的间隔，如图3所示，每个间隔中只有第一次和最后一次迭代包含全局聚合。我们使用速记符号[k]来表示迭代间隔[3][（k−1）τ，kτ]，对于=1,2，…，k。*TKk*

对于每个区间[k]，我们使用（t）来表示一个辅助参数向量，它遵循一个集中梯度下降，根据**五**[克]

**vv型**[克]（t） =v[k]（t−1）−η∇F（v[k]（t−1））（8），其中（t）仅定义为给定的∈[（k−1）τ，kτ]。该更新规则基于全局损失函数（w），该函数仅当所有数据样本在中心位置可用时才可观测（因此我们称之为集中式梯度下降），而（4）中的迭代基于局部损失函数（w）。[克]*tkF金融机构*

我们定义（t）与（t）在每个间隔[k]开始时“同步”，即（（k−1）τ），（（k−1）τ），其中（t）**大众汽车**[克][克]是（5）中定义的局部参数的平均值。请注意，我们还有

因为全局聚集（或初始化当=1时）是在迭代（k−1）τ中执行的。*我k*

通过以上定义，我们可以通过两步的方法找到算法1的收敛界。第一步是找出（kτ）和（kτ）之间的差距，即在没有全局聚集的局部更新步骤后，分布梯度下降和集中梯度下降之间的差异。第二步是在每个区间[k]内将此间隙与（t）的收敛界相结合，得到（t）的收敛界。**wvvw公司**[克]*kτ*[克]

为了分析的目的，我们对损失函数作了如下假设。

假设1。我们假设如下*一：*

*（一）金融机构*（w）*是凸的*

*（二）金融机构*（w） kFi（w）−Fi（w0）k≤ρkw−w0k*是ρ-Lipschitz，即。，对于任何人***栈单***,*0

*（三）金融机构*（w） k∇Fi（w）–∇Fi（w0）k≤βkw−w0k*是β-平滑，即。，对于任何人***栈单***,*0

假设1适用于平方支持向量机和线性回归（见表1）。第七节将介绍的实验结果表明，对于损失函数不满足假设1的模型（如神经网络），我们的控制算法也能很好地工作。

引理1。（w）*F是凸的，ρ-Lipschitz，和β-平滑。*

*证据。*直接从假设1，定义（w）和三角不等式。*F*

我们还定义了以下度量来捕获局部损失函数的梯度和全局损失函数的梯度之间的分歧。这种差异与数据在不同节点上的分布方式有关。

定义1。（梯度散度）对于任何k∇Fi（w）–∇F（w）k*我和***w***，我们定义δi作为，即。，*

                                                                                *.* （九）

*我们还定义*

*B、 主要成果*

下面的定理给出了区间[k]内（t）与（t）之差的上界。**wv公司**[克]*t*

定理1。对于任何区间[k]∈[k]*和t，我们有*

（十）

*哪里*

（十一）

*对于任何人十*=0,1,2，。。。*.*

*此外，作为F*（·）（w（t））−F（v[k]（t））≤ρh（t−（k−1）τ）*是ρ-Lipschitz，我们有F.*

*证据。*我们首先得到每个节点的上界*我*得到最终结果。详见我们的在线技术报告[38，附录B]。

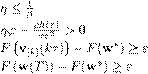
注意，我们总是有0和0，否则梯度下降过程或损失函数变得微不足道。因此，我们得到（ηβ+1）x≥ηβx+1，对于=0,1,2，。。。由于伯努利不等式。将其代入（11）中，证明了（x）≥0。*η >β >十h*

很容易看出（0）=h（1）=0。因此，当=（k−1）τ时，即在区间[k]的开始处，（10）中的上界为零。这与（（k−1）τ）=w（（k−1）τ）的定义一致。当=（k−1）τ+1（即间隔[k]中的第二次迭代），（10）中的上界也是零。这与第III-C节末尾的讨论一致，表明当在全局聚集之后只执行一次局部更新时，分布式梯度下降和集中梯度下降之间没有差距。如果[k]为[1]且[k]为[1]，则∈1。因此，（10）中的上界对于=1变得精确。*ht***五**[克]*ktτttτ*

对于1，=t−（k−1）的值可以更大。当较大时，（11）中的（ηβ+1）x的指数项占优势，（t）与（t）之间的间隙随着∈[k]而呈指数增大。我们还注意到（x）与梯度散度成正比（见（11）），这是直观的，因为局部梯度与全局梯度（对于同一参数）的差异越大，间隙就越大。这种差距是由于在每个全局聚集之后的第二次局部更新开始的不同节点的局部梯度的差异造成的。在极端情况下，当所有节点具有完全相同的数据样本（因此具有相同的局部损失函数），梯度将始终相同且=0，在这种情况下（t）和（t）始终相等。*τ >十十***wvwwv公司**[克]*tthδδ*[克]

定理1给出了每个迭代区间[k]的分布梯度下降和集中梯度下降之间的差的上界，假设集中梯度下降中的（t）与每个[k]开始时的（t）同步。基于这个结果，我们首先得到以下引理。**大众汽车**[克]

引理2。当满足以下所有条件时：

*（一）*

*（二）*

*3） 对所有人k*

*（四）*

*对某些人来说ε >*0*，我们定义**和ω*,

*，则收敛上界*

*算法1之后T迭代由*

*.* （十二）

*证据。*我们首先分析每个区间内的收敛性[k] 一。然后，我们将这个结果与定理1中的（w（t））和（v[k]（t））之间的差距相结合，得到最终的结果。详见我们的在线技术报告[38，附录C]。*FF*

我们得到了下面的定理。

定理2。什么时候？*η*≤*β*1 *，我们有*



*证据。*引理2中的条件1总是满足的，因为这个定理中的条件≤β1。*η*

当（τ）=0时，我们可以选择任意小（但大于零），从而满足引理2中的条件2–4。我们看到（12）和（13）的右边在这种情况下是相等的（当（τ）=0时），并且（13）中的结果直接来自引理2，因为（wf）−F（w∗）≤F（w（T））−F（w∗）。*ρhερhF***w**f

我们在下面考虑（τ）>0。考虑（12）的右边，让*ρh*

*.* （十四）

求解，我们得到*ε*0

（十五）

其中负解被忽略，因为引理2中为0。由于（15）中的0，（14）的分母大于0，因此引理2中的条件2满足任何≥ε0的条件*ε >ε*0 *>ε*，我们注意到随着*ε*当（τ）>0时。*ρh*

假设引理2中存在满足条件3和4，则引理2中的所有条件都满足。应用引理2并考虑（14），我们得到*ε > ε*0



这与引理2中的条件4相矛盾。因此，引理2中不存在同时满足条件3和4的条件。这意味着1）k*ε > ε*0

或2）（w（T））−F（w∗）≤ε0。*F*

因此



根据定理1，（w（kτ））≤F（v[k]（kτ））+ρh（τ）。结合（16），我们得到*Fk*



我们可以回忆一下，Kτ。利用（6）和（15），我们得到了（13）中的结果。*T*

我们注意到（13）中的边界对数据如何分布在不同的节点没有限制。不同数据分布的影响通过梯度散度来捕捉，该散度包含在（τ）中。从（11）中很容易看出（τ）是非负的，非递减的，并且与之成正比。因此，正如人们直观地期望的那样，对于给定的本地更新步骤总数，当且更大时，最优性差距（即（wf）−F（w∗））变得更大。对于给定的和，最优性差距越大，越小。当=1时，我们有（τ）=0，并且最优性差距收敛到零，作为→∞。当1时，我们有（τ）>0，从（13）可以看出，在这种情况下，只保证收敛到非零的最优性间隙为→∞。这意味着当我们对所有类型的资源（即→∞，∀m）有无限的预算时，在每一步的局部更新之后，设置=1并执行全局聚合总是最优的。但是，当资源预算有限时，在有限的迭代次数后训练将终止，因此的值是有限的。在这种情况下，执行全局聚合的频率越低越好，这样就可以将更多的资源用于本地更新，我们将在本文后面介绍。*δhhτδTFτδτδTτhTτ >hT林吉特τ林吉特米T*

六、 控制算法

在本节中，我们提出了一个近似求解（7）的算法。我们首先假设资源消耗和（∀m）已知，然后求解和的值。然后，我们考虑实际情况，其中，和其他一些参数是未知的，并且可能随时间变化，我们提出了一个控制算法，估计参数和动态调整的值，实时。*厘米bm公司τT厘米bm公司τ*

*A、 近似解（7）*

我们假设选择足够小以至于≤β1，并使用（13）中的上界作为（wf）−F（w∗）的近似值。因为对于给定的全局损失函数（w），其最小值（w∗）是一个常数，所以（7）中的（wf）的最小化等价于最小化（wf）−F（w∗）。利用这个近似值并重新排列（7）中的不等式约束，我们可以将（7）重写为*ηηFFFFF*



s、 t。

哪里。

很容易看出，（17）中的目标函数随着而减小，因此它也随着=Kτ而减小。在那里-*TKT*

0 因此，对于任何一个，其最优值，即最大值不违反（17）中的任何不等式约束，其中b·c表示向下舍入为整数的下限函数。为了简化分析，我们通过忽略舍入运算和代入来近似*τK*

进入

（17）中的目标函数，产生



我们可以把（近似）最优定义为*τ*

*τ*∗=argmin G（τ）（十九）

*τ*∈{1,2,3，…}

从中我们可以直接得到（近似）最优

*K*和（大约）

最优的。

命题1。什么时候，我们limR→∞=1，最小Rm最小*τ*∗*，其中R*最小*.*

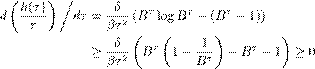
*证据。*因为→∞⇐⇒Rm→∞，∀m⇐⇒*R*最小

*R米*0       →∞，∀m，我们有limmin→∞max0=*Rm c公司米Rτ米*+*τb米*

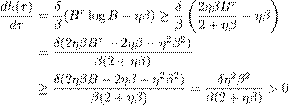
. 因此，limR→∞（τ）=最小*G*

. 让，+1。有点滥用*Bηβ*

表示法，我们考虑连续值≥1。我们有*τ*



其中第一个不等式来自对数函数的下界[41]。我们也有

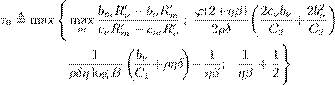


其中第一个不等式来自logB[41]的下界，第二个不等式是因为1且≥1。*B>τ*

因此，对于任何≥1，（τ）随√τ而增加*τh*，且不随*τ*. 我们还注意到，对于任何≥0且（1）=0，随增加。结果表明，对于任何≥1的情况，limR→∞（τ）随着增大而增大。因此，limR→∞=1。*十十十h*最小*Gττ*最小*τ*∗

结合命题1和定理2，我们知道使用（19）中的发现可以保证收敛，最优性差为→∞*τ*∗*R*最小（因此和*T*∗→∞），因为limR→∞=1和（1）=0。对于（和）的一般值，我们得到以下结果。最小*τ*∗*h林吉特林吉特*0

命题2。当≤β1000有限值∀m≤τ0时*η, ρ>, β>, δ>，存在一个τ*0*，这只取决于η,β,ρ,δ,ϕ,厘米，bm公司，林吉特*0 *()，这样τ*∗*. 数量τ*0 *定义为*

;

*where索引ν*, 阿格麦克斯，argmax公司*),*

*B*, *ηβ*+ 1*, C*1 , , . 为了方便起见，我们允许阿格麦克斯*为了互换地返回一个集合和该集合中的任意值，我们还定义**.*

*我们还注意到*0<ηβ≤1*，因此**.*

*证据。*根据*ν*而且，很容易看出这一点*τ*0 是有限的。然后我们展示argmax对于any，在这种情况下，在（18）中的最大化变为fixing=ν。然后，证明分别考虑（18）中平方根内外的项。结果表明，两部分的一阶导数都大于零。因为平方根是一个递增函数，（τ）随着for而增大，因此≤τ0。详见我们的在线技术报告[38，附录D]。*τ > τ*0*米米τ > τ*0*Gττ > τ*0*τ*∗

There is no closed-form solution for because (τ) includes both polynomial and exponential terms of , where the exponential term is embedded in (τ). Because can only be a positive integer, according to Proposition 2, we can compute (τ) within a finite range of to find that minimizes (τ).*τ*∗ *Gτhτ*∗ *Gτ τ*∗ *G*

*B. Adaptive Federated Learning*

In this subsection, we present the complete control algorithm for adaptive federated learning, which recomputes in every global aggregation step based on the most recent system state. We use the theoretical results above to guide the design of the algorithm.*τ*∗

As mentioned earlier, the local updates run on edge nodes and the global aggregation is performed through the assistance of an aggregator, where the aggregator is a logical component that may also run on one of the edge nodes. The complete procedures at the aggregator and each edge node are presented in Algorithms 2 and 3, respectively, where Lines 8–12 of Algorithm 3 are for local updates and the rest is considered as part of global aggregation, initialization, or final operation. We assume that the aggregator initiates the learning process, and the initial model parameter (0) is sent by the aggregator to all edge nodes. We note that instead of transmitting the entire model parameter vector in every global aggregation step, one can also transmit compressed or quantized model parameters to further save the communication bandwidth, where the compression or quantization can be performed using techniques described in [19], [20], for instance.**w**

*1) Estimation of Parameters in G*(τ)The expression of (τ), which includes (τ), has parameters which need to be estimated in practice. Among these parameters, and (∀m) are related to resource consumption, , , and are related to the loss function characteristics. These parameters are estimated in real time during the learning process.*: Ghcm bm ρβδ*

The values of and (∀m) are estimated based on measurements of resource consumptions at the edge nodes and the aggregator (Line 22 of Algorithm 2). The estimation depends on the type of resource under consideration. For example, when the type-resource is energy, the sum energy consumption (per local update) at all nodes is considered as ; when the type-resource is time, the maximum computation time (per local update) at all nodes is considered as . The aggregator also monitors the total resource consumption of each resource type based on the estimates, and compares the total resource consumption against the resource budget (Line 24 of Algorithm 2). If the consumed resource is at the budget limit for some , it stops the learning and returns the final result.*cm bm m cmm cmm Rm m*

The values of , , and are estimated based on the local and global losses and gradients computed at (t) and (t), see Line 11 and Lines 17–19 of Algorithm 2 and Lines 6, 7, and 17 of Algorithm 3. To perform the estimation, each edge node needs to have access to both its local model parameter (t) and the global model parameter (t) for the same iteration (see Lines 6 and 7 of Algorithm 3), which is only possible when global aggregation is performed in iteration . Because (t) is only observable by each node after global aggregation, estimated values of , , and are only available for recomputing starting from the second global aggregation step after initialization, which uses estimates obtained in the previous global aggregation step[4].*ρβδ* **wwwww***iit tρβδ τ*∗

*Remark:* In the extreme case where **w***i*(t) = w(t) in Lines 6 and 7 of Algorithm 3, we estimate *ρ*ˆi and *β*ˆ*i* as zero. When Algorithm 2: Procedure at the aggregator

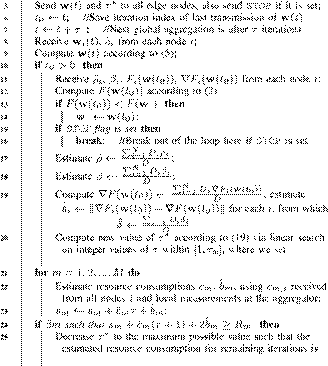
Input: Resource budget , control parameter , search range parameter , maximum value Output: *Rϕγτ τ*max **w**f

1   Initialize ← 1, ← 0, ← 0; //is a resource counter*τ*∗ *t s s*

2   Initialize (0) as a constant or a random vector;**w**

3   Initialize ←w(0);**w**f

4   repeat

∗

*N*

                    we estimate ;

*τ*m ← min{γτ∗;τ}; within budget for all , set STOP flag;max*Rm m*

26    Send (t) to all edge nodes;**w**

27    Receive (w(t)) from each node ;*Fii*

28    Compute (w(t)) according to (2)*F*

29    if (w(t)) < F(wf) then*F*

f

is undefined, we define that (τ) = 0 for all ≥ 1. This is because for 0, (t) = w(t) only occurs when different nodes have extremely similar (often equal) datasets, in which case a large value of does not make the convergence worse than a small value of , thus it makes sense to define (τ) = 0 in this case.*hτ t >* **w***iτ τh*

The parameter is the gradient-descent step size which is pre-specified and known. The remaining parameter includes which is non-straightforward to estimate because the algorithm does not know , thus we regard as a control parameter that is manually chosen and remains fixed for the same machine learning model[5]. Experimentation results presented in the next section show that a fixed value of works well across different data distributions, various numbers of nodes, and various resource consumptions/budgets. If we multiply both sides of (18) by , we can see that a larger value of gives a higher weight to the terms with (τ), yielding a smaller value of (because (τ) increases with ), and vice Algorithm 3: Procedure at each edge node *η ϕ ω* **w**∗*ϕ ϕ ϕϕ hτ*∗ *hτi*

1                                    Initialize ← 0;*t*

2                                    repeat

3                                    Receive (t) and new from aggregator, set ei(t) ←w(t);**ww***τ*∗

4                                    *t*0 ← t; //Save iteration index of last transmission of **w**(t)

5                                    if 0 then*t >*

6                                    Estimate ˆi ←kFi(wi(t)) − Fi(w(t))k/kwi(t) −w(t)k;*ρ*

7                                    Estimate

*β*ˆ*i* ←k∇Fi(wi(t)) −∇Fi(w(t))k/kwi(t) −w(t)k;

8                                    for = 1,2,...,τ∗ do*µ*

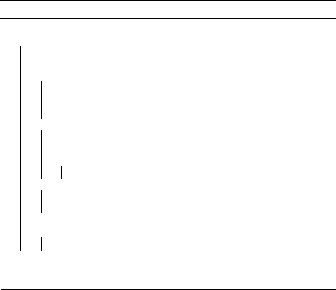
9                                    *t* ← t + 1; //Start of next iteration

10                                  Perform local update and obtain (t) according to (4);**w***i*

11                                  if then*µ < τ*∗

12                                  Set ei(t) ←wi(t);**w**

13                                  for = 1,2,...,M do*m*

14                                  Estimate type-resource consumption ˆm,i for one local update at node ;*m ci*

15                                  Send (t), ˆm,i (∀m) to aggregator;**w***ic*

16                                  if 0 then*t*0 *>*

17                                  Send ˆi, , (w(t0)), ∇Fi(w(t0)) to aggregator;*ρβ*ˆ*iFi*

18                                  until STOP flag is received;

19                                  Receive (t) from aggregator; 20 Send (w(t)) to aggregator;**w***Fi*

versa. Therefore, in practice, it is not hard to tune the value of on a small and simple setup, which can then be applied to general cases. See also the results on the sensitivity of in Section VII-B6.*ϕ ϕ*

*2) Recomputing τ*∗*:* The value of is recomputed by the aggregator during each global aggregation step, based on the most updated parameter estimations. When searching for , we use the following search range instead of the range in Proposition 2 due to practical considerations of estimation error. As shown in Line 20 of Algorithm 2, we search for new values of up to times the current value of , and find that minimizes (τ), where 0 is a fixed parameter. The presence of limits the search space and also avoids from growing too quickly as initial parameter estimates may be inaccurate. We also impose a maximum value of , denoted by , because if is too large, it is more likely for the system to operate beyond the resource budget due to inaccuracies in the estimation of local resource consumption, see Line 24 of Algorithm 2. The new value of is sent to each node together with (t) (Line 5 of Algorithm 2).*τ*∗ *τ*∗*τ*∗ *γ τ*∗*τ*∗ *Gγ > γ τ*∗ *ττ*max*τ*∗ *τ*∗ **w**

*3) Distributed Gradient Descent:* The local update steps of distributed gradient descent at the edge node include Lines 8– 12 of Algorithm 3, where Line 10 of Algorithm 3 corresponds to Line 3 of Algorithm 1 and Line 12 of Algorithm 3 corresponds to Line 8 of Algorithm 1. When global aggregation is performed, Line 9 of Algorithm 2 computes the global model parameter (t) at the aggregator, which is sent to the edge nodes in Line 5 of Algorithm 2, and each edge node receives (t) in Line 3 of Algorithm 3 and sets ei(t) ← w(t) to use (t) as the initial model parameter for the next round of local update; this corresponds to Line 5 of Algorithm 1.**wwww**

The final model parameter that minimizes (w) is obtained at the aggregator in Lines 13–14 of Algorithm 2, corresponding to Line 6 of Algorithm 1. As discussed in Section IV, the computation of lags for one round of global aggregation, because for any iteration that includes a global aggregation step, (w(t0)) can only be computed after each edge node has received (t0) and sent the local loss (w(t0)) to the aggregator in the next round of global aggregation. To take into account the final value of (t) in the computation of , Lines 26–30 of Algorithm 2 and Lines 19– 20 of Algorithm 3 perform an additional round of computation of the loss and , as also discussed in Section IV.**wwwwww**f *F*f *t*0 *FFi*ff

Overall, when global aggregation is executed for times in total, the computational complexity of Algorithm 2 is (K(NM + τ)), because each global aggregation step includes the computation of global parameters from the local parameters collected from different nodes for resource types and the linear search step in Line 20 of Algorithm 2 which has at most steps. When steps of local updates are performed in total, Algorithm 3 has a computational complexity of (T + KM), where the additional term corresponds to the additional local processing (at each node) in global aggregation steps.*K O*max*N M τ*max *T OKM*

*C. Extension to Stochastic Gradient Descent*

When the amount of training data is large, it is usually computationally prohibitive to compute the gradient of the loss function defined on the entire (local) dataset. In such cases, stochastic gradient descent (SGD) is often used [6], [7], [37], which uses the gradient computed on the loss function defined on a randomly sampled subset (referred to as a minibatch) of data to approximate the real gradient. Although the theoretical analysis in this paper is based on deterministic gradient descent (DGD), the proposed approach can be directly extended to SGD. As discussed in [39], SGD can be seen as an approximation to DGD.

When using SGD with our proposed algorithm, all losses and their gradients are computed on mini-batches. Each local iteration step corresponds to a step of gradient descent where the gradient is computed on a mini-batch of local training data. The mini-batch changes for every step of local iteration, i.e., for each new local iteration, a new mini-batch of a given size is randomly selected from the local training data. However, to reduce errors introduced by random data sampling when estimating the parameters , , and , the first iteration after global aggregation uses the same mini-batch as the last iteration before global aggregation. When = 1, the minibatch changes if the same mini-batch has already been used in two iterations, to ensure that different mini-batches are used for training over time.*ρβδτ*

To avoid approximation errors caused by mini-batch sampling when determining , when using SGD, the aggregator informs the edge nodes whether the current (t0) is selected as using an additional flag sent together with the message in Line 5 of Algorithm 2. The edge nodes save their own copies of . When an edge node computes (w(t0)) that is sent in Line 17 of Algorithm 3, it also recomputes (wf) using the same mini-batch as for computing (w(t0)). It then sends both (wf) and (w(t0)) to the aggregator in Line 17 of Algorithm 3. The aggregator recomputes (wf) based on the most recently received (wf). In this way, the values of (wf) and (w(t0)) used for the comparison in Lines 13 and 29 of Algorithm 2 are computed on the same mini-batch at each edge node.**wwww**ff f*FiFiFiFiFiFFiFF*

VII. EXPERIMENTATION RESULTS

*A. Setup*

To evaluate the performance of our proposed adaptive federated learning algorithm, we conducted experiments both on networked prototype system with 5 nodes and in a simulated environment with the number of nodes varying from 5 to 500. The prototype system consists of three Raspberry Pi (version 3) devices and two laptop computers, which are all interconnected via Wi-Fi in an office building. This represents an edge computing environment where the computational capabilities of edge nodes are heterogeneous. All these 5 nodes have local datasets on which model training is conducted. The aggregator is located on one of the laptop computers, and hence co-located with one of the local datasets.

*1) Resource Definition:* For ease of presentation and interpretation of results, we let = 1 and consider time as the single resource type in our experiments. For the prototype system, we train each model for a fixed amount of time budget. The values of and (we omit the subscript = 1 for simplicity) correspond to the actual time used for each local update and global aggregation, respectively. The simulation environment performs model training with simulated resource consumptions, which are randomly generated according to Gaussian distribution with mean and standard deviation values (see [38, Appendix E] for these values) obtained from measurements of the squared-SVM model on the prototype. See Section VII-A4 below for definitions of models and datasets.*M c b m*

*2) Baselines:* We compare with the following baseline approaches:

(a) Centralized gradient descent [6], [7], where the entiretraining dataset is stored on a single edge node and the model is trained directly on that node using a standard (centralized) gradient descent procedure;

(b) Canonical federated learning approach presented in [9],which is equivalent to using a fixed (non-adaptive) value of in our setting;*τ*

(c) Synchronous distributed gradient descent [17], which isequivalent to fixing = 1 in our setting.*τ*

For a fair comparison, we implement the estimation of resource consumptions for all baselines and the training stops when we have reached the resource (time) budget. When conducting experiments on the prototype system, the centralized gradient descent is performed on a Raspberry Pi device. To avoid resource consumption related to loss computation, centralized gradient descent uses the last model parameter (T) (instead of ) as the result, because convergence of (T) can be proven in the centralized case [40]. We do not explicitly distinguish the baselines (b) and (c) above because they both correspond to an approach with non-adaptive of a certain value. When is non-adaptive, we use the same protocol as in Algorithms 2 and 3, but remove any parts related to parameter estimation and recomputation of .**www**f*τ τ τ*

*3) DGD and SGD:* We consider both DGD and SGD in the experiments to evaluate the general applicability of the proposed algorithm. For SGD, the mini-batch sampling uses the same initial random seed at all nodes, which means that when the datasets at all nodes are identical, the mini-batches at all nodes are also identical in the same iteration (while they are generally different across different iterations). This setup is for a better consideration of the differences between equal and non-equal data distributions (see Section VII-A5 below).

*4) Models and Datasets:* We evaluate the training of four different models on five different datasets, which represent a large variety of both small and large models and datasets, as one can expect all these variants to exist in edge computing scenarios. The models include squared-SVM, linear regression, K-means, and deep convolutional neural networks (CNN)[6]. See Table I for a summary of the loss functions of these models, and see [6], [7], [37] for more details. Among them, the loss functions for squared-SVM (which we refer to as SVM in short in the following) and linear regression satisfy Assumption 1, whereas the loss functions for K-means and CNN are non-convex and thus do not satisfy Assumption 1.

SVM is trained on the original MNIST dataset (referred to as MNIST-O) [43], which contains gray-scale images of 70,000 handwritten digits (60,000 for training and 10,000 for testing). The SVM outputs a binary label that corresponds to whether the digit is even or odd. We consider both DGD and SGD variants of SVM. The DGD variant only uses 1,000 training and 1,000 testing data samples out of the entire dataset in each simulation round, because DGD cannot process a large amount of data. The SGD variant uses the entire MNIST dataset.

Linear regression is performed with SGD on the energy dataset [44], which contains 19,735 records of measurements from multiple sensors and the energy consumptions of appliances and lights. The model learns to predict the appliance energy consumption from sensor measurements.

K-means is performed with DGD on the user knowledge modeling dataset [45], which has 403 samples each with 5 attributes summarizing the user interaction with a web environment. The samples can be grouped into 4 clusters representing different knowledge levels, but we assume that we do not have prior knowledge of this grouping.

CNN is trained using SGD on three different datasets, including MNIST-O as described above, the fashion MNIST dataset (referred to as MNIST-F) which has the same format as MNIST-O but includes images of fashion items instead of digits [46], and the CIFAR-10 dataset which includes 60,000 color images (50,000 for training and 10,000 for testing) of 10 different types of objects [47]. A separate CNN model is trained on each dataset, to perform multi-class classification among the 10 different labels in the dataset.

*5) Data Distribution at Different Nodes (Cases 1–4):* For the distributed settings, we consider four different ways of distributing the data into different nodes. In Case 1, each data sample is randomly assigned to a node, thus each node has uniform (but not full) information. In Case 2, all the data samples in each node have the same label[7]. This represents the case where each node has non-uniform information, because the entire dataset has samples with multiple different labels. In Case 3, each node has the entire dataset (thus full information). In Case 4, data samples with the first half of the labels are distributed to the first half of the nodes as in Case 1; the other samples are distributed to the second half of the nodes as in Case 2. This represents a combined uniform and non-uniform case. For datasets that do not have ground truth labels, such the energy dataset used with linear regression, the data to node assignment is based on labels generated from an unsupervised clustering approach.

*6) Training and Control Parameters:* In all our experiments, we set the search range parameter = 10, the maximum value = 100. Unless otherwise specified, we set the control parameter = 0.025 for SVM, linear regression, and K-means, and = 5 × 10−5 for CNN. The gradient descent step size is = 0.01. The resource (time) budget is set as = 15 seconds unless otherwise specified. Except for the instantaneous results in Section VII-B5, the average results of 15 independent experiment/simulation runs are shown.*γ τ τ*max *ϕ ϕ η R*

*B. Results*

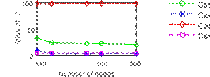
*1) Loss and Accuracy Values:* In our first set of experiments, the SVM, linear regression, and K-means models were trained on the prototype system. Due to the resource limitation of Raspberry Pi devices, the CNN model was trained in a simulated environment of 5 nodes, with resource consumptions generated in the way described in Section VII-A1.

We compare the loss function values of our proposed algorithm (with adaptive ) to baseline approaches, and also compare the classification accuracies for the SVM and CNN classifiers. The results are shown in Fig. 4. We note that the proposed approach only has one data point (represented by a single marker in the figure) in each case, because the value of is adaptive in this case and the marker location shows the average with the corresponding loss or accuracy. The centralized case also only has one data point but we show a flat line across different values of for the ease of comparison. We see that the proposed approach performs close to the optimal point for all cases and all models[8]. We also see that the (empirically) optimal value of is different for different cases and models, so a fixed value of does not work well for all cases. In some cases, the distributed approach can perform better than the centralized approach, because for a given amount of time budget, federated learning is able to make use of the computation resource at multiple nodes. For*ττ τ*∗ *τ τ τ*

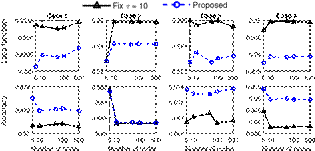
DGD approaches, Case 3 does not perform as well as Case 1,

|  |
| --- |
| Fig. 4: Loss function values and classification accuracy with different . Only SVM and CNN classifiers have accuracy values. The curves show the results from the baseline with different fixed values of . Our proposed solution (represented by a single marker for each case) gives an average and loss/accuracy*τττ* |

that is close to the optimum in all cases.



(a) in proposed algorithm*τ*∗



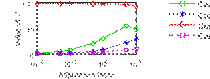
(b) Loss function values and classification accuracy

Fig. 5: SVM (SGD) with different numbers of nodes.

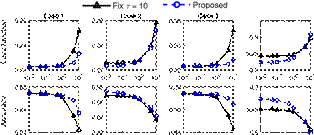
because the amount of data at each node in Case 3 is larger than that in Case 1, and DGD processes the entire amount of data thus Case 3 requires more resource for each local update. Due to the high complexity of evaluating CNN models and the fact that linear regression and K-means models do not provide accuracy values, we focus on the SVM model in the following and provide further insights on the system.

*2) Varying Number of Nodes:* Results of SVM (SGD) for the number of nodes varying from 5 to 500 are shown in Fig. 5, which are obtained in the simulated environment. Our proposed approach performs better than or similar to the fixed = 10 baseline in all cases, where we choose fixed = 10 as the baseline in this and the following evaluations because it is empirically a good value for non-adaptive in different cases according to the results in Fig. 4.*τ τ τ*

*3) Varying Global Aggregation Time:* To study the impact of different resource consumption (time) for global aggregation, we modify the simulation environment so that the global aggregation time is scaled by an adjustment factor. The actual time of global aggregation is equal to the original global aggregation time multiplied by the adjustment factor,



(a) in proposed algorithm*τ*∗

0.35 **Case 4**

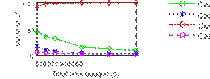
|  |  |  |  |
| --- | --- | --- | --- |
| 10-2      10-1      100             101  Adjustment factor | 10-2     10-1       100                101  Adjustment factor | 10-2               10-1      100                101  Adjustment factor | 10-2               10-1      100       101  Adjustment factor |

(b) Loss function values and classification accuracy

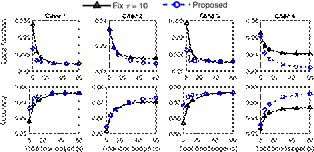
Fig. 6: SVM (SGD) with different global aggregation times.

thus a small adjustment factor corresponds to a small global aggregation time. The results for SVM (SGD) are shown in Fig. 6. Additional results for SVM (DGD) are included in [38, Appendix F]. We can see that as one would intuitively expect, a larger global aggregation time generally results in a larger for the proposed algorithm, because when it takes more time to perform global aggregation, the system should perform global aggregation less frequently, to make the best use of available time (resource). The fact that slightly decreases when the adjustment factor is large is because in this case, the global aggregation time is so large that only a few rounds of global aggregation can be performed before reaching the resource budget, and the value of will be decreased in the last round to remain within the resource budget (see Line 25 of Algorithm 2). Comparing to the fixed = 10 baseline, the proposed algorithm performs better in (almost) all cases.*τ*∗ *τ*∗ *τ*∗ *τ*

*4) Varying Total Time Budget:* We evaluate the impact of the total time (resource) budget on the prototype system. Results for SVM (SGD) are shown in Fig. 7. Further results for SVM (DGD) are included in [38, Appendix G]. We see that



(a) in proposed algorithm*τ*∗



(b) Loss function values and classification accuracy

Fig. 7: SVM (SGD) with different total time budgets.

except for Case 3 where all nodes have the same dataset, the value of of the proposed algorithm decreases with the total time budget. This aligns with the discussion in Section VI-A that becomes close to one when the resource budget is large enough. We also see that the proposed algorithm performs better than or similar to the fixed = 10 baseline in all cases.*τ*∗ *τ*∗ *τ*

*5) Instantaneous Behavior:* We further study the instantaneous behavior of our system for a single run of 30 seconds (for each case) on the prototype system. Results for SVM (DGD) is shown in Fig. 8. Further results for SVM (SGD) are available in [38, Appendix H]. We see that the value of remains stable after an initial adaptation period, showing that the control algorithm is stable. The value of decreases at the end due to adjustment caused by the system reaching the resource budget (see Line 25 of Algorithm 2). As expected, the gradient deviation is larger for Cases 2 and 4 where the data samples at different nodes are non-uniform. The same is observed for and , indicating that the model parameter is in a less smooth region for Cases 2 and 4. In Case 3, the data at different nodes are equal so we always have (t) = w(t) regardless of whether global aggregation is performed in iteration . Thus, the estimated and values are zero by definition, as explained in the remark in Section VI-B1. Case 3 of SVM (DGD) has a much larger value of because it processes more data than in other cases and thus takes more time, as explained before. The value of exhibits fluctuations because of the randomness of the wireless channel.*τ*∗ *τ*∗ *δ ρ β***w w***itρ β c b*

*6) Sensitivity of ϕ:* The sensitivity of the control parameter evaluated on the prototype system is shown in Fig. 9. We see that the relationship among in different cases is mostly maintained with different values of . The value of decreases approximately linearly with logϕ, which is consistent with the fact that there is an exponential term w.r.t. in (τ) (and thus (τ)). For Case 3, remains the same with different , because (τ) = 0 in this case by definition (see the remark in Section VI-B1) and the value of does not affect *ϕ τ*∗ *ϕτ*∗ *τ hGτ*∗ *ϕhϕ τ*∗, as independently of *τ* in this case according to (18). We also see that small changes of does*ϕ*

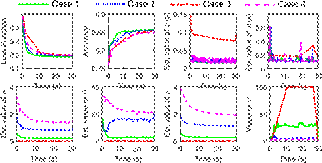


Fig. 8: Instantaneous results of SVM (DGD) with the proposed algorithm.

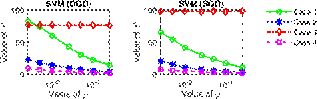


Fig. 9: Impact of on the average value of in the proposed algorithm.*ϕ τ*∗

not change much, indicating that one can take big steps when tuning in practice and the tuning is not difficult.*τ*∗ *ϕ*

*7) Comparison to Asynchronous Distributed Gradient Descent:* Asynchronous gradient descent [17] is an alternative to the typically used synchronous gradient descent in federated learning. With asynchronous gradient descent, the edge nodes operate in an asynchronous manner. Each edge node pulls the most up-to-date model parameter from the aggregator, computes the gradient on its local dataset, then sends the gradient back to the aggregator. The aggregator performs gradient descent according to the step size weighted by the dataset sizes of each node, similar to the combination of (4) and (5). The process repeats until the training finishes. Asynchronous gradient descent is able to fully utilize the available computational resource at each node by running more gradient descent steps at more powerful (faster) nodes. However, the asynchronism may hurt the overall performance.*η*

It was shown in [17] that synchronous gradient descent has benefits over asynchronous gradient descent in a datacenter setting. Here, we study their differences in the edge computing setting with heterogeneous resources (laptops and Raspberry Pis in our experiment) and different data distributions (Cases 1–4). The results for DGD and SGD with SVM are shown in Figs. 10 and 11, respectively. We see that the performance of asynchronous gradient descent is much worse than synchronous gradient descent for non-uniform data distribution in Cases 2 and 4, with slower convergence, sudden changes (indicating instability of the training process), and convergence to higher loss and lower accuracy values. This is because the model tends overfit the datasets on the faster nodes, as many more steps of gradient descent are performed on these nodes compared to the slower nodes. With uniform data distribution (Cases 1 and 3), asynchronous gradient descent performs similar as or slightly better than synchronous gradient descent, because when the datasets at different nodes are similar (Case 1) or equal (Case 3), there is not much harm caused by overfitting the data on the faster nodes.

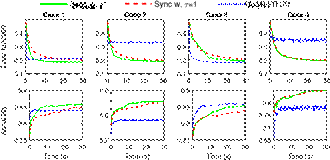


Fig. 10: Synchronous vs. asynchronous distributed DGD with SVM.

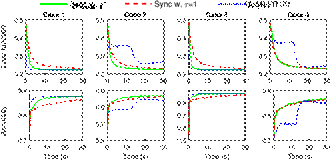


Fig. 11: Synchronous vs. asynchronous distributed SGD with SVM.

Considering the overall performance in all Cases 1–4, we can conclude that it is still better to perform federated learning with synchronous gradient descent as we do throughout this paper. However, how to make more efficient use of heterogeneous resources is something worth investigating in the future.

VIII. CONCLUSION

In this paper, we have focused on gradient-descent based federated learning that include local update and global aggregation steps. Each step of local update and global aggregation consumes resources. We have analyzed the convergence bound for federated learning with non-i.i.d. data distributions. Using this theoretical bound, a control algorithm has been proposed to achieve the desirable trade-off between local update and global aggregation in order to minimize the loss function under a resource budget constraint. Extensive experimentation results confirm the effectiveness of our proposed algorithm. Future work can investigate how to make the most efficient use of heterogeneous resources for distributed learning, as well as the theoretical convergence analysis of some form of non-convex loss functions representing deep neural networks.

REFERENCES

[1] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “When edge meets learning: adaptive control for resourceconstrained distributed machine learning,” in IEEE INFOCOM, Apr. 2018.

[2] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854– 864, Dec. 2016.

[3] R. Kelly, “Internet of Things data to top 1.6 zettabytes by 2020,” Apr. 2015. [Online].

Available: https://campustechnology.com/articles/2015/04/15/internetof-things-data-to-top-1-6-zettabytes-by-2020.aspx

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” IEEE Communications Surveys Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] S. Shalev-Shwartz and S. Ben-David, Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

[7] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

[8] B. McMahan and D. Ramage, “Federated learning: Collaborative machine learning without centralized training data,” Apr.

2017. [Online]. Available: https://ai.googleblog.com/2017/04/federatedlearning-collaborative.html

[9] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in International Conference on Artificial Intelligence and Statistics (AISTATS), 2016.

[10] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, “Wireless network intelligence at the edge,” arXiv, Dec. 2018. [Online]. Available:

http://arxiv.org/abs/1812.02858

[11] Y. Xiao and M. Krunz, “QoE and power efficiency tradeoff for fog computing networks with fog node cooperation,” in IEEE INFOCOM, May 2017, pp. 1–9.

[12] L. Tong and W. Gao, “Application-aware traffic scheduling for workload offloading in mobile clouds,” in IEEE INFOCOM, Apr. 2016.

[13] L. Tong, Y. Li, and W. Gao, “A hierarchical edge cloud architecture for mobile computing,” in IEEE INFOCOM, Apr. 2016.

[14] H. Tan, Z. Han, X.-Y. Li, and F. Lau, “Online job dispatching and scheduling in edge-clouds,” in IEEE INFOCOM, May 2017.

[15] L. Wang, L. Jiao, J. Li, and M. Muhlhauser, “Online resource allocation for arbitrary user mobility in distributed edge clouds,” in IEEE ICDCS, June 2017, pp. 1281–1290.

[16] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dynamic service placement for mobile micro-clouds with predicted future costs,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[17] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, “Revisiting distributed synchronous SGD,” in ICLR Workshop Track, 2016.

[18] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for federated learning on user-held data,” in NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[19] J. Konen, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” in NIPS Workshop on Private Multi-Party Machine Learning, 2016. [Online]. Available: https://arxiv.org/abs/1610.05492

[20] C. Hardy, E. Le Merrer, and B. Sericola, “Distributed deep learning on edge-devices: feasibility via adaptive compression,” in Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on. IEEE, 2017, pp. 1–8.

[21] J. Konen, H. B. McMahan, D. Ramage, and P. Richtarik, “Federated optimization: Distributed machine learning for on-device intelligence,” 2016. [Online]. Available: https://arxiv.org/abs/1610.02527

[22] T. Nishio and R. Yonetani, “Client selection for federated learning with heterogeneous resources in mobile edge,” arXiv preprint arXiv:1804.08333, 2018.

[23] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data,” arXiv preprint arXiv:1806.00582, 2018.

[24] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server.” in OSDI, vol. 14, 2014, pp. 583– 598.

[25] J. Wang and G. Joshi, “Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD,” in SysML, Mar.–Apr. 2019. [Online]. Available: http://arxiv.org/abs/1810.08313

[26] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, “Gaia: Geo-distributed machine learning approaching LAN speeds,” in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017, pp. 629–647.

[27] A. Agarwal and J. C. Duchi, “Distributed delayed stochastic optimization,” in Advances in Neural Information Processing Systems, 2011, pp. 873–881.

[28] X. Lian, Y. Huang, Y. Li, and J. Liu, “Asynchronous parallel stochastic gradient for nonconvex optimization,” in Advances in Neural Information Processing Systems, 2015, pp. 2737–2745.

[29] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, “Asynchronous stochastic gradient descent with delay compensation,” in Proceedings of the 34th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, D. Precup and Y. W.

Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 4120–4129.

[30] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, “Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent,” in Advances in Neural Information Processing Systems, 2017, pp. 5330–5340.

[31] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” arXiv preprint arXiv:1710.06952, 2017.

[32] J. Wang and G. Joshi, “Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms,” arXiv, Jan. 2019. [Online]. Available: http://arxiv.org/abs/1808.07576

[33] H. Yu, S. Yang, and S. Zhu, “Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning,” in AAAI Conference on Artificial Intelligence, Jan.–Feb. 2019.

[34] Y. Zhang, M. J. Wainwright, and J. C. Duchi, “Communication-efficient algorithms for statistical optimization,” in Advances in Neural Information Processing Systems, 2012, pp. 1502–1510.

[35] Y. Arjevani and O. Shamir, “Communication complexity of distributed convex learning and optimization,” in Advances in neural information processing systems, 2015, pp. 1756–1764.

[36] C. Ma, J. Konecnˇ y, M. Jaggi, V. Smith, M. I. Jordan, P. Richt` arik,´ and M. Taka´c, “Distributed optimization with arbitrary local solvers,”ˇ

*Optimization Methods and Software*, vol. 32, no. 4, pp. 813–848, 2017.

[37] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in Proceedings of COMPSTAT&apos;2010. Springer, 2010, pp. 177– 186.

[38] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive federated learning in resource constrained edge computing systems,” Tech. Rep., 2019. [Online]. Available:

https://arxiv.org/abs/1804.05271

[39] T. Tuor, S. Wang, K. K. Leung, and K. Chan, “Distributed machine learning in coalition environments: overview of techniques,” in 21st International Conference on Information Fusion, Jul. 2018.

[40] S. Bubeck, “Convex optimization: Algorithms and complexity,” Foundations and trends in Machine Learning, vol. 8, no. 3-4, 2015.

[41] F. Topsok, “Some bounds for the logarithmic function,” Inequality theory and applications, vol. 4, 2006.

[42] “Advanced convolutional neural networks.” [Online]. Available: https://www.tensorflow.org/tutorials/images/deep%5Fcnn

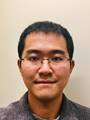
[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] L. M. Candanedo, V. Feldheim, and D. Deramaix, “Data driven prediction models of energy use of appliances in a low-energy house,” Energy and Buildings, vol. 140, pp. 81 – 97, 2017.

[45] H. Kahraman, S.Sagiroglu, and I.Colak, “Developing intuitive knowledge classifier and modeling of users&apos; domain dependent data in web,” Knowledge Based Systems, vol. 37, pp. 283–295, 2013.

[46] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[47] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Tech. Rep., 2009.

Shiqiang Wang (S&apos;13-M&apos;15) received his Ph.D. from the Department of Electrical and Electronic Engineering, Imperial College London, United Kingdom, in 2015. Before that, he received his master&apos;s and bachelor&apos;s degrees at Northeastern University, China, in 2011 and 2009, respectively. He joined IBM T. J. Watson Research Center in 2016 as a Research Staff Member, where he was also a Graduatelevel Co-op in the summers of 2014 and 2013. In the fall of 2012, he was at NEC Laboratories

Europe, Heidelberg, Germany. His current research

focuses on theoretical and practical aspects of mobile edge computing, cloud computing, and machine learning. He has over 40 scholarly publications. Dr. Wang currently serves as an associate editor of IEEE Access and a technical program committee (TPC) member of IEEE ICDCS 2019 and IFIP Networking 2019. In the past, he served as a TPC member of several international conferences including IEEE GLOBECOM, IEEE ICC, IEEE VTC, and as a reviewer for a number of international journals and conferences. He received multiple Invention Achievement Awards from IBM since 2016, and the Best Student Paper Award of the Network and Information Sciences International Technology Alliance (NIS-ITA) in 2015. He was recognized as an exemplary reviewer of the IEEE Transactions on Communications in 2017.

Tiffany Tuor received her B.S. degree in Communication Systems from the Swiss Federal Institute of Technology of Lausanne (EPFL) in 2014. She obtained an M.S degree in Financial Engineering in 2015 and an M.S. degree in Communications and

Signal Processing in 2016 from Imperial College London, UK. Currently, she is working towards a Ph.D. degree in the department of Electrical and Electronic Engineering of Imperial College London.

In summer 2017, she interned at IBM T. J. Watson

Research Center in New York. Her research interest

includes machine learning, cloud computing, and various aspects of future communication networks.

Theodoros Salonidis (S&apos;98-M&apos;04-SM&apos;17) received the Diploma in electronic and computer engineering from the Technical University of Crete, Chania, Greece, in 1997, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, MD, USA, in 1999 and 2004, respectively. He is a Research Staff Member with the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. He was a Post-doctoral Researcher with Rice University, Houston, TX, USA from 2004 to 2006, and a Researcher with Intel Research, Cambridge, U.K., in 2006 and Thomson/ Technicolor, Paris, France, from 2007 to 2012. His current research interests are in the areas of automated AI, distributed analytics, machine learning, and performance analysis, design, and implementation of mobile and cloud computing systems. Dr. Salonidis is a senior member of IEEE and the Technical Chamber of Greece.

Kin K. Leung (F&apos;01) received his B.S. degree from the Chinese University of Hong Kong in 1980, and his M.S. and Ph.D. degrees from University of California, Los Angeles, in 1982 and 1985, respectively. He joined AT&T Bell Labs in New Jersey in 1986 and worked at its successors, AT&T Labs and Lucent Technologies Bell Labs, until 2004. Since then, he has been the Tanaka Chair Professor in the Electrical and Electronic Engineering (EEE), and Computing Departments at Imperial College in London. He is the Head of Communications and Signal Processing Group in the EEE Department. His current research focuses on protocols, optimization and modeling of wireless networks and computer systems. He also works on multi-antenna and cross-layer designs for wireless networks. He received the Distinguished Member of Technical Staff Award from AT&T

Bell Labs (1994), and was a co-recipient of the Lanchester Prize Honorable Mention Award (1997). He was elected an IEEE Fellow (2001), received the Royal Society Wolfson Research Merits Award (2004-09) and became a member of Academia Europaea (2012). Along with his co-authors, he received several best paper awards, including the IEEE PIMRC 2012 and ICDCS 2013. He served as a member (2009-11) and the chairman (2012-15) of the IEEE Fellow Evaluation Committee for ComSoc. He has served as a guest editor for the IEEE JSAC, IEEE Wireless Communications and the MONET journal, and as an editor for the JSAC: Wireless Series, IEEE Trans. on Wireless Communications and IEEE Trans. on Communications. Currently, he is an editor for the ACM Computing Survey and International Journal on Sensor Networks.

Christian Makaya is currently a senior research scientist at HP Labs. Previously, he was a research staff member at IBM T. J. Watson Research

Center and a senior research scientist at Telcordia Technologies. His work has been a catalyst behind several new initiatives and technologies resulting in delivery of high-value capabilities to products and services. For his technical contributions, he has been recognized by several high-prestige awards by IBM and Telcordia. Dr. Makaya leads several technical research activities in the areas of

distributed computing systems and big data analytics with the mission of delivering deep technical breakthroughs and technology transfer. The focus of his current research interests is on edge computing, machine learning, artificial intelligence, network intelligence, Internet of Things (IoT), network functions virtualization, policy-based management systems, and cyber-security. He has authored numerous technical papers in peer-reviewed journals and conferences. He has several filed and issued patents. Dr. Makaya received his Ph.D. in computer engineering from Polytechnique Montreal, University of Montreal. He serves on the Industry Outreach Board of IEEE Communication Society (ComSoc) and as a member of technical program committees of various conferences. He served as the co-chair of IEEE Young Professionals for the IEEE Princeton/Central Jersey Section.

Ting He (S&apos;04-M&apos;07-SM&apos;13) received the B.S. degree in computer science from Peking University, China, in 2003 and the Ph.D. degree in electrical and computer engineering from Cornell University, Ithaca, NY, in 2007. She is an Associate Professor in the School of Electrical Engineering and Computer Science at Pennsylvania State University, University Park, PA. Between 2007 and 2016, she was a Research Staff Member in the Network Analytics Research Group at the IBM T. J. Watson Research

Center, Yorktown Heights, NY. Her work is in the

broad areas of network modeling and optimization, statistical inference, and information theory. Dr. He is a senior member of IEEE. She is an Associate Editor for IEEE Transactions on Communications (2017-2020) and IEEE/ACM Transactions on Networking (2017-2019). She was the Membership co-chair of ACM N2Women in 2013-2014 and was listed in N2Women: Rising Stars in Networking and Communications in 2017. She has served on the TPC of a range of communications and networking conferences, including IEEE INFOCOM (Distinguished TPC Member), IEEE ICNP, IEEE SECON, IEEE WiOpt, IEEE/ACM IWQoS, IEEE MILCOM, IEEE ICNC,

IFIP Networking, etc. She received the Research Division Award and the Outstanding Contributor Awards from IBM in 2016, 2013, and 2009. She received the Most Collaboratively Complete Publications Award by ITA in 2015, the Best Paper Award at the 2013 International Conference on Distributed Computing Systems (ICDCS), a Best Paper Nomination at the

2013 Internet Measurement Conference (IMC), and the Best Student Paper Award at the 2005 International Conference on Acoustic, Speech and Signal

Processing (ICASSP). Her students received the Outstanding Student Paper Award at the 2015 ACM SIGMETRICS and the Best Student Paper Award at the 2013 ITA Annual Fall Meeting.

Kevin Chan (S&apos;02-M&apos;09-SM&apos;18) received his B.S. in Electrical and Computer Engineering and Engineering & Public Policy from Carnegie Mellon University, Pittsburgh, PA in 2001 and his M.S.E.C.E and PhD in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, GA in 2003 and 2008, respectively. Dr. Chan is currently a research scientist with the Computational and Information Sciences Directorate at the U.S. Army Combat Capabilities Development Command Army

Research Laboratory (Adelphi, MD). Dr. Chan is

actively involved in research on network science, distributed analytics and cybersecurity. He has received multiple best paper awards and the NATO Scientific Achievement Award. He currently is the co-editor for the IEEE Communications Magazine Military Communications and Networks Series.

[[1]](" \l "_ftnref1" \o ") Note that some unsupervised models (such as K-means) only learn on and do not require the existence of in the training data. In such cases, the loss function value only depends on .**xx***j yj j*

[[2]](" \l "_ftnref2" \o ") While our focus is on non-probabilistic learning models, similar loss functions can be defined for probabilistic models where the goal is to minimize the negative of the log-likelihood function, for instance.

[[3]](" \l "_ftnref3" \o ") With slight abuse of notation, we use [(k−1)τ,kτ] to denote the integers contained in the interval for simplicity. We use the same convention in other parts of the paper as long as there is no ambiguity.

[[4]](" \l "_ftnref4" \o ") See the condition in Line 10 of Algorithm 2 and Lines 5 and 16 of Algorithm 3. Also note that the parameters ˆi, , (w(t0)), ∇Fi(w(t0)) sent in Line 17 of Algorithm 3 are obtained at the previous global aggregation step (, ˆi, and are obtained in Lines 4–7 of Algorithm 3).*ρβ*ˆ*iFit*0*ρβ*ˆ*i*

[[5]](" \l "_ftnref5" \o ") Although is related to and we estimate separately, we found that it is good to keep a constant value that does not vary with the estimated value of in practice, because there can be occasions where the estimated is large causing 0, which causes abnormal behavior when computing from (τ).*ϕ β β ϕ β β ϕ < τ*∗ *G*

[[6]](" \l "_ftnref6" \o ") The CNN has 9 layers with the following structure: 5×5×32 Convolutional → 2 × 2 MaxPool → Local Response Normalization → 5 × 5 × 32 Convolutional → Local Response Normalization → 2×2 MaxPool → z×256 Fully connected → 256×10 Fully connected → Softmax, where depends on the input image size and = 1568 for MNIST-O and MNIST-F and = 2048 for CIFAR-10. This configuration is similar to what is suggested in the TensorFlow tutorial [42].*z z z*

[[7]](" \l "_ftnref7" \o ") When there are more labels than nodes, each node may have data with more than one label, but the number of labels at each node is no more than the total number of labels divided by the total number of nodes rounded to the next integer.

[[8]](" \l "_ftnref8" \o ") Note that the loss and accuracy values shown in Fig. 4 can be improved if we allow a longer training time. For example, the accuracy of CNN on MNIST data can become close to 1.0 if we allow a long enough time for training. The goal of our experiments here is to show that our proposed approach can operate close to the optimal point with a fixed and limited amount of training time (resource budget) as defined in Section VII-A6.