联合优化：

面向设备智能的分布式机器学习

|  |  |  |
| --- | --- | --- |
| 雅库布·科恩cny´  爱丁堡大学库博科内尼@gmail.com网站 | H、 布伦丹·麦克马汉  谷歌mcmahan@google.com | 丹尼尔·拉马奇  谷歌dreamage@google.com |

彼得·里克塔里克

爱丁堡大学彼得.里奇塔里克@英国教育部

2016年10月11日

# 摘要

我们为机器学习中的分布式优化引入了一个新的且越来越相关的设置，其中定义优化的数据不均匀地分布在大量的节点上。目标是培养一个高质量的集中式模型。我们将此设置称为联邦优化。在这种情况下，沟通效率是最重要的，减少沟通的次数是主要目标。

当我们将培训数据本地保存在用户的移动设备上，而不是将其记录到数据中心进行培训时，就会出现一个令人鼓舞的例子。在联邦优化中，设备被用作计算节点，对其本地数据执行计算，以更新全局模型。我们假设网络中有非常多的设备，相当于一个给定服务的用户数量，每一个用户的可用数据只占总数据的一小部分。特别是，我们期望本地可用的数据点的数量远远小于设备的数量。此外，由于不同的用户以不同的模式生成数据，因此可以合理地假设没有设备具有代表性的总体分布样本。

我们指出现有的算法不适合这种情况，并提出了一种新的算法，在稀疏凸问题上取得了令人鼓舞的实验结果。这项工作也为联邦优化的未来研究提供了一条道路。

# 1           介绍

平板电脑和移动电话现在是许多人的主要计算设备。在许多情况下，这些设备很少与所有者分离[19]，丰富的用户交互和强大的传感器相结合意味着它们可以访问前所未有的数据量，其中大部分数据本质上是私有的。从这些数据中学习到的模型有希望通过支持更智能的应用程序来大大提高可用性，但数据的敏感特性意味着将其存储在集中位置存在风险和责任。

我们提倡另一种方法——联合学习——将训练数据分布在移动设备上，并通过中央协调服务器聚合本地计算的更新来学习共享模型。这是对2012年白宫消费者数据隐私报告提出的集中收集或数据最小化原则的直接应用[98]。由于这些更新是为改进当前模型而专门设计的，所以它们可能只是昙花一现—一旦应用了这些更新，就没有理由将它们存储在服务器上。此外，它们永远不会包含比原始训练数据更多的信息（通过数据处理不平等），并且通常包含的信息要少得多。这种方法的一个主要优点是模型训练与直接访问原始训练数据的需要脱钩。显然，仍然需要对协调训练的服务器进行一些信任，并且根据模型和算法的细节，更新可能仍然包含私有信息。但是，对于可以根据每个客户机上可用的数据指定培训目标的应用程序，联合学习可以通过将攻击面仅限于设备而不是设备和云来显著降低隐私和安全风险。

如果需要额外的隐私，可以使用来自差异隐私的随机化技术。可以修改集中式算法来生成一个差异私有模型[17，33，1]，它允许发布模型，同时保护参与培训过程更新的个人的隐私。如果需要防止恶意（或受损）协调服务器，则可以应用来自本地差异隐私的技术来私有化单个更新[32]。这方面的细节超出了目前的工作范围，但它是未来研究的一个有希望的方向。

关于联合学习的应用以及隐私影响的更完整的讨论可以在[62]中找到。我们在这项工作中的重点将放在联邦优化上，这是一个必须解决的优化问题，以便使联邦学习成为当前方法的一个实际替代方案。

## 1.1         问题表述

近年来，优化界对用有限和结构求解问题的兴趣激增。总的来说，目标被表述为

)在哪里。（一）

动机的主要来源是机器学习中出现的问题。问题结构（1）包括线性或逻辑回归、支持向量机，但也包括更复杂的模型，如条件随机场或神经网络。

我们假设我们有一组输入输出对，和一个损失函数，产生函数fi。通常，席Rand yi∈R或yi∈{−1,1}。简单的例子包括*d*

•线性回归：

•逻辑回归：fi（w）=−log（1+exp（−yixTi w）），yi∈{−1,1}

•支持向量机：fi（w）=max{0,1−yixTi w}，yi∈{−1,1}

在神经网络的背景下出现了更复杂的非凸问题，而不是通过特征映射XTI W中的线性，网络通过特征向量席的非凸函数进行预测。然而，产生的损耗仍然可以写成fi（w），并且可以使用反向传播有效地计算梯度。

企业、政府和学术项目收集的数据量正在迅速增加。因此，在实践中出现的问题（1）通常不可能在单个节点上解决，因为仅仅在单个节点上存储整个数据集变得不可行。这就需要使用分布式计算框架，在这个框架中，描述问题的训练数据以分布式的方式存储在多个相互连接的节点上，优化问题由节点群共同解决。

不严格地说，可以使用任何节点网络来模拟单个强大的节点，在该节点上可以运行任何算法。实际问题是，在同一个节点上的处理器和内存之间进行通信所需的时间通常比两个节点通信所需的时间小许多个数量级；对于所需的能量，也有类似的结论[89]。此外，为了利用每个节点上的并行计算能力，有必要将问题细分为适合独立/并行计算的子问题。

最先进的优化算法通常是连续的。此外，它们通常依赖于执行大量非常快速的迭代。问题源于这样一个事实：如果一个人需要在每次迭代后执行一轮通信，那么实际性能就会急剧下降，因为这一轮通信比算法的一次迭代要耗费更多的时间。

这些考虑导致了专门用于分布式优化的新算法的开发（我们将彻底的回顾推迟到第2节）。目前，我们注意到文献中的大多数结果都是在数据均匀分布的情况下工作的，并进一步假设其中K是节点数。当数据被存储时，这确实非常接近现实

在大型数据中心。此外，分布式学习领域的一个重要子领域依赖于这样一个假设：每台机器都有一个本地可用数据的代表性样本。也就是说，假设每台机器都有一个来自底层分布的IID样本。然而，这种假设往往过于强烈；事实上，即使在数据中心范式中，情况也往往并非如此，因为单个节点上的数据可以在时间尺度上彼此接近，或按其地理来源聚集。由于数据中的模式会随时间而变化，因此一个特性可能会频繁地出现在一个节点上，而在另一个节点上根本不会出现。

联邦优化设置描述了一个新的优化场景，上面的假设都不成立。我们将在下一节中更详细地概述这种设置。

## 1.2         联邦优化的设置

本文的主要目的是让机器学习和优化界注意到一个新的、越来越切合实际的分布式优化设置，在这种情况下，没有一个典型的假设得到满足，而通信效率是最重要的。特别是，联邦优化算法必须处理具有以下特征的训练数据：

•：数据点存储在大量的节点K上，特别是节点的数量可能远大于存储在给定节点（n/K）上的训练示例的平均数量。**大规模分布**

•：每个节点上的数据可能来自不同的分布；也就是说，本地可用的数据点远远不是整个分布的代表性样本。**非IID**

•：不同的节点在它们所拥有的训练示例的数量级上可能会有所不同。**不平衡**

在这项工作中，我们特别关注稀疏数据，其中一些特征只出现在一小部分节点或数据点上。虽然这不是联邦优化设置的必要特征，但是我们将证明稀疏结构可以用于开发有效的联邦优化算法。请注意，在目前正在解决的最大的机器学习问题中，广告点击率预测中出现的数据非常稀少。

我们对用户移动设备（手机和平板电脑）上的培训数据的存储环境特别感兴趣，这些数据可能对隐私敏感。数据{席，y}是通过设备使用，例如通过与应用程序的交互而生成的。例如，预测用户将键入的下一个单词（智能键盘应用程序的语言建模），预测用户最有可能分享哪些照片，或者预测哪些通知最重要。

为了使用传统的分布式学习算法来训练这些模型，人们将在一个集中的位置（数据中心）收集训练示例，在那里可以对训练样本进行洗牌并均匀地分布在专有计算节点上。在本文中，我们提出并研究了一个替代模型：训练样本不被发送到一个集中的位置，这样可以节省大量的网络带宽并提供额外的隐私保护。作为交换，用户允许使用他们的设备的计算能力，这将被用来训练模型。

在本文的通信模型中，每轮发送一个更新δ∈Rto给一个集中式服务器，其中d是被计算/改进的模型的维数。例如，更新δ可以是梯度向量。虽然在某些应用中δ可以对用户的一些私人信息进行编码，但是它可能比原始数据本身敏感得多（并且数量级更小）。例如，考虑原始训练数据是移动设备上的大量视频文件集合的情况。更新δ的大小与本地训练数据集的大小无关。我们证明，一个全局模型可以使用少量的通信轮进行训练，因此与将数据复制到数据中心相比，这也减少了训练所需的网络带宽数量级。*d*

此外，非正式地，我们选择δ作为改进全局模型所需的最小信息；与原始数据相比，它在其他用途上的效用显著降低。因此，设计一个存储这些δ的时间不超过更新模型所需时间的系统是很自然的，这同样增加了隐私，减少了集中式模型培训师的责任。在这种情况下，每一轮都有一个向量δ∈Ris进行通信，涵盖了大多数现有的一阶方法，包括CoCoA+[57]等双重方法。*d*

在大规模分布式环境中，通信约束自然会产生，因为网络连接可能受到限制（例如，我们可能希望在移动设备充电并连接到wi-fi网络之前解除所有通信）。因此，在现实情况下，我们可能仅限于每天进行一次沟通。这意味着，在合理的范围内，我们可以访问基本上无限的本地计算能力。因此，实际目标仅仅是尽量减少沟通轮数。

这项工作的主要目的是发起研究，并设计第一个实际的联邦优化实现。我们的结果表明，在适当的优化算法下，由于没有可用的IID样本，损失很小，而且即使在存在大量节点的情况下，我们仍然可以在相对较少的通信周期内实现收敛。

# 2           相关工作

在本节中，我们将提供相关文献的详细概述。我们特别关注在各种情况下可以用来解决问题（1）的算法。首先，在第2.1节和第2.2节中，我们将研究设计在单台计算机上运行的算法。在第2.3节中，我们将讨论分布式环境，在这种情况下，没有任何一个节点可以直接访问描述f的所有数据。我们描述了一个衡量分布式方法效率的范例，然后概述了现有的方法，并对它们的设计是否考虑了通信效率进行了评论。

## 2.1         基线算法

在这一节中，我们将描述几个基本的基线算法，这些算法可以用来解决形式（1）的问题。

**梯度下降。**当函数fi是光滑的（或非光滑函数的次梯度下降）时，解决结构（1）问题的一个简单基准是梯度下降（GD）[69]。GD算法执行迭代

*wt*+1 =w−h∇f（w），*ttt*

其中ht>0是stepsize参数。正如我们前面提到的，函数的数量，或者相当于训练数据对的数量n，通常非常大。这使得GD不切实际，因为它需要处理整个数据集，以便评估单个梯度并更新模型。

在理论和实践中，通过增加动量项，可以大大加速梯度下降。凸优化中梯度法的加速思想可以追溯到Polyak[73]和Nesterov[68,69]的工作。GD方法至少需要一个更快的收敛速度。因此，对于n非常大的问题，它们是不实用的。

**随机梯度下降。**目前，GD的一个基本的（尽管在实践中非常流行）替代方法是随机梯度下降（SGD），这可以追溯到Robbins和Monro的开创性工作[82]。在（1）的上下文中，SGD在迭代t中抽取一个随机函数（即随机数据标签对），它∈{1，2，…，n}，并执行更新

*wt*+1 =w−h∇f（w），*tt我tt*

其中ht>0是stepsize参数。直观地说，这种方法是有效的，因为如果它从指数1到n均匀随机抽样，更新方向是梯度-E[∇fi（w）]=∇f（w）的无偏估计。然而，采样过程中引入的噪声会减慢收敛速度，收敛需要一个步长hk的小序列。对于凸函数的理论分析，我们建议读者参考[66，64，65]和[87，93]来解决支持向量机问题。在最近的一篇综述[12]中，作者概述了进一步的研究方向。关于更具实际意义的讨论，见[11]。在神经网络的背景下，随机梯度的计算被称为反向传播[49]。反向传播不是显式地指定函数fi及其梯度，而是计算梯度的一种通用方法。在[70]中比较了几种用于训练深层神经网络的竞争算法的性能。*t*

一个被实际观察到的提供优异性能的常见技巧是在每次迭代中用随机顺序遍历所有函数来代替随机抽样。在每一个这样的循环之后，这个顺序被另一个随机顺序所取代[10]。对这一现象的理论理解是一个长期存在的开放性问题，最近在[40]中得到了理解。

GD和SGD之间的核心区别可以总结如下。GD的收敛速度很快，但是（1）上下文中的每次迭代都可能非常慢，因为它需要在每次迭代中处理整个数据集。另一方面，SGD的收敛速度较慢，但每次迭代都很快，因为所需的工作量与数据点n的数量无关。对于（1）的问题结构，SGD通常更好，因为实际需要的精度相对较低，在极端情况下，SGD可以通过单次通过数据来实现，而GD只会进行一次更新。然而，如果需要高精度，GD或其更快的变体将占优势。

## 2.2         一种新的随机算法

近年来出现了大量新的随机方法，在一级近似中，将SGD的廉价迭代和GD的快速收敛相结合。这些方法大多可以说是两类方法中的一类：随机坐标下降变量的对偶方法和方差降阶的随机梯度下降的原始方法。

**随机坐标下降。**尽管坐标下降的概念在不同的环境中已经存在了几十年（对于二次函数，可以追溯到更久远的高斯-赛德尔方法），但随着Nesterov[67]的工作，它在机器学习和优化中变得突出，它为该方法配备了随机化策略。Nesterov在随机坐标下降（RCD）方面的工作推广了这种方法，并证明随机化对于结构问题（1）非常有用。

每次迭代的RCD算法选择一个随机坐标jt∈{1，…，d}，并执行更新w+1=w−h∇（w）e*ttjtj转铁蛋白tjt，*

其中hj0是一个步长参数，∇jf（w）表示函数f的第j个偏导数，ej是R中第j个单位标准基向量。对于广义线性模型，当数据表现出一定的稀疏结构时，可以有效地计算偏导数∇jf（w），即：。，在不需要处理整个数据集的情况下，就可以得到一个实际有效的算法，参见示例[79，第6节]。*t>d*

大量的后续工作将这一概念扩展到近端设置[79]，单处理器并行[15，80]，并开发出高效可实现的加速[51]。在[35]中，所有这三个属性在一个单独的算法中被联系在一起，我们向读者推荐RCD领域早期发展的回顾，特别是其中表1的概述。

**随机对偶上升。**当一个显式强凸（但不一定是光滑的）正则化器加到平均损失（1）上时，可以写出它的（Fenchel）对偶，并且对偶变量存在于n维空间中。应用RCD得到了一种求解（1）的算法，其名称为随机双坐标上升[88]。这种方法得到了实践工作者的广泛欢迎，可能是因为对于许多损失函数，该方法不需要调整任何超参数。这项工作[88]首先表明，通过将RCD[79]应用于对偶问题，我们也解决了原始问题（1）。关于将RCD应用于原始问题和对偶问题的理论和计算比较，见[21]。

文献[75]提出了一种直接原始-双重随机坐标下降法，称为石英法。最近发现，有时在低维坐标系中可以包含一些随机曲率信息。最近的工作[86,20]解释了仅在原始设置下的SDCA方法，阐明了为什么这种方法作为一种具有方差缩减特性的SGD方法工作。

我们现在移动第二类新的随机算法，通常可以解释为SGD的变体，试图减少梯度估计过程中固有的方差。

**随机平均梯度。**第一个值得注意的算法是随机平均梯度（SAG）[83，85]。SAG算法存储了算法历史上不同点的n个函数梯度的平均值。在每一次迭代中，算法都会从这个平均值中随机选择更新梯度，并朝平均值的方向前进一步。这样，每次迭代的复杂度与n无关，且算法收敛速度快。该算法的缺点是由于更新操作需要在内存中存储n个梯度。在广义线性模型的情况下，由于梯度是数据点的标量倍数，这种内存需求可以减少到n个标量。这种方法最近被扩展用于条件随机域[84]。然而，内存的需求使得该算法即使在相对较小的神经网络中也不适用。

后续算法SAGA[26]及其简化[25]对SAG算法进行了改进，以实现梯度的无偏估计。内存需求仍然存在，但是该方法大大简化了理论分析，并且产生了稍微更强的收敛保证。

**随机方差约化梯度。**另一种方法是[44]的随机梯度[1]。SVRG算法在两个嵌套循环中运行。在外环中，它计算整个函数的全梯度∇f（wt），这是一个通常要避免的昂贵运算。在内部循环中，更新步骤迭代计算为w=w−h[∇fi（w）–∇fi（wt）+∇f（wt）]。

其核心思想是利用随机梯度来估计wt点和w点之间的梯度变化，而不是直接估计梯度。我们将在第3.2节中返回对该算法的更详细描述。

SVRG的优点是它不需要SAG/SAGA额外的内存需求，但它需要时不时地处理整个数据集。实际上，与SGD相比，SVRG通常在第一次通过数据方面取得了显著进展，SVRG没有进行任何更新，因为它需要计算完整的梯度。这一点和其他几个实际问题最近在[41]中得到了解决，使算法在早期与SGD竞争，在以后的迭代中更为优越。虽然没有什么可以阻止我们在深度学习中应用SVRG及其变体，但是我们还不知道在这种情况下对SVRG的性能进行任何系统的评估。[43，77]中的普通实验表明SVRG与基本SGD匹配，甚至在迭代方差对SVRG来说明显更小的意义上表现得更好。然而，为了得出任何有意义的结论，人们需要进行广泛的实验，并与通常配备有许多启发式方法的最新方法进行比较。

已经有人尝试将SVRG型算法与随机坐标下降相结合[46，97]。虽然这些工作突出了一些有趣的理论性质，但目前这些算法似乎并不实用，这方面还需要更多的工作。SAGA论文[26]首次尝试统一SVRG和SAG/SAGA等算法，作者将SAGA解释为SAG和SVRG之间的中点。最近的工作[76]提出了一种通用算法，它将SVRG、SAGA、SAG和GD恢复为特殊情况，并得到这些算法的一个异步变量作为公式的副产品。（SVRG）是一种新的动量加速方法（SVRG）。SVRG可以通过原始的集群机制进一步加速

[4] 一。

**随机拟牛顿方法。**第三类新算法是随机拟线性方法[16,9]。这些算法通常试图模仿有限内存BFGS方法（L-BFGS）[54]，但是使用来自SGD过程的不精确梯度来建模局部曲率信息。最近尝试将这些方法与SVRG相结合可以在[63]中找到。在文[38]中，作者利用随机矩阵反演[39]的最新进展，揭示了拟牛顿方法的新联系，并设计了一种新的随机有限记忆BFGS方法与SVRG相结合。与上述方法相比，这一研究分支的理论理解最少，并且有几个细节使得实现更加困难，这可能限制了它的广泛应用。然而，一旦理解得更好，这种方法对于深度学习可能是最有希望的。

机器学习的一个重要方面是，我们正在解决的经验风险最小化问题（1）只是我们最终感兴趣的预期风险的代理。当一个人能找到经验风险的精确最小值时，一切都简化为平衡近似值——大量文献的目标——估计权衡——例如见[96]。文献[13]介绍了一些优化算法作为学习算法在大规模学习问题中的渐近性能评估[2]。最近在[41]中的扩展表明，方差减少算法（SAG，SVRG，…）在某些情况下可以比SGD更好地学习算法，而不仅仅是更好的优化算法。

**进一步说明。**一种被称为通用催化剂的通用方法[53，37]，可以有效地将前面章节中提到的许多算法转换为它们的“加速”变体。由此产生的收敛性保证在许多情况下几乎匹配下界。但是，由于需要调整附加参数，因此该方法相当不切实际。

最近，关于形式问题随机方法复杂性的上下界

（1） 最近在[99]中获得。

## 2.3         分布式设置

在这一节中，我们回顾了有关在分布式环境下求解（1）的算法的文献。当我们谈到分布式设置时，我们指的是描述功能fi的数据没有存储在任何单个存储设备上的情况。这可以包括设置一个人的数据不适合一个单独的RAM/计算机/节点，但两个就足够了。这还包括数据分布在世界各地的几个数据中心以及这些数据中心中的许多节点的情况。关键是在系统中，没有一个处理单元可以直接访问所有数据。因此，分布式设置不包括单处理器并行[3]。与任何单个节点上的本地计算相比，节点之间的通信成本在速度和能耗方面都要高得多[6，89]，这不仅给优化过程带来了新的计算挑战。

在第2.3.1节中，我们首先回顾了为给定问题确定实际最佳算法的理论决策规则，然后在第2.3.2节中概述了分布式算法，在第2.3.3节中介绍了通信效率算法。下面的范例强调了为什么通信高效算法类不仅仅是琐碎意义上的首选。通信效率高的算法为设计整体优化过程提供了更为灵活的工具，使算法能够内在地适应计算资源和体系结构的差异。

**2.3.1衡量分布式优化效率的范例**

本节回顾了一个比较分布式算法效率的范例。让我们假设我们有许多现成的算法来解决这个问题（1）。问题是：“我们如何决定哪种算法最适合我们的目的？“这种推理的最初版本已经出现在[57]中，也适用于[78]。

首先，考虑单台机器上的基本设置。让我们定义）作为迭代次数，算法A需要收敛到某个固定精度。设TA为单个迭代所需的时间。然后，在实践中，最好的算法是使以下数量最小化的算法

                                                                                    时间=。（二）

迭代次数）通常由理论保证给出或从经验中观察。TA可以通过经验来观察，也可以知道每次迭代所需的时间在不同的算法之间是如何变化的。这种简化设置的要点是通过将算法扩展到分布式设置来突出关键问题。

分布式设置的自然扩展如下。设c是算法a的一次迭代过程中通信所需的时间。为了清楚起见，我们假设我们只考虑在Rper轮通信中需要通信单个向量的算法。注意，基本上所有的一阶算法都属于这一类，所以这并不是一个限制性的假设，它有效地将c设置为常量，给定任何一个可以使用的特定分布式体系结构。*d*

                                                                               时间=）（3）

通信成本c不仅包括实际的数据交换，还包括许多其他的事情，如建立和关闭节点之间的连接。因此，即使我们需要传递非常少量的信息，c始终保持在一个重要的阈值之上。

在（2）的设置中，大多数（如果不是全部）最先进的算法都是随机的，并且依赖于进行大量（大的）非常快（小TA）的迭代。因此，即使是相对较小的c也会导致这些算法的实际性能急剧下降，因为。

这在实践中确实得到了观察，并推动了新方法的开发，从零开始就考虑到了这一点，我们在第2.3.2节中对此进行了回顾。虽然这对学术界来说是一个很好的发展——探索新环境的动机，但对行业来说并不一定是个好消息。

许多公司已经花费了大量的资源来构建优秀的算法来解决表单（1）中的问题，并根据它们所需的数据和辅助应用程序中出现的特定模式进行了微调。当公司收集的数据变得太大而无法在一台机器上处理时，可以理解的是，他们不愿意放弃他们微调过的算法。这个问题在CoCoA[57]中首次明确提出，这是一个框架，而不是一个算法，其工作原理如下（更详细的描述见第2.3.3节）。

CoCoA框架基于本地可用的数据和需要分发到所有节点的单个共享向量，制定了在每个节点上形成特定子问题的通用方法。在框架的一次迭代中，每个节点使用任何优化算法a，以达到局部子问题的相对精度。然后，来自所有节点的更新被聚合，形成对全局模型的更新。

效率范式的变化如下：

                                                                         时间=（Θ））（4）

迭代次数Θ）与作为局部解算器的算法A的选择无关，因为有理论预测，如果我们将局部子问题求解到相对精确度，则需要多少次CoCoA框架迭代才能达到精度。这里，Θ=0意味着我们需要子问题被解决到最优性，而Θ=1表示我们不需要任何进展。对于强凸目标，CoCoA框架迭代次数的一般上界是[42，58，57]。从对1−Θ的逆依赖关系中，我们可以看出所需通信轮数有一个基本限制。因此，花费过多的资源来获得非常高的局部精度（小Θ）可能是不有效的。每次迭代时间TA（Θ）表示算法A在局部子问题上达到相对Θ精度所需的时间。

这种效率范式更强大的原因有很多。

1.    它允许实践者继续使用他们的微调解算器，这些解算器只能在单机上运行，而不必从头开始实现全新的算法。

2.    实际性能在通信轮数方面与优化算法的选择无关，使得优化整体性能更加容易。

3.    由于常数c依赖于体系结构，所以在一个节点网络上运行的优化算法不必在另一个节点上是最优的。在设置（3）中，这可能意味着从一个集群移动到另一个集群，一个完全不同的算法是最优的，这是一个重大的变化。在设置（4）中，这可以通过简单地改变Θ来改进，这通常是由算法A运行的迭代次数隐式确定的。

在这项工作中，我们提出了一种不同的方法来描述局部子问题，它不像CoCoA那样依赖于二元性。我们还强调了一些算法似乎特别适合于解决这些局部子问题，有效地导致了新的分布式优化算法。

**2.3.2分布式算法**

如下文第2.3.1节所述，这种设置带来了独特的挑战。分布式优化算法通常每次迭代需要少量（1-4）次通信轮。通过沟通，我们通常理解一个单一的MapReduce操作[24]，有效地为迭代过程[36]实现，例如优化算法。Spark[102]是一个流行的开源框架，用于实现分布式迭代算法，包括本节中提到的几种算法。

分布式环境下的优化研究已经进行了几十年，至少可以追溯到Bertsekas和Tsitsiklis[8，7，95]的工作。近十年来，由于机器学习应用中数据可用性的迅速增加，人们对这一领域的兴趣激增。

最近的大部分工作都集中在创建新的优化算法上，通过构建适合在单处理器上运行的流行算法的变体（参见第2.1节）。其中许多方法的一个相对常见的特点是A）同步算法的计算开销，以及b）在没有限制性假设的情况下分析异步算法的难度。如果在计算周期内更新所有的计算节点，就意味着我们不能更新所有的计算节点。这导致一些节点处于空闲状态，而剩余的节点则完成其部分的计算，这显然是对计算资源的低效利用。这种模式通常会减少或完全恢复分布式计算的潜在加速。通常，在异步设置中，可以对参数向量应用更新，然后根据该参数向量的过时版本进行计算。在形式上把握这一格局往往是颇具挑战性的。因此，这是一个非常开放的领域，在任何特定情况下，算法的最佳选择通常很大程度上取决于问题的大小、其结构的细节、可用的计算体系结构，尤其是从业者的专业知识。

这个普遍的问题最好地表现在多次尝试并行化随机梯度下降及其变体。作为一个例子，[27，29]理论上提供了节点数量的线性加速，但是很难有效地实现，因为节点需要频繁地同步以计算合理的梯度平均值。作为替代，在[71，2，31]中假设工人之间没有同步。因此，每个工作者从存储器中读取wt，在时间点t处的参数向量w，计算一个随机梯度∇fi（wt），并将其应用于参数向量wt+τ已经改变的状态。上述方法假设延迟τ受一个常数的约束，这不一定是现实的假设[4]。一些工作还引入了关于稀疏结构或f的Hessian条件的假设。渐近最优收敛速度在[30]中用相当温和的假设进行了证明。在[22]中还提出了异步SGD的改进分析，同时引入了一个在不牺牲性能的情况下使用较低精度算法的版本，这是一个在未来几年可能在机器学习的其他部分得到应用的趋势。

当用于训练非常大的深层网络的任务时，SGD的异步分布式实现的负面影响似乎可以忽略不计——这是当今的最终工业应用。例如，谷歌的倾盆大雨SGD[23]和微软的项目Adam[18]已经证明了其实用性。

坐标下降算法的第一个分布式版本是Hydra及其加速变体Hydra2，[81,34]，它已经被证明在计算集群上实现的大型稀疏问题上非常有效。关于实现细节的描述的扩展版本见[61]。在[56，55]的著作中，已经对异步的影响进行了探索，并部分地从理论上理解了这一点。[72]中提出了另一种适用于更广泛目标类别的异步坐标更新框架，而不是算法。

在上述算法中，假设数据按特征/坐标被划分为节点。如果不能预先分发数据，而是“按原样”分布数据，则此设置可能会受到限制，在这种情况下，数据最常见的分布方式是数据点。如果使用坐标下降的双重版本，这不一定是一个问题，在这种情况下，通过数据点[94]进行分配，然后进行通信有效的双坐标上升，如下节所述。然而，对偶的使用需要使用额外的显式强凸正则化项，因此可以用来解决小类问题。尽管分布式坐标下降算法在实际应用中存在明显的缺点，但它仍然是实际应用最广泛的方法之一。

Moving to variance reduced methods, distributed versions of SAG/SAGA algorithms have not been proposed yet. However, several distributed versions of the SVRG algorithm already exist. A scheme for replicating data to simulate iid sampling in distributed environment was proposed in [50]. Although the performance is well analysed, the setting requires significantly stronger control of data distribution which is rarely practicaly feasible. A relatively similar method to Algorithm 3 presented here has been proposed in [78], which was analysed, and in [59], a largely experimental work that can be also cast as communication efficient — described in detail in Section 2.3.3.

Another class of algorithms relevant for this work is Alternating Direction Method of Multipliers (ADMM) [14, 28]. These algorithms are in general applicable to much broader class of problems, and hasn&apos;t been observed to perform better than other algorithms presented in this section, in the machine learning setting of (1).

**2.3.3 Communication-Efficient Algorithms**

In this Section, we describe algorithms that can be cast as “communication efficient”. The common theme of the algorithms presented here, is that in order to perform better in the sense of (3), one should design algorithms with high TA, in order to make the cost of communcation c negligible.

Before moving onto specific methods, it is worth the noting some of the core limits concerning the problem we are trying to solve in distributed setting. Fundamental limitations of stochastic versions of the problem (1) in terms of runtime, communication costs and number of samples used are studied in [90]. Efficient algorithms and lower bounds for distributed statistical estimation are established in [104, 103].

However, these works do not fit into our framework, because they assume that each node has access to data generated IID from a single distribution. In the case of [104, 103] also, that the number of nodes K is much smaller than the number of data point on each node is also assumed. As we stress in the Introduction, these assumptions are far from being satisfied in our setting. Intuitively, relaxing these assumptions should make the problem harder. However, it is not as straightforward to conclude this, as there are certainly particular non-iid data distributions that simplify the problem — for instance if data are distributed according to separability structure of the objective. Lower bounds on communication complexity of distributed convex optimization of (1) are presented in [5], concluding that for IID data distributions, existing algorithms already achieve optimal complexity in specific settings.

Probably first, rather extreme, work [107] proposed to parallelize SGD in a single round of communication. Each node simply runs SGD on the data available locally, and their outputs are averaged to form a final result. This approach is however not very robust to differences in data distributions available locally, and it has been shown [91, Appendix A] that in general it cannot perform better than using output of a single machine, ignoring all the other data.

Shamir et al. proposed the DANE algorithm, Distributed Approximate Newton [91], to exactly solve a general subproblem available locally, before averaging their solutions. The method relies on similarity of Hessians of local objectives, representing their iterations as an average of inexact Newton steps. We describe the algorithm in greater detail in Section 3.4 as our proposed work builds on it. A quite similar approach was proposed in [59], with richer class class of subproblems that can be formulated locally, and solved approximately. An analysis of inexact version of DANE and its accelerated variant, AIDE, appeared recently in [78]. Inexact DANE is closely related to the algorithms presented in this paper. We, however, continue in different direction shaped by the setting of federated optimization.

The DiSCO algorithm [105] of Zhang and Xiao is based on inexact damped Newton method. The core idea is that the inexact Newton steps are computed by distributed preconditioned conjugate gradient, which can be very fast, if the data are distributed in an IID fashion, enabling a good preconditioner to be computed locally. The theoretical upper bound on number of rounds of communication improves upon DANE and other methods, and in certain settings matches the lower bound presented in [5]. The DiSCO algorithm is related to [52, 106], a distributed truncated Newton method. Although it was reported to perform well in practice, the total number of conjugate gradient iterations may still be high to be considered a communication efficient algorithm.

Common to the above algorithms is the assumption that each node has access to data points sampled IID from the same distribution. This assumption is not required only in theory, but can cause the algorithms to converge significantly slower or even diverge (as reported for instance in [91, Table 3]). Thus, these algorithms, at least in their default form, are not suitable for the setting of Federated Optimization presented here.

An algorithm that bypasses the need for IID data assumption is CoCoA, which provably converges under any distribution of the data, while the convergence rate does depend on properties of the data distribution. The first version of the algorithm was proposed as DisDCA in [101], without convergence guarantees. First analysis was introduced in [42], with further improvements in [58], and a more general version in [57]. Recently, its variant for L1-regularized objectives was introduced in [92].

The CoCoA framework formulates general local subproblems based on the dual form of (1) (See for instance [57, Eq. (2)]). Data points are distributed to nodes, along with corresponding dual variables. Arbitrary optimization algorithm is used to attain a relative Θ accuracy on the local subproblem — by changing only local dual variables. These updates have their corresponding updates to primal variable w, which are synchronously aggregated (could be averaging, adding up, or anything in between; depending on the local subproblem formulation).

From the description in this section it appears that the CoCoA framework is the only usable tool for the setting of Federated Optimization. However, the theoretical bound on number of rounds of communications for ill-conditioned problems scales with the number of nodes K. Indeed, as we will show in Section 4 on real data, CoCoA framework does converge very slowly.

# 3           Algorithms for Federated Optimization

In this section we introduce the first algorithm that was designed with the unique challenges of federated optimization in mind. Before proceeding with the explanation, we first revisit two important and at first sight unrelated algorithms. The connection between these algorithms helped to motivate our research. Namely, the algorithms are the Stochastic Variance Reduced Gradient (SVRG) [43, 47], a stochastic method with explicit variance reduction, and the Distributed Approximate Newton (DANE) [91] for distributed optimization.

The descriptions are followed by their connection, giving rise to a new distributed optimization algorithm, at first sight almost identical to the SVRG algorithm, which we call Federated SVRG (FSVRG).

Although this algorithm seems to work well in practice in simple circumstances, its performance is still unsatisfactory in the general setting we specify in Section 3.3. We proceed by making the FSVRG algorithm adaptive to different local data sizes, general sparsity patterns and significant differences in patterns in data available locally, and those present in the entire data set.

## 3.1         Desirable Algorithmic Properties

It is a useful thought experiment to consider the properties one would hope to find in an algorithm for the non-IID, unbalanced, and massively-distributed setting we consider. In particular:

(A) If the algorithm is initialized to the optimal solution, it stays there.

(B) If all the data is on a single node, the algorithm should converge in O(1) rounds of communication.

(C) If each feature occurs on a single node, so the problems are fully decomposable (each machine is essentially learning a disjoint block of parameters), then the algorithm should converge in O(1) rounds of communication[5].

(D) If each node contains an identical dataset, then the algorithm should converge in O(1) rounds of communication.

For convex problems, “converges” has the usual technical meaning of finding a solution sufficiently close to the global minimum, but these properties also make sense for non-convex problems where “converge” can be read as “finds a solution of sufficient quality”. In these statements, O(1) round is ideally exactly one round of communication.

Property (A) is valuable in any optimization setting. Properties (B) and (C) are extreme cases of the federated optimization setting (non-IID, unbalanced, and sparse), whereas (D) is an extreme case of the classic distributed optimization setting (large amounts of IID data per machine). Thus, (D) is the least important property for algorithms in the federated optimization setting.

## 3.2         SVRG

The SVRG algorithm [43, 47] is a stochastic method designed to solve problem (1) on a single node. We present it as Algorithm 1 in a slightly simplified form.



**Algorithm 1** SVRG

|  |  |
| --- | --- |
| 1: = number of stochastic steps per epoch, h = stepsize**parameters:** *m*  2: = 0,1,2,... do**for** *s*  3: Compute and store | *.* Full pass through data |
| 4: Set w = wt  5: = 1 to m do**for** *t*  6: Pick  7: = w − h ∇fi(w) − ∇fi(wt) + ∇f(wt)*w*  8: **end for**  9: +1 = w*wt*  10: **end for** | *.* Stochastic update |

The algorithm runs in two nested loops. In the outer loop, it computes gradient of the entire function f (Line 3). This constitutes for a full pass through data — in general expensive operation one tries to avoid unless necessary. This is followed by an inner loop, where m fast stochastic updates are performed. In practice, m is typically set to be a small multiple (1–5) of n. Although the theoretically optimal choice for m is a small multiple of a condition number [47, Theorem 6], this is often of the same order as n in practice.

The central idea of the algorithm is to avoid using the stochastic gradients to estimate the entire gradient ∇f(w) directly. Instead, in the stochastic update in Line 7, the algorithm evaluates two stochastic gradients, ∇fi(w) and ∇fi(wt). These gradients are used to estimate the change of the gradient of the entire function between points wt and w, namely ∇f(w) − ∇f(wt). Using this estimate together with ∇f(wt) pre-computed in the outer loop, yields an unbiased estimate of ∇f(w).

Apart from being an unbiased estimate, it could be intuitively clear that if w and wt are close to each other, the variance of the estimate ∇fi(w)−∇fi(wt) should be small, resulting in estimate of ∇f(w) with small variance. As the inner iterate w goes further, variance grows, and the algorithm starts a new outer loop to compute new full gradient ∇f(wt+1) and reset the variance.

The performance is well understood in theory. For λ-strongly convex f and L-smooth functions fi, convergence results are in the form

E[f(wt) − f(w∗)] ≤ ct[f(w0) − f(w∗)], (5)

where w∗ is the optimal solution, and



7See [47, Theorem 4] and [43, Theorem 1] for details.

It is possible to show [47, Theorem 6] that for appropriate choice of parameters m and h, the convergence rate (5) translates to the need of



evaluations of ∇fi for some i to achieve.

## 3.3         Distributed Problem Formulation

In this section, we introduce notation and specify the structure of the distributed version of the problem we consider (1), focusing on the case where the fi are convex. We assume the data , describing functions fi are stored across a large number of nodes.

Let K be the number of nodes. Let Pk for k ∈ {1,...,K} denote a partition of data point indices {1,...,n}, so Pk is the set stored on node k, and define nk = |Pk|. That is, we assume that Pk ∩ Pl = ∅ whenever k 6= l, and. We then define local empirical loss as

*,* (6)

which is the local objective based on the data stored on machine k. We can then rephrase the objective (1) as

*.* (7)

The way to interpret this structure is to see the empirical loss) as a convex combination of the local empirical losses Fk(w), available locally to node k. Problem (1) then takes the simplified form

*.* (8)

## 3.4         DANE

In this section, we introduce a general reasoning providing stronger intuitive support for the DANE algorithm [91], which we describe in detail below. We will follow up on this reasoning in Appendix A and draw a connection between two existing methods that was not known in the literature.

If we wanted to design a distributed algorithm for solving the above problem (8), where node k contains the data describing function Fk. The first, and as we shall see, a rather naive idea is to ask each node to minimize their local functions, and average the results (a variant of this idea appeared in [107]):

*K*

                                                       *wkt*+1 = arg min .

*w*

*k*=1

Clearly, it does not make sense to run this algorithm for more than one iteration as the output w will always be the same. This is simply because does not depend on t. In other words, this method effectively performs just a single round of communication. While the simplicity is appealing, the drawback of this method is that it can&apos;t work. Indeed, there is no reason to expect that in general the solution of (8) will be a weighted average of the local solutions, unless the local functions are all the same — in which case we do not need a distributed algorithm in the first place and can instead solve the much simpler problem minw∈R*d F*1(w). This intuitive reasoning can be also formally supported, see for instance [91, Appendix A].

One remedy to the above issue is to modify the local problems before each aggregation step. One of the simplest strategies would be to perturb the local function Fk in iteration t by a quadratic term of the form: and to ask each node to solve the perturbed problem instead. With this change, the improved method then takes the form

*K*

*.* (9)

*k*=1

The idea behind iterations of this form is the following. We would like each node k ∈ [K] to use as much curvature information stored in Fk as possible. By keeping the function Fk in the subproblem in its entirety, we are keeping the curvature information nearly intact — the Hessian of the subproblem is ∇2Fk + µI, and we can even choose µ = 0.

As described, the method is not yet well defined, since we have not described how the vectors atk would change from iteration to iteration, and how one should choose µ. In order to get some insight into how such a method might work, let us examine the optimality conditions. Asymptotically as t → ∞, we would like atk to be such that the minimum of each subproblem is equal to w∗; the minimizer of (8). Hence, we would wish for w∗ to be the solution of

∇Fk(w) − atk + µ(w − wt) = 0.

Hence, in the limit, we would ideally like to choose atk = ∇Fk(w∗)+µ(w∗−wt) ≈ ∇Fk(w∗), since w∗ ≈ wt. Not knowing w∗ however, we cannot hope to be able to simply set atk to this value. Hence, the second option is to come up with an update rule which would guarantee that converges to ∇Fk(w∗) as t → ∞. Notice at this point that it has been long known in the optimization community that the gradient of the objective at the optimal point is intimately related to the optimal solution of a dual problem. Here the situation is further complicated by the fact that we need to learn K such gradients. In the following, we show that DANE is in fact a particular instantiation of the scheme above.

**DANE.** We present the Distributed Approximate Newton algorithm (DANE) [91], as Algorithm 2. The algorithm was originally analysed for solving the problem of structure (7), with nk being identical for each k — i.e., each computer has the same number of data points. Nothing prevents us from running it in our more general setting though.

As alluded to earlier, the main idea of DANE is to form a local subproblem, dependent only on local data, and gradient of the entire function — which can be computed in a single round of communication (Line 3). The subproblem is then solved exactly (Line 4), and updates from individual nodes are averaged to form a new iterate (Line 5). This approach allows any algorithm to be used to solve the local subproblem (10). As a result, it often achieves communication efficiency in the sense of requiring expensive local computation between rounds of communication, hopefully rendering the time needed for communication insignificant (see Section 2.3.1). Further, note that DANE belongs to the family of distributed method that operate via the quadratic perturbation trick (9) with atk = ∇Fk(wt) − η∇f(wt).

**Algorithm 2** Distributed Approximate Newton (DANE)



1: regularizer µ ≥ 0, parameter η (default: µ = 0,η = 1)**Input:**

2: = 0,1,2,... do**for** *s*

3: Compute) and distribute to all machines

4: For each node k ∈ {1,...,K}, solve

                                        *wk* = argmin (10)

*w*∈

5: Compute

6: **end for**



If we assumed that the method works, i.e., that wt → w∗ and hence ∇f(wt) → ∇f(w∗) = 0, then atk → ∇Fk(w∗), which agrees with the earlier discussion.

In the default setting when µ = 0 and η = 1, DANE achieves desirable property (D) (immediate convergence when all local datasets are identical), since in this case ∇Fk(wt) − η∇f(wt) = 0, and so we exactly minimize Fk(w) = f(w) on each machine. For any choice of µ and η, DANE also achieves property (A), since in this case ∇f(wt) = 0, and wt is a minimizer of Fk(w)−∇Fk(wt)·w as well as of the regularization term. Unfortunately, DANE does not achieve the more federated optimization-specific desirable properties (B) and (C).

|  |
| --- |
|  |
|  |  |

The convergence analysis for DANE assumes that the functions are twice differentiable, and relies on the assumption that each node has access to IID samples from the same underlying distribution. This implies that that the Hessians of√ ∇2Fk(w) are similar to each other [91, Lemma



1]. In case of linear regression, with λ = O(1/ n)-strongly convex functions, the number of DANE iterations needed to achieve -accuracy is )). However, for general L-smooth loss, the theory is significantly worse, and does not match its practical performance.

The practical performance also depends on the additional local regularization parameter µ. For small number of nodes K, the algorithm converges quickly with µ = 0. However, as reported [91, Figure 3], it can diverge quickly with growing K. Bigger µ makes the algorithm more stable at the cost of slower convergence. Practical choice of µ remains an open question.

## 3.5         SVRG meets DANE

As we mentioned above, the DANE algorithm can perform poorly in certain settings, even without the challenging aspects of federated optimization. Another point that is seen as drawback of DANE is the need to find the exact minimum of (10) — this can be feasible for quadratics with relatively small dimension, but infeasible or extremely expensive to achieve for other problems. We adapt the idea from the CoCoA algorithm [57], in which an arbitrary optimization algorithm is used to obtain relative Θ accuracy on a locally defined subproblem. We replace the exact optimization with an approximate solution obtained by using any optimization algorithm.

Considering all the algorithms one could use to solve (10), the SVRG algorithm seems to be a particularly good candidate. Starting the local optimization of (10) from point wt, the algorithm automatically has access to the derivative at wt, which is identical for each node — ∇f(wt). Hence, the SVRG algorithm can skip the initial expensive operation, evaluation of the entire gradient (Line 3, Algorithm 1), and proceed only with the stochastic updates in the inner loop.

It turns out that this modified version of the DANE algorithm is equivalent to a distributed version of SVRG.

**Proposition 1.** *Consider the following two algorithms.*

*1.    Run the DANE algorithm (Algorithm 2) with η* = 1 and µ = 0, and use SVRG (Algorithm 1) as a local solver for (10), running it for a single iteration, initialized at point wt.

*2.    Run a distributed variant of the SVRG algorithm, described in Algorithm 3.*

*The algorithms are equivalent in the following sense. If both start from the same point wt, they generate identical sequence of iterates* {wt}.

*Proof.* We construct the proof by showing that single step of the SVRG algorithm applied to the problem (10) on computer k is identical to the update on Line 8 in Algorithm 3.

The way to obtain a stochastic gradient of (10) is to sample one of the functions composing

), and add the linear term ∇Fk(wt)−ηf(wt), which is known and does not

need to be estimated. Upon sampling an index i ∈ Pk, the update direction follows as

*,*

which is identical to the direction in Line 8 in Algorithm 3. The claim follows by chaining the



**Algorithm 3** naive Federated SVRG (FSVRG)

|  |  |  |
| --- | --- | --- |
| 1: = # of stochastic steps per epoch, h = stepsize, data partition**parameters:** *m*  2: = 0,1,2,... do**for** *s*  3: Compute | | *.* Overall iterations |
| 4: | **for** *k* = 1 to K do in parallel over nodes k | *.* Distributed loop |
| 5: | Initialize: wk = wt |  |
| 6: | **for** *t* = 1 to m do | *.* Actual update loop |
| 7: | Sample |  |
| 8: | *wk* = wk − h ∇fi(wk) − ∇fi(wt) + ∇f(wt) |  |
| 9: | **end for** |  |
| 10: | **end for** |  |
| 11:  12: **end for** | | *.* Aggregate |

**Remark 2.** *The algorithms considered in Proposition 1 are inherently stochastic. The statement of the proposition is valid under the assumption that in both cases, identical sequence of samples i* ∈ Pk would be generated by all nodes k ∈ {1,2,...,K}.

**Remark 3.** *In the Proposition 1 we consider the DANE algorithm with particular values of η and µ. The Algorithm 3 and the Proposition can be easily gereralized, but we present only the default version for the sake of clarity.*

Since the first version of this paper, this connection has been mentioned in [78], which analyses an inexact version of the DANE algorithm. We proceed by adapting the above algorithm to other challenges arising in the context of federated optimization.

## 3.6         Federated SVRG

Empirically, the Algorithm 3 fits in the model of distributed optimization efficiency described in Section 2.3.1, since we can balance how many stochastic iterations should be performed locally against communication costs. However, several modifications are necessary to achieve good performance in the full federated optimization setting (Section 3.3). Very important aspect that needs to be addressed is that the number of data points available to a given node can differ greatly from the average number of data points available to any single node. Furthermore, this setting always comes with the data available locally being clustered around a specific pattern, and thus not being a representative sample of the overall distribution we are trying to learn. In the Experiments section we focus on the case of L2 regularized logistic regression, but the ideas carry over to other generalized linear prediction problems.

**3.6.1 Notation**

Note that in large scale generalized linear prediction problems, the data arising are almost always sparse, for example due to bag-of-words style feature representations. This means that only a small subset of d elements of vector xi have nonzero values. In this class of problems, the gradient ∇fi(w) is a multiple of the data vector xi. This creates additional complications, but also potential for exploitation of the problem structure and thus faster algorithms. Before continuing, let us summarize and denote a number of quantities needed to describe the algorithm.

• — number of data points / training examples / functions.*n*

• Pk — set of indices, corresponding to data points stored on device k.

• = |Pk| — number of data points stored on device k.*nk*

— the number of data points with nonzero jth coordinate

— the number of data points stored on node k with nonzero jth

coordinate

• = nj/n — frequency of appearance of nonzero elements in jth coordinate*φj*

• = njk/nk — frequency of appearance of nonzero elements in jth coordinate on node k*φjk*

• = φj/φjk — ratio of global and local appearance frequencies on node k in jth coordinate*sjk*

• = Diag(sjk) — diagonal matrix, composed of sjk as jth diagonal element*Sk*

— Number of nodes that contain data point with nonzero jth coordinate

• = K/ωj — aggregation parameter for coordinate j*aj*

• = Diag(aj) — diagonal matrix composed of aj as jth diagonal element*A*

With these quantities defined, we can state our proposed algorithm as Algorithm 4. Our experiments show that this algorithm works very well in practice, but the motivation for the particular scaling of the updates may not be immediately clear. In the following section we provide the intuition that lead to the development of this algorithm.

**3.6.2 Intuition Behind FSVRG Updates**

The difference between the Algorithm 4 and Algorithm 3 is in the introduction of the following properties.

1.    Local stepsize — hk = h/nk.

|  |  |
| --- | --- |
| **Algorithm 4** Federated SVRG (FSVRG) |  |
| 1: **parameters:** *h* = stepsize, data partition, diagonal matrices A,Sk ∈ Rfor k ∈ {1,...,K}*d*×d  2: = 0,1,2,... do**for** *s* | *.* Overall iterations |
| 3: Compute  4: = 1 to K do in parallel over nodes k**for** *k* | *.* Distributed loop |
| 5: Initialize: wk = wt and hk = h/nk  6: Let be random permutation of Pk  7: = 1,...,nk do**for** *t* | *.* Actual update loop |
| 8:  9: **end for**  10: **end for**  11:  12: **end for** | *.* Aggregate |

2.    Aggregation of updates proportional to partition sizes —

3.    Scaling stochastic gradients by diagonal matrix — Sk

4.    Per-coordinate scaling of aggregated updates — A(wk − wt)

Let us now explain what motivated us to get this particular implementation.

As a simplification, assume that at some point in time, we have for some w, wk = w for all k ∈ [K]. In other words, all the nodes have the same local iterate. Although this is not exactly the case in practice, thinking about the issue in this simplified setting will give us insight into what would be meaningful to do if it was true. Further, we can hope that the reality is not too far from the simplification and it will still work in practice. Indeed, all nodes do start from the same point, and adding the linear term ∇Fk(wt)−∇f(wt) to the local objective forces all nodes to move in the same direction, at least initially.

Suppose the nodes are about to make a single step synchronously. Denote the update direction on node k as Gk = ∇fi(w)−∇fi(wt)+∇f(wt), where i is sampled uniformly at random from Pk.

If we had only one node, i.e., K = 1, it is clear that we would have E[G1] = ∇f(wt). If K is more than 1, the values of Gk are in general biased estimates of ∇f(wt). We would like to achieve the following:), for some choice of αk. This is motivated by the general desire to make stochastic first-order methods to make a gradient step in expectation. We have

*.*

By setting, we get

*.*

This motivates the aggregation of updates from nodes proportional to nk, the number of data points available locally (Point 2).

Next, we realize that if the local data sizes, nk, are not identical, we likely don&apos;t want to do the same number of local iterations on each node k. Intuitively, doing one pass through data (or a fixed number of passes) makes sense. As a result, the aggregation motivated above does not make perfect sense anymore. Nevertheless, we can even it out, by setting the stepsize hk inversely proportional to nk, making sure each node makes progress of roughly the same magnitude overall. Hence, hk = h/nk (Point 1).

To motivate the Point 3, scaling of stochastic gradients by diagonal matrix Sk, consider the following example. We have 1,000,000 data points, distributed across K = 1,000 nodes. When we look at a particular feature of the data points, we observe it is non-zero only in 1,000 of them. Moreover, all of them happen to be stored on a single node, that stores only these 1,000 data points. Sampling a data point from this node and evaluating the corresponding gradient, will clearly yield an estimate of the gradient ∇f(w) with 1000-times larger magnitude. This would not necessarily be a problem if done only once. However, repeatedly sampling and overshooting the magnitude of the gradient will likely cause the iterative process to diverge quickly.

Hence, we scale the stochastic gradients by a diagonal matrix. This can be seen as an attempt to enforce the estimates of the gradient to be of the correct magnitude, conditioned on us, algorithm designers, being aware of the structure of distribution of the sparsity pattern.

Let us now highlight some properties of the modification in Point 4. Without any extra information, or in the case of fully dense data, averaging the local updates is the only way that actually makes sense — because each node outputs approximate solution of a proxy to the overall objective, and there is no induced separability structure in the outputs such as in CoCoA [57]. However, we could do much more in the other extreme. If the sparsity structure is such that each data point only depends on one of disjoint groups of variables, and the data were distributed according to this structure, we would efficiently have several disjoint problems. Solving each of them locally, and adding up the results would solve the problem in single iteration — desired algorithm property (C).

What we propose is an interpolation between these two settings, on a per-variable basis. If a variable appears in data on each node, we are going to take average. However, the less nodes a particular variable appear on, the more we want to trust those few nodes in informing us about the meaningful update to this variable — or alternatively, take a longer step. Hence the per-variable scaling of aggregated updates.

## 3.7         Further Notes

Looking at the Proposition 1, we identify equivalence of two algorithms, take the second one and try modify it to make it suitable for the setting of federated optimization. A question naturally arise: Is it possible to achieve the same by modifying the first algorithm suitable for federated optimization — by only altering the local optimization objective?

We indeed tried to experiment with idea, but we don&apos;t report the details for two reasons. First, the requirement of exact solution of the local subproblem is often impractical. Relaxing it gradually moves us to the setting we presented in the previous sections. But more importantly, using this approach we have only managed to get results significantly inferior to those reported later in the Experiments section.

# 4           Experiments

In this section we present the first experimental results in the setting of federated optimization. In particular, we provide results on a dataset based on public Google+ posts[6], clustered by user — simulating each user as a independent node. This preliminary experiment demonstrates why none of the existing algorithms are suitable for federated optimization, and the robustness of our proposed method to challenges arising there.

## 4.1         Predicting Comments on Public Google+ Posts

The dataset presented here was generated based on public Google+ posts. We randomly picked 10,000 authors that have at least 100 public posts in English, and try to predict whether a post will receive at least one comment (that is, a binary classification task).

We split the data chronologically on a per-author basis, taking the earlier 75% for training and the following 25% for testing. The total number of training examples is n = 2,166,693. We created a simple bag-of-words language model, based on the 20,000 most frequent words in dictionary based on all Google+ data. This results in a problem with dimension d = 20,002. The extra two features represent a bias term and variable for unknown word. We then use a logistic regression model to make a prediction based on these features.

We shape the distributed optimization problem as follows. Suppose that each user corresponds to one node, resulting in K = 10,000. The average nk, number of data points on node k is thus roughly 216. However, the actual numbers nk range from 75 to 9,000, showing the data is in fact substantially unbalanced.

It is natural to expect that different users can exhibit very different patterns in the data generated. This is indeed the case, and hence the distribution to nodes cannot be considered an IID sample from the overall distribution. Since we have a bag-of-words model, our data are very sparse — most posts contain only small fraction of all the words in the dictionary. This, together with the fact that the data are naturally clustered on a per-user basis, creates additional challenge that is not present in the traditional distributed setting.

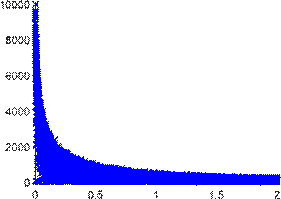
Figure 1 shows the frequency of different features across nodes. Some features are present everywhere, such as the bias term, while most features are relatively rare. In particular, over 88% of features are present on fewer than 1,000 nodes. However, this distribution does not necessarily resemble the overall appearance of the features in data examples. For instance, while an unknown word is present in data of almost every user, it is far from being contained in every data point.

**Naive prediction properties.** Before presenting the results, it is useful to look at some of the important basic prediction properties of the data. We use L2-regularized logistic regression, with regularization parameter λ = 1/n. We chose λ to be the best in terms of test error in the optimal solution.

• If one chooses to predict −1 (no comment), classification error is 33.16%.

• The optimal solution of the global logistic regression problem yields 26.27% test set error.

• Predicting the per-author majority from the training data yields 17.14% test error. That is, predict +1 or −1 for all the posts of an author, based on which label was more common in



×104

Figure 1: Features vs. appearance on nodes. The x-axis is a feature index, and the y-axis represents the number of nodes where a given feature is present.

that author&apos;s training data. This indicates that knowing the author is actually more useful than knowing what they said, which is perhaps not surprising.

In summary, this data is representative for our motivating application in federated optimization. It is possible to improve upon naive baseline using a fixed global model. Further, the per-author majority result suggests it is possible to improve further by adapting the global model to each user individually. Model personalization is common practice in industrial applications, and the techniques used to do this are orthogonal to the challenges of federated optimization. Exploring its performance is a natural next step, but beyond the scope of this work.

While we do not provide experiments for per user personalized models, we remark that this could be a good descriptor of how far from IID the data is distributed. Indeed, if each node has access to an IID sample, any adaptation to local data is merely over-fitting. However, if we can significantly improve upon the global model by per user/node adaptation, this means that the data available locally exhibit patterns specific to the particular node.

The performance of the Algorithm 4 is presented below. The only parameter that remains to be chosen by user is the stepsize h. We tried a set of stepsizes, and retrospectively choose one that works best — a typical practice in machine learning.

In Figure 2, we compare the following optimization algorithms[7]:

• The blue squares (OPT) represent the best possible offline value (the optimal value of the optimization task in the first plot, and the test error corresponding to the optimum in the second plot).

• The teal diamonds (GD) correspond to a simple distributed gradient descent.

• The purple triangles (COCOA) are for the CoCoA+ algorithm [57].

• The green circles (FSVRG) give values for our proposed algorithm.

• The red stars (FSVRGR) correspond to the same algorithm applied to the same problem with randomly reshuffled data. That is, we keep the unbalanced number of examples per node, but populate each node with randomly selected examples.

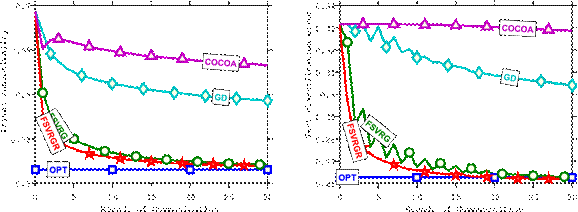


Figure 2: Rounds of communication vs. objective function (left) and test prediction error (right).

The first thing to notice is that CoCoA+ seems to be worse than trivial benchmark — distributed gradient descent. This behaviour can be predicted from theory, as the overall convergence rate directly depends on the best choice of aggregation parameter σ0. For sparse problems, it is upperbounded by the maximum of the values reported in Figure 1, which is K, and it is close to it also in practice. Althought it is expected that the algorithm could be modified to depend on average of these quantities (which could be orders of magnitude smaller), akin to coordinate descent algorithms [79], it has not been done yet. Note that other communication efficient algorithms fail to converge altogether.

The algorithm we propose, FSVRG, converges to optimal test classification accuracy in just 30 iterations. Recall that in the setting of federated optimization we introduced in Section 1.2, minimization of rounds of communication is the principal goal. However, concluding that the approach is stunningly superior to existing methods would not be completely fair nor correct. The conclusion is that the FSVRG is the first algorithm to tackle federated optimization, a problem that existing methods fail to generalize to. It is important to stress that none of the existing methods were designed with these particular challenges in mind, and we formulate the first benchmark.

Since the core reason other methods fail to converge is the non-IID data distribution, we test our method on the same problem, with data randomly reshuffled among the same number of nodes (FSVRGR; red stars). Since the difference in convergence is subtle, we can conclude that the techniques described in Section 3.6.2 serve its purpose and make the algorithm robust to challenges present in federated optimization.

This experiment demonstrates that learning from massively decentralized data, clustered on a per-user basis is indeed problem we can tackle in practice. Since the first version of this paper [45], additional experimental results were presented in [62]. We refer the reader to this paper for experiments in more challenging setting of deep learning, and a further discussion on how such system would be implemented in practice.

# 5           Conclusions and Future Challenges

We have introduced a new setting for distributed optimization, which we call federated optimization. This setting is motivated by the outlined vision, in which users do not send the data they generate to companies at all, but rather provide part of their computational power to be used to solve optimization problems. This comes with a unique set of challenges for distributed optimization. In particular, we argue that the massively distributed, non-IID, unbalanced, and sparse properties of federated optimization problems need to be addressed by the optimization community.

We explain why existing methods are not applicable or effective in this setting. Even the distributed algorithms that can be applied converge very slowly in the presence of large number of nodes on which the data are stored. We demonstrate that in practice, it is possible to design algorithms that work surprisingly efficiently in the challenging setting of federated optimization, which makes the vision conceptually feasible.

We realize that it is important to scale stochastic gradients on a per-coordinate basis, differently on each node to improve performance. To the best of our knowledge, this is the first time such per-node scaling has been used in distributed optimization. Additionally, we use per-coordinate aggregation of updates from each node, based on distribution of the sparsity patterns in the data.

Even though our results are encouraging, there is a lot of room for future work. One natural direction is to consider fully asynchronous versions of our algorithms, where the updates are applied as soon as they arrive. Another is developing a better theoretical understanding of our algorithm, as we believe that development of a strong understanding of the convergence properties will drive further research in this area.

Study of the federated optimization problem for non-convex objectives is another important avenue of research. In particular, neural networks are the most important example of a machine learning tool that yields non-convex functions fi, without any convenient general structure. Consequently, there are no useful results describing convergence guarantees of optimization algorithms. Despite the lack of theoretical understanding, neural networks are now state-of-the-art in many application areas, ranging from natural language understanding to visual object detection. Such applications arise naturally in federated optimization settings, and so extending our work to such problems is an important direction.

The non-IID data distribution assumed in federated optimization, and mobile applications in particular, suggest that one should consider the problem of training a personalized model together with that of learning a global model. That is, if there is enough data available on a given node, and we assume that data is drawn from the same distribution as future test examples for that node, it may be preferable to make predictions based on a personalized model that is biased toward good performance on the local data, rather than simply using the global model.

# References

[1] Mart´ın Abadi, Andy Chu, Ian Goodfellow, Brendan H McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. arXiv:1607.00133, 2016.

[2] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In Advances in Neural Information Processing Systems, pages 873–881, 2011.

[3] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. arXiv:1603.05953, 2016.

[4] Zeyuan Allen-Zhu, Yang Yuan, and Karthik Sridharan. Exploiting the structure: Stochastic gradient methods using raw clusters. arXiv:1602.02151, 2016.

[5] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In Advances in Neural Information Processing Systems, pages 1756–1764, 2015.

[6] Ron Bekkerman, Mikhail Bilenko, and John Langford. Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press, 2011.

[7] Dimitri P Bertsekas. Distributed asynchronous computation of fixed points. Mathematical Programming, 27(1):107–120, 1983.

[8] Dimitri P Bertsekas and John N Tsitsiklis. Parallel and distributed computation: numerical methods. PrenticeHall, Inc., 1989.

[9] Antoine Bordes, L´eon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-Newton stochastic gradient descent. The Journal of Machine Learning Research, 10:1737–1754, 2009.

[10] L´eon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In Proceedings of the symposium on learning and data science, 2009.

[11] L´eon Bottou. Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade, pages 421–436. Springer, 2012.

[12] L´eon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. arXiv:1606.04838, 2016.

[13] Olivier Bousquet and L´eon Bottou. The tradeoffs of large scale learning. In Advances in Neural Information Processing Systems, pages 161–168, 2008.

[14] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends R , 3(1):1–122, 2011.*in Machine Learning*

[15] Joseph Bradley, Aapo Kyrola, Daniel Bickson, and Carlos Guestrin. Parallel coordinate descent for l1regularized loss minimization. In Proceedings of the 28th International Conference on Machine Learning, pages 321–328, 2011.

[16] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-Newton method for large-scale optimization. SIAM Journal on Optimization, 26(2):1008–1031, 2016.

[17] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. Journal of Machine Learning Research, 12:1069–1109, 2011.

[18] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 571–582, 2014.

[19] CNN. Where (and when) do you use your smartphone: Bedroom? Church? News article http://edition. cnn.com/2013/07/13/tech/smartphone-use-survey/, 2013.

[20] Dominik Csiba and Peter Richta´rik. Primal method for ERM with flexible mini-batching schemes and nonconvex losses. arXiv:1506.02227, 2015.

[21] Dominik Csiba and Peter Richta´rik. Coordinate descent face-off: primal or dual? arXiv:1605.08982, 2016.

[22] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher R´e. Taming the wild: A unified analysis of hogwild-style algorithms. In Advances in Neural Information Processing Systems, pages 2656–2664, 2015.

[23] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In Advances in Neural Information Processing Systems, pages 1223–1231, 2012.

[24] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[25] Aaron Defazio. A simple practical accelerated method for finite sums. arXiv preprint arXiv:1602.02442, 2016.

[26] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In Advances in Neural Information Processing Systems, pages 1646–1654, 2014.

[27] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. The Journal of Machine Learning Research, 13(1):165–202, 2012.

[28] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. Journal of Scientific Computing, 66(3):889–916, 2016.

[29] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. Automatic control, IEEE Transactions on, 57(3):592–606, 2012.

[30] John C Duchi, Sorathan Chaturapruek, and Christopher R´e. Asynchronous stochastic convex optimization. arXiv:1508.00882, 2015.

[31] John C Duchi, Michael I Jordan, and Brendan H McMahan. Estimation, optimization, and parallelism when data is sparse. In Advances in Neural Information Processing Systems, pages 2832–2840, 2013.

[32] John C Duchi, Michael I Jordan, and Martin J Wainwright. Privacy aware learning. Journal of the Association for Computing Machinery, 2014.

[33] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2014.

[34] Olivier Fercoq, Zheng Qu, Peter Richta´rik, and Martin Taka´ˇc. Fast distributed coordinate descent for nonstrongly convex losses. In Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on, pages 1–6. IEEE, 2014.

[35] Olivier Fercoq and Peter Richta´rik. Accelerated, parallel, and proximal coordinate descent. SIAM Journal on Optimization, 25(4):1997–2023, 2015.

[36] The MPI Forum. MPI: A message passing interface standard, Version 3.1. Document available at http:

//www.mpi-forum.org/, 2015.

[37] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015.

[38] Robert Mansel Gower, Donald Goldfarb, and Peter Richta´rik. Stochastic block BFGS: squeezing more curvature out of data. In Proceedings of The 33rd International Conference on Machine Learning, pages 1869–1878, 2016.

[39] Robert Mansel Gower and Peter Richta´rik. Randomized quasi-Newton updates are linearly convergent matrix inversion algorithms. arXiv:1602.01768, pages 1869–1878, 2016.

[40] Mert Gu¨rbu¨zbalaban, Asu Ozdaglar, and Pablo Parrilo. Why random reshuffling beats stochastic gradient descent. arXiv:1510.08560, 2015.

[41] Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Koneˇcny´, and Scott Sallinen. Stop wasting my gradients: Practical SVRG. In Advances in Neural Information Processing Systems, pages 2251–2259, 2015.

[42] Martin Jaggi, Virginia Smith, Martin Taka´ˇc, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In Advances in Neural Information Processing Systems, pages 3068–3076, 2014.

[43] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in Neural Information Processing Systems, pages 315–323, 2013.

[44] Jakub Koneˇcny´, Jie Liu, Peter Richta´rik, and Martin Taka´ˇc. Mini-batch semi-stochastic gradient descent in the proximal setting. IEEE Journal of Selected Topics in Signal Processing, 10(2):242–255, 2016.

[45] Jakub Koneˇcny´, Brendan H McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. arXiv:1511.03575, 2015.

[46] Jakub Koneˇcny´, Zheng Qu, and Peter Richta´rik. Semi-stochastic coordinate descent. arXiv:1412.6293, 2014.

[47] Jakub Koneˇcny´ and Peter Richta´rik. Semi-stochastic gradient descent methods. arXiv:1312.1666, 2013.

[48] R´emi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. ASAGA: Asynchronous parallel saga. arXiv:1606.04809, 2016.

[49] Yann A LeCun, L´eon Bottou, Genevieve B Orr, and Klaus-Robert Mu¨ller. Efficient backprop. In Neural networks: Tricks of the trade, pages 9–48. Springer, 2012.

[50] Jason Lee, Tengyu Ma, and Qihang Lin. Distributed stochastic variance reduced gradient methods.

*arXiv:1507.07595*, 2015.

[51] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, pages 147–156. IEEE, 2013.

[52] Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, and Chih-Jen Lin. Large-scale logistic regression and linear support vector machines using spark. In Big Data (Big Data), 2014 IEEE International Conference on, pages 519–528. IEEE, 2014.

[53] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In Advances in Neural Information Processing Systems, pages 3366–3374, 2015.

[54] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1-3):503–528, 1989.

[55] Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. SIAM Journal on Optimization, 25(1):351–376, 2015.

[56] Ji Liu, Stephen J Wright, Christopher R´e, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. Journal of Machine Learning Research, 16:285–322, 2015.

[57] Chenxin Ma, Jakub Koneˇcny´, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richta´rik, and Martin Taka´ˇc. Distributed optimization with arbitrary local solvers. arXiv:1512.04039, 2015.

[58] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael Jordan, Peter Richta´rik, and Martin Taka´ˇc. Adding vs.

averaging in distributed primal-dual optimization. In Proceedings of The 32nd International Conference on Machine Learning, pages 1973–1982, 2015.

[59] Dhruv Mahajan, Nikunj Agrawal, S Sathiya Keerthi, S Sundararajan, and Leon Bottou. An efficient distributed learning algorithm based on effective local functional approximations. arXiv preprint arXiv:1310.8418, 2013.

[60] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. arXiv:1507.06970, 2015.

[61] Jakub Mareˇcek, Peter Richta´rik, and Martin Taka´ˇc. Distributed block coordinate descent for minimizing partially separable functions. In Numerical Analysis and Optimization, pages 261–288. Springer, 2015.

[62] Brendan H McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas. Federated learning of deep networks using model averaging. arXiv:1602.05629, 2016.

[63] Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic l-bfgs algorithm. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, pages 249–258, 2016.

[64] Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In Advances in Neural Information Processing Systems, pages 451–459, 2011.

[65] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In Advances in Neural Information Processing Systems, pages 1017–1025, 2014.

[66] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. SIAM Journal on Optimization, 19(4):1574–1609, 2009.

[67] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM Journal on Optimization, 22(2):341–362, 2012.

[68] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o(1/k2). Soviet Mathematics Doklady, 27(2):372–376, 1983.

[69] Yurii Nesterov. Introductory Lectures on Convex Optimization. A Basic Course. Kluwer, 2004.

[70] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In Proceedings of the 28th International Conference on Machine Learning, pages 265–272, 2011.

[71] Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 693–701, 2011.

[72] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. arXiv:1506.02396, 2015.

[73] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.

[74] Zheng Qu, Peter Richt´arik, Martin Taka´ˇc, and Olivier Fercoq. SDNA: Stochastic dual newton ascent for empirical risk minimization. In Proceedings of The 33rd International Conference on Machine Learning, pages 1823–1832, 2016.

[75] Zheng Qu, Peter Richta´rik, and Tong Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In Advances in Neural Information Processing Systems, volume 28, pages 865–873, 2015.

[76] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnaba´s Po´czo´s, and Alex Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In Advances in Neural Information Processing Systems, pages 2647–2655, 2015.

[77] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnaba´s Po´czo´s, and Alex Smola. Stochastic variance reduction for nonconvex optimization. arXiv:1603.06160, 2016.

[78] Sashank J Reddi, Jakub Koneˇcny´, Peter Richta´rik, Barnaba´s Po´czo´s, and Alex Smola. AIDE: Fast and communication efficient distributed optimization. arXiv:1608.06879, 2016.

[79] Peter Richt´arik and Martin Taka´ˇc. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Mathematical Programming, 144(1-2):1–38, 2014.

[80] Peter Richta´rik and Martin Taka´ˇc. Parallel coordinate descent methods for big data optimization. Mathematical Programming, 156(1):433–484, 2016.

[81] Peter Richta´rik and Martin Tak´aˇc. Distributed coordinate descent method for learning with big data. Journal of Machine Learning Research, 17(75):1–25, 2016.

[82] Herbert Robbins and Sutton Monro. A stochastic approximation method. The Annals of Mathematical Statistics, pages 400–407, 1951.

[83] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In Advances in Neural Information Processing Systems, pages 2663–2671, 2012.

[84] Mark Schmidt, Reza Babanezhad, Mohamed Ahmed, Aaron Defazio, Ann Clifton, and Anoop Sarkar. Nonuniform stochastic average gradient method for training conditional random fields. In Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, pages 819–828, 2015.

[85] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. arXiv:1309.2388, 2013.

[86] Shai Shalev-Shwartz. SDCA without duality, regularization, and individual convexity. arXiv:1602.01582, 2016.

[87] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated subgradient solver for svm. Mathematical programming, 127(1):3–30, 2011.

[88] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. The Journal of Machine Learning Research, 14(1):567–599, 2013.

[89] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In High Performance Computing for Computational Science–VECPAR 2010, pages 1–25. Springer, 2011.

[90] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In Communication, Control and Computing (Allerton), 2014 52nd Annual Allerton Conference on, pages 850–857. IEEE, 2014.

[91] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In Proceedings of the 31st International Conference on Machine Learning, pages 1000–1008, 2014.

[92] Virginia Smith, Simone Forte, Michael I Jordan, and Martin Jaggi. L1-regularized distributed optimization: A communication-efficient primal-dual framework. arXiv:1512.04011, 2015.

[93] Martin Taka´ˇc, Avleen Bijral, Peter Richta´rik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. In Proceedings of the 30th International Conference on Machine Learning, 2013.

[94] Martin Taka´ˇc, Peter Richta´rik, and Nathan Srebro. Distributed mini-batch SDCA. arXiv:1507.08322, 2015.

[95] John Nikolas Tsitsiklis. Problems in decentralized decision making and computation. Technical report, DTIC Document, 1984.

[96] Vladimir N Vapnik. An overview of statistical learning theory. Neural Networks, IEEE Transactions on, 10(5):988–999, 1999.

[97] Huahua Wang and Arindam Banerjee. Randomized block coordinate descent for online and stochastic optimization. arXiv:1407.0107, 2014.

[98] White House Report. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. Journal of Privacy and Confidentiality, 2013.

[99] Blake Woodworth and Nathan Srebro. Tight complexity bounds for optimizing composite objectives. arXiv:1605.08003, 2016.

[100] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. SIAM Journal on Optimization, 24(4):2057–2075, 2014.

[101] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In Advances in Neural Information Processing Systems, pages 629–637, 2013.

[102] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, volume 10, page 10, 2010.

[103] Yuchen Zhang, John Duchi, Michael I Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In Advances in Neural Information Processing Systems, pages 2328–2336, 2013.

[104] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Communication-efficient algorithms for statistical optimization. Journal of Machine Learning Research, 14:3321–3363, 2013.

[105] Yuchen Zhang and Xiao Lin. DiSCO: Distributed optimization for self-concordant empirical loss. In Proceedings of The 32th International Conference on Machine Learning, pages 362–370, 2015.

[106] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. Distributed newton methods for regularized logistic regression. In Advances in Knowledge Discovery and Data Mining, pages 690–703. Springer, 2015.

[107] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 2595–2603, 2010.

# A Distributed Optimization via Quadratic Perturbations

This appendix follows from the discussion motivating DANE algorithm by a general algorithmic perturbation template (9) for λ-strongly convex objectives. We use this to propose a similar but new method, which unlike DANE converges under arbitrary data partitioning , and we highlight its relation to the dual CoCoA algorithm for distributed optimization.

For simplicity and ease of drawing the above connections we assume that nk is identical for all k ∈ {1,2,...,K} throughout the appendix. All the arguments can be simply extended, but would unnecessarily complicate the notation for current purpose.

## A.1 New Method

We now present a new method (Algorithm 5), which also belongs to the family of quadratic perturbation methods (9). However, the perturbation vectors atk are different from those of DANE. In particular, we set

*atk* def= ∇F(w) − (η∇F(w) + g),*ktktkt*

where η > 0 is a parameter, and the vectors gkt are maintained by the method. As we show in Lemma 4, Algorithm 5 satisfies

for all iterations t. This implies that ). That is, both DANE and the new method use a linear perturbation which, when averaged over the nodes, involves the gradient of the objective function f at the latest iterate wt. Therefore, the methods have one more property in common beyond both being of the form (9). However, as we shall see in the rest of this section, Algorithm 5 allows an insightful dual interpretation. Moreover, while DANE may not converge for arbitrary problems (even when restricted to ridge regression)—and is only known to converge under the assumption that the data stored on each node are in some precise way similar, Algorithm 5 converges for any ridge regression problem and any data partitioning.

Let us denote by Xk the matrix obtained by stacking the data points xi as column vectors for all i ∈ Pk. We have the following Lemma.

**Lemma 4.** *For all t* ≥ 0 we have.

*Proof.* The statement holds for t = 0. Indeed,

*,*

where the last step follows from the definition of w0. Assume now that the statement hold for t. Then

*.*

The first equation follows from the way gk is updated in the algorithm. The second equation follows from the inductive assumption, and the last equation follows from the definition of wt+1 in the algorithm. 

**Algorithm 5** Primal Method



1: ∈ [1,K]**Input:** *σ*

2: ∈ Rfor k = 1,2,...,K 3: **Choose: Set:***αk*0 |P*k*| 

4: **Set:**

5: **Set:**

6: = 0,1,2,... do**for** *t*

7: = 1 to K do**for** *k*

8: *wkt*+1 = argmin

9: **end for**

10: 

11: = 1 to K do**for** *k*

12: +1 = g+ λη(w+1 − w+1)*gktkt ktt*

13: **end for**

14: **return** *wt*

15: **end for**



## A.2 L2-Regularized Linear Predictors

|  |
| --- |
|  |
|  |  |

In the rest of this section we consider the case of L2-regularized linear predictors. That is, we focus on problem (1) with fi of the form

*,*

where λ > 0 is a regularization parameter. This leads to L2 regularized empirical risk minimization (ERM) problem

*.* (11)

We assume that the loss functions φi : R → R are convex and 1/γ-smooth for some γ > 0; these are standard assumptions. As usual, we allow the loss function φi to depend on the label yi. For instance, we may choose the quadratic loss: (for which γ = 1).

Let X = [x1,...,xn] ∈ R*d*×n. As described in Section 3.3, we assume that the data ( is distributed among K nodes of a computer cluster as follows: node k = 1,2,...,K contains pairs (x,yii) for i ∈ Pk, where P1,...,PK forms a partition of the set [n] = {1,2,...,n}. Letting X = [X1,...,XK], where Xk ∈ Ris a submatrix of A corresponding to columns i ∈ Pk, and yk ∈ Ris the subvector of y corresponding to entries i ∈ Pk. Hence, node k contains the pair (Xk,yk). With this notation, we can write the problem in the form (8), where*d*×|P*k*| |P*k*|

*.* (12)

## A.3 A Dual Method: Dual Block Proximal Gradient Ascent

The dual of (11) is the problem

*,* (13)

where φ∗i is the convex conjugate of φi. Since we assume that φi is 1/γ smooth, it follows that φ∗i is γ strongly convex. Therefore, D is a strongly concave function.

**From dual solution to a primal solution.** It is well known that if α∗ is the optimal solution of the dual problem (11), then w∗ def= λn1 Xα∗ is the optimal solution of the primal problem. Therefore, for any dual algorithm producing a sequence of iterates αt, we can define a corresponding primal algorithm via the linear mapping

*.* (14)

Clearly, if αt → α∗, then wt → w∗. We shall now design a method for maximizing the dual function D and then in Theorem 5 we claim that for quadratic loss functions, Algorithm 5 arises as an image, defined via (14), of dual iterations of this dual ascent method.

**Design of the dual gradient ascent method.** Let. Since ξ is a convex

quadratic, we have

*,*

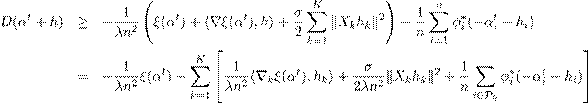
where ∇ξ(α) = XXα Tand ∇2ξ(α) = XXT. Further, we define the block-diagonal matrix B def= ), and a norm associate with this matrix:

*K*

khk2B def= XkXk2*khk.*

*k*=1

By σ we refer to a large enough constant for which. In order to avoid unnecessary technicalities, we shall assume that the matrices XXkTk are positive definite, which implies that k·kB is a norm. It can be shown that 1 ≤ σ ≤ K. Clearly, ξ is σ-smooth with respect to the norm k · kB. In view of the above, for all h ∈ Rwe can estimate D from below as follows:*n*

*,*

where ∇kξ(αt) corresponds to the subvector of ∇ξ(αt) formed by entries i ∈ Pk.

We now let) be the maximizer of this lower bound. Since the lower bound is separable in the blocks {htk}k, we can simply set

                                             

:= arg min . (15)

*u*  *i*∈Pk 

|  |  |
| --- | --- |
| **Algorithm 6** Dual Method |  |
| 1: ∈ [1,K]**Input:** *σ*  2: ∈ Rfor k = 1,2,...,K**Choose:** *αk*0 |P*k*|  3: = 0,1,2,... do**for** *t*  4: = 1 to K do**for** *k*  5: +1 = argmin∈R|Pk| (u)*htkuDkt*  6: **end for**  7: +1 = α+ h*αtt t*  8: **end for**  9: **return** *wt* | *.* See (15) |

Having computed for all k, we can setfor all k, or equivalently, αt+1 = αt + ht. This is formalized as Algorithm 6. Algorithm 6 is a proximal gradient ascent method applied to the dual problem, with smoothness being measured using the block norm khkB. It is known that gradient ascent converges at a linear rate for smooth and strongly convex (for minimization problems) objectives.

One of the main insights of this section is the following equivalence result.

**Theorem 5** (Equivalence of Algorithms 5 and 6 for Quadratic Loss). Consider the ridge regression problem. That is, set for all i. Assume is chosen in the same way in Algorithms 5 and 6. Then the dual iterates αand the primal iterates wproduced by the two algorithms are related via t t (14) for all t ≥ 0.

Since the dual method converges linearly, in view of the above theorem, so does the primal method. Here we only remark that the popular algorithm CoCoA+ [57] arises if Step 5 in Algorithm 6 is done inexactly. Hence, we show that duality provides a deep relationship between the CoCoA+ and DANE algorithms, which were previously considered completely different.

## A.4 Proof of Theorem 5

In this part we prove the theorem.

**Primal and Dual Problems.** Since , the primal problem (11) is a ridge regression problem of the form

*,* (16)

where X ∈ Rand y ∈ R. In view of (13), the dual of (16) is*d*×n *n*

(17)

**Primal Problem: Distributed Setup.** The primal objective function is of the form (8), where

in view of (12), we have. Therefore,

(18)

and

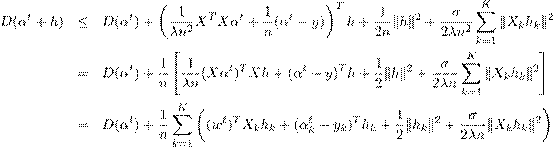
**Dual Method.** Since D is a quadratic, we have



with

*,* 

We know that). With this approximation, for all h ∈ Rwe can estimate D from above by a node-separable quadratic function as follows:*n*

*.*

Next, we shall define

= arg min (19)

|  |  |  |
| --- | --- | --- |
| for k = 1,2,...,K and then set |  |  |
|  | *αt*+1 = α+ h*t t.* | (20) |

**Primal Version of the Dual Method.** Note that (19) has the same form as (17), with X replaced by Xk, λ replaced by λ/σ and y replaced by. Hence, we know that

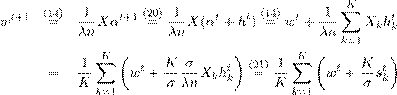
(21)

is the optimal solution of the primal problem of (22):

                                                              *stk* = arg min . (22)

*s*

Hence, the primal version of method (20) is given by

*.*

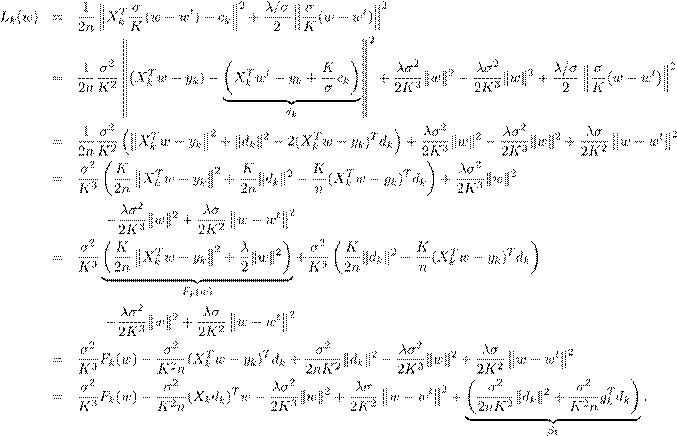
With the change of variables (i.e., s = Kσ (w − wt)), from (22) we know that solves

                      *wkt*+1 = arg min (23)

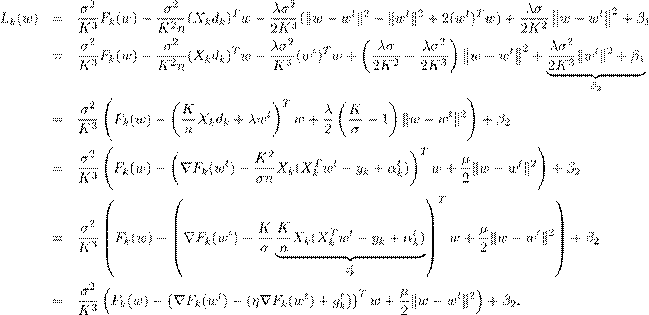
*w*

and.

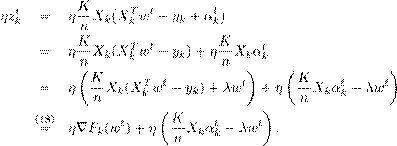
Let us now rewrite the function in (23) so as to connect it to Algorithm 5:



Next, since kwk2 = kw − wtk2 − kwtk2 + 2(wt)Tw, we can further write



where the last step follows from the claim that ηzkt = η∇Fk(wt) + gkt. We now prove the claim. First, we have



Due to the definition of gk0 in Step 5 of Algorithm 5 as), we observe that the claim holds for t = 0. If we show that



for all t ≥ 0, then we are done. This can be shown by induction. This finishes the proof of Theorem 5.

[[1]](" \l "_ftnref1" \o ") The same algorithm was simultaneously introduced as Semi-Stochastic Gradient Descent (S2GD) [47]. Since the former work gained more attention, we will for clarity use the name SVRG throughout this paper.

[[2]](" \l "_ftnref2" \o ") See [13, Section 2.3] for their definition of large scale learning problem.

[[3]](" \l "_ftnref3" \o ") It should be noted that some of the works presented in this section were originally presented as parallel algorithms.

We include them anyway as many of the general ideas in carry over to the distributed setting. 4Considering only algorithms that can be run on a given machine.

[[4]](" \l "_ftnref4" \o ") A bound on the delay τ can be deterministic or probabilistic. However, in practice, the delays are mostly about the number of nodes in the network, and there rare very long delays, when a variety of operating system-related events can temporarily postpone computation of a single node. To the best of our knowledge, no formal assumptions reflect this setting well. In fact, two recent works [60, 48] highlight subtle but important issue with labelling of iterates in the presence of asynchrony, rendering most of the existing analyses of asynchronous optimization algorithms incorrect.

[[5]](" \l "_ftnref5" \o ") This is valid only for generalized linear models.

[[6]](" \l "_ftnref6" \o ") The posts were public at the time the experiment was performed, but since a user may decide to delete the post or make it non-public, we cannot release (or even permanently store) any copies of the data.

[[7]](" \l "_ftnref7" \o ") We thank Mark Schmidt for his prettyPlot function, available on his website.