Distributed Learning Systems with First-Order Methods

An Introduction

Other titles in Foundations and Trends® in Databases

Algorithmic Aspects of Parallel Data Processing
Paraschos Koutris, Semih Salihoglu and Dan Suciu
ISBN: 978-1-68083-406-2

Data Infrastructure for Medical Research
Thomas Heinis and Anastasia Ailamaki
ISBN: 978-1-68083-348-5

Main Memory Database Systems
Franz Faerber, Alfons Kemper, Per-Ake Larson, Justin Levandoski, Thomas Neumann and Andrew Pavlo
ISBN: 978-1-68083-324-9

Query Processing on Probabilistic Data: A Survey
Guy Van den Broeck and Dan Suciu
ISBN: 978-1-68083-314-0

Big Graph Analytics Platforms
Da Yan, Yingyi Bu, Yuanyuan Tian and Amol Deshpande
978-1-68083-242-6

Distributed Learning Systems with First-Order Methods

An Introduction

Ji Liu
University of Rochester and
Kuaishou Inc., USA
ji.liu.uwisc@gmail.com

Ce Zhang
ETH Zurich, Switzerland
ce.zhang@inf.ethz.ch

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Databases

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

J. Liu and C. Zhang. *Distributed Learning Systems with First-Order Methods*. Foundations and Trends® in Databases, vol. 9, no. 1, pp. 1–100, 2020.

ISBN: 978-1-68083-701-8

Foundations and Trends® in Databases
Volume 9, Issue 1, 2020

Editorial Board

Editorial Scope

Topics

Foundations and Trends® in Databases publishes survey and tutorial articles in the following topics:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2020, Volume 9, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Contents

Distributed Learning Systems with First-Order Methods

Ji Liu[1] and Ce Zhang[2]

[1] *University of Rochester and Kuaishou Inc., USA; ji.liu.uwisc@gmail.com*
[2] *ETH Zurich, Switzerland; ce.zhang@inf.ethz.ch*

ABSTRACT

Scalable and efficient distributed learning is one of the main driving forces behind the recent rapid advancement of machine learning and artificial intelligence. One prominent feature of this topic is that recent progress has been made by researchers in *two* communities: (1) *the system community* such as database, data management, and distributed systems, and (2) *the machine learning and mathematical optimization community*. The interaction and knowledge sharing between these two communities has led to the rapid development of new distributed learning systems and theory.

In this monograph, we hope to provide a brief introduction of some distributed learning techniques that have recently been developed, namely *lossy communication compression* (e.g., quantization and sparsification), *asynchronous communication*, and *decentralized communication*. One special focus in this monograph is on making sure that it can be easily understood by researchers in *both* communities — on the system side, we rely on a simplified system model hiding many system details that are not necessary for the intuition behind the system speedups; while, on the theory side, we

Ji Liu and Ce Zhang (2020), "Distributed Learning Systems with First-Order Methods", Foundations and Trends® in Databases: Vol. 9, No. 1, pp 1–100. DOI: 10.1561/1900000062.

rely on minimal assumptions and significantly simplify the proof of some recent work to achieve comparable results.

Notations and Definitions

Throughout this monograph, we make the following definitions.

- All vectors are assumed to be column vectors by default.
- α, β, and γ usually denote constants.
- Bold lowercase letters usually denote vectors, such as $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{v}$.
- $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$.
- Capital letters usually denote matrices, such as W.
- $\lesssim$ means "small and equal to up to a constant factor", for example, $a_t \lesssim b_t$ means that there exists a constant $\alpha > 0$ independent of t such that $a_t \leq \alpha b_t$.
- $\mathbf{1}$ denotes a vector with 1 at everywhere and its dimension depends on the context.
- $f'(\cdot)$ denotes the gradient or differential of the function $f(\cdot)$.
- $[M] := \{1, 2, \ldots, M\}$ denotes a set containing integers from 1 to M.

1

Introduction

Real-world distributed learning systems, especially those relying on first-order methods, are often constructed in two "phases" — first comes the textbook (stochastic) gradient descent (SGD) algorithm, and then certain aspects of the system design are "relaxed" to remove the system bottleneck, be that communication bandwidth, latency, synchronization cost, etc. Throughout this work, we will describe multiple popular ways of system relaxation developed in recent years and analyze their system behaviors and theoretical guarantees.

In this section, we provide the background for both the theory and the system. On the theory side, we describe the intuition and theoretical properties of standard gradient descent (GD) and stochastic gradient descent (SGD) algorithms (we refer the reader to Bottou *et al.*, 2016 for more details). On the system side, we introduce a simplified performance model that hides many details but is just sophisticated enough for us to reason about the performance impact of the different system relaxation techniques that we will introduce in the later sections.

Summary of Results. In this monograph, we focus on three different system relaxation techniques, namely *lossy communication compression*,

Table 1.1: Summary of results covered in this monograph. For distributed settings, we assume that there are N workers and the latency and the bandwidth of the network are α and β, respectively. The lossy compression scheme has a compression ratio of $\eta(< 1)$ (which introduces additional variance σ' to the gradient estimator) and the decentralized communication scheme uses a communication graph g of degree $\deg(G)$. ς measures the data variation among workers in the decentralized scenario — $\varsigma = 0$ if all workers have the same dataset. We assume the simplified communication model and communication pattern as described in Subsection 1.3

Algorithm	System Optimization	# Iterations to ϵ	Communication Cost
`GD`	/	$O\left(\frac{1}{\epsilon}\right)$	N/A
`SGD`	/	$O\left(\frac{1}{\epsilon} + \frac{\sigma^2}{\epsilon^2}\right)$	N/A
`mb-SGD`	Distributed Baseline	$O\left(\frac{1}{\epsilon} + \frac{\sigma^2}{N\epsilon^2}\right)$	$O(N\alpha + \beta)$
`CSGD`	Compression	$O\left(\frac{1}{\epsilon} + \frac{\sigma^2}{N\epsilon^2} + \frac{\sigma'^2}{\epsilon^2}\right)$	$O(N\alpha + \beta\eta)$
`EC-SGD`	Compression	$O\left(\frac{1}{\epsilon} + \frac{\sigma^2}{N\epsilon^2} + \frac{\sigma'}{\epsilon^{2/3}}\right)$	$O(N\alpha + \beta\eta)$
`ASGD`	Asynchronization	$O\left(\frac{N}{\epsilon} + \frac{\sigma^2}{N\epsilon^2}\right)$	$O(N\alpha + \beta)$
`DSGD`	Decentralization	$O\left(\frac{1}{\epsilon} + \frac{\sigma^2}{N\epsilon^2} + \frac{\varsigma}{(1-\rho)\epsilon^{2/3}}\right)$	$O(\deg(G)(\alpha + \beta))$

asynchronous communication, and *decentralized communication*. For each system relaxation technique, we study their convergence behavior (i.e., # iterations we need to achieve ϵ precision) and the communication cost per iteration. Table 1.1 summarizes the results we will cover in this monograph.

1.1 Gradient Descent

Let us consider the generic machine learning objective that can be summarized by the following form

$$\min_{\mathbf{x}\in\mathbb{R}^d} \quad \left\{ f(\mathbf{x}) := \frac{1}{M}\sum_{m=1}^{M} F_m(\mathbf{x}) \right\}. \tag{1.1}$$

Let $f^\star := \min_{\mathbf{x}} f(\mathbf{x})$ and assume that it exists by default. Each F_m corresponds to a *data sample* in the context of machine learning.

The gradient descent (GD) can be described as

$$(\texttt{GD}) \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \gamma f'(\mathbf{x}_t) \tag{1.2}$$

where t is the iteration index and $f'(\mathbf{x}_t)$ is the gradient of f at $\mathbf{x}_t$.

1.1.1 Intuitions

We provide two intuitions about the gradient descent GD algorithm to indicate why it will work.

Steepest Descent Direction. The gradient (or a differential) of a function is the steepest direction to increase the function value given an infinitely small step, which can be seen from the property of the function gradient $\forall \|\mathbf{v}\| = 1$

$$\langle f'(\mathbf{x}), \mathbf{v} \rangle = f'_{\mathbf{v}}(\mathbf{x}) := \lim_{\delta \to 0} \frac{f(\mathbf{x} + \mathbf{v}\delta) - f(\mathbf{x})}{\delta}.$$

$f'_{\mathbf{v}}(\mathbf{x})$ is the directional gradient, which indicates how much increment there is on function value along the direction $\mathbf{v}$ by a tiny unit step. To find the steepest unit descent direction is to maximize

$$\max_{\|\mathbf{v}\|=1} \quad f'_{\mathbf{v}}(\mathbf{x}).$$

Since $f'_{\mathbf{v}}(\mathbf{x}) = \langle f'(\mathbf{x}), \mathbf{v} \rangle$, it is easy to verify that the steepest direction is $\mathbf{v}^\star = \frac{f'(\mathbf{x})}{\|f'(\mathbf{x})\|}$. Note that our goal is to minimize the function value. Therefore, GD is a natural idea via moving the model $\mathbf{x}_t$ along the steepest "descent" direction $-f'(\mathbf{x}_t)$.

Minimizing a Model Function. Another perspective from which to view gradient descent is based on the model function. Since the original objective function $f(\mathbf{x})$ is usually very complicated, it is very hard to minimize the objective function directly. A straightforward idea is to construct a model function to locally approximate (at $\mathbf{x}_t$) the original objective in each iteration. The model function needs to be simple and to approximate the original function well enough. Therefore, the most natural idea is to choose a quadratic function (that is usually simple

to solve)

$$M_{\mathbf{x}_t,\gamma}(\mathbf{x}) := f(\mathbf{x}_t) + \langle f'(\mathbf{x}_t), \mathbf{x} - \mathbf{x}\rangle + \frac{1}{2\gamma}\|\mathbf{x} - \mathbf{x}_t\|^2.$$

This model function is a good approximation in the sense that

- $f(\mathbf{x}_t) = M_{\mathbf{x}_t,\gamma}(\mathbf{x}_t)$
- $f'(\mathbf{x}_t) = M'_{\mathbf{x}_t,\gamma}(\mathbf{x}_t)$
- $f(\cdot) \leq M_{\mathbf{x}_t,\gamma}(\cdot)$ if the learning rate γ is sufficiently small.

For the first two, it is easy to understand why they are important. The last one is important to the convergence, which will be seen soon. Figure 1.1 illustrates the geometry of the model function. One can verify that the GD algorithm is nothing but iteratively update the optimization variable $\mathbf{x}$ via minimizing the model function at the current point $\mathbf{x}_t$:

$$\begin{aligned}\mathbf{x}_{t+1} &= \operatorname*{argmin}_{\mathbf{x}} \; M_{\mathbf{x}_t,\gamma}(\mathbf{x}) \\ &= \operatorname*{argmin}_{\mathbf{x}} \; \frac{1}{2\gamma}\left\|\mathbf{x} - (\mathbf{x}_t - \gamma f'(\mathbf{x}_t))\right\|^2 + \text{constant} \\ &= \mathbf{x}_t - \gamma f'(\mathbf{x}_t).\end{aligned}$$

The convergence of GD can also be revealed by this intuition — $\mathbf{x}_{t+1}$ always improves $\mathbf{x}_t$ unless the gradient is zero

$$f(\mathbf{x}_{t+1}) \leq M_{\mathbf{x}_t,\gamma}(\mathbf{x}_{t+1}) \leq M_{\mathbf{x}_t,\gamma}(\mathbf{x}_t) = f(\mathbf{x}_t),$$

where $f(\mathbf{x}_{t=1}) = f(\mathbf{x}_t)$ holds if and only if $f'(\mathbf{x}_t) = 0$.

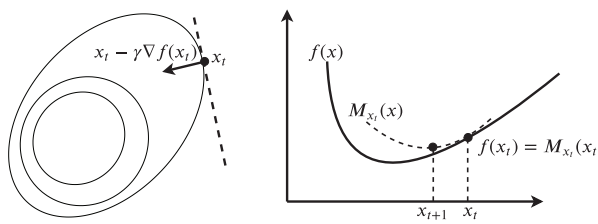

Figure 1.1: (Left) Illustration of gradient and steepest descent direction. (Right) Illustration of model function.

1.1.2 Convergence Rate

From the intuition of GD, the convergence of GD is automatically implied. This subsection provides the convergence *rate* via rigorous analysis. To show the convergence rate, let us first make some commonly used assumptions in the following.

Assumption 1. We assume:

- (**Smoothness**) All functions $F_m(\cdot)$'s are differentiable.
- (L**-Lipschitz gradient**) The objective function is assumed to have a Lipschitz gradient, that is, there exists a constant L satisfying $\forall \mathbf{x}$, $\forall \mathbf{y}$

$$\|f'(\mathbf{x}) - f'(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \tag{1.3}$$

$$f(\mathbf{y}) - f(\mathbf{x}) \leq \langle f'(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2. \tag{1.4}$$

The smoothness assumption on $F_m(\cdot)$'s implies that the overall objective function $f(\cdot)$ is differentiable or smooth too. The assumption (1.4) can be deduced from (1.3), and we refer readers to the textbook by Boyd and Vandenberghe (2004) or their course link.[1] The Lipschitz gradient assumption essentially assumes that the curvature of the objective function is bounded by L. We make the assumption of (1.4) just for convenience of use later.

We apply the Lipschitz gradient assumption and immediately obtain the following golden inequality:

$$\begin{aligned} f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) &\leq \langle f'(\mathbf{x}_t), \mathbf{x}_{t+1} - \mathbf{x}_t\rangle + \frac{L}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \\ &= -\gamma\|f'(\mathbf{x}_t)\|^2 + \frac{\gamma^2 L}{2}\|f'(\mathbf{x}_t)\|^2 \\ &= -\gamma\left(1 - \frac{\gamma L}{2}\right)\|f'(\mathbf{x}_t)\|^2. \end{aligned} \tag{1.5}$$

We can see that as long as the learning rate γ is small enough such that $1 - \gamma L/2 > 0$, $f(\mathbf{x}_{t+1})$ can improve $f(\mathbf{x}_t)$. Therefore, the learning rate

[1] http://www.seas.ucla.edu/~vandenbe/236C/lectures/gradient.pdf.

cannot be too large to guarantee the progress in each step. However, it is also a bad idea if the learning rate is too small, since the progress is proportional to $\gamma(1-\gamma L/2)$. The optimal learning rate can be obtained by simply maximizing

$$\gamma(1-\gamma L/2)$$

over γ, which gives the optimal learning rate for the gradient descent method as $\gamma^\star = 1/L$. Substituting $\gamma = \gamma^\star$ into (1.5) yields

$$f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) \leq -\frac{1}{2L}\|f'(\mathbf{x}_t)\|^2$$

or equivalently

$$f(\mathbf{x}_t) - f(\mathbf{x}_{t+1}) \geq \frac{1}{2L}\|f'(\mathbf{x}_t)\|^2. \tag{1.6}$$

Summarizing Eq. (1.6) over t from $t=1$ to $t=T$ yields

$$\begin{aligned}\frac{1}{2L}\sum_{t=1}^{T}\|f'(\mathbf{x}_t)\|^2 &\leq \sum_{t=1}^{T}(f(\mathbf{x}_t) - f(\mathbf{x}_{t+1})) \\ &= f(\mathbf{x}_1) - f(\mathbf{x}_{t+1}) \\ &\leq f(\mathbf{x}_1) - f^\star.\end{aligned}$$

Rearranging the inequality yields the following convergence rate for gradient descent.

Theorem 1.1. Under Assumption 1, the gradient descent method admits the following convergence rate

$$\frac{1}{T}\sum_{t=1}^{T}\|f'(\mathbf{x}_t)\|^2 \lesssim \frac{L}{T} \tag{1.7}$$

by choosing the learning rate $\gamma = \frac{1}{L}$. Here, we treat $f(\mathbf{x}_1) - f^\star$ as a constant.

This result indicates that the averaged gradient norm converges in the rate of $1/T$. It is worth noting that, unlike the convex case, we are unable to use the commonly used criterion $f(\mathbf{x}_t) - f^\star$ to evaluate the convergence (efficiency). That is to say, the algorithm guarantees

the convergence only to a stationary point ($\|f'(\mathbf{x}_t)\|^2 \to 0$) because of the nonconvexity. The connection between two criteria $f(\mathbf{x}_t) - f^\star$ and $\|f'(\mathbf{x}_t)\|^2$ can be seen from

$$\frac{1}{L}\|f'(\mathbf{x}_t)\|^2 \leq f(\mathbf{x}_t) - f^\star.$$

The proof can be found in the standard textbook or the course link.[2]

There are two major disadvantages for the GD method:

- The computational complexity and system overhead can be too high in each iteration to compute a single gradient.
- For nonconvex objectives, the gradient descent often sticks on a bad (shallow) local optimum.

1.1.3 Iteration/Query/Computation Complexity

The convergence rate is the key to analyzing the overall complexity. People usually consider three types of overall complexity: (1) iteration complexity, (2) query complexity, and (3) computation complexity. To evaluate the overall complexity to solve the optimization problem in (1.1), we need first to specify a precision of our solution, since in practice it is difficult (also not really necessary) to exactly solve the optimization problem. In particular, in our case the overall complexity must take into account how many iterations/queries/computations are required to ensure the average gradient norm $\frac{1}{T}\sum_{t=1}^{T}\|f'(\mathbf{x}_t)\|^2 \leq \epsilon$.

Iteration Complexity. From Theorem 1.1, it is straightforward to verify that the iteration complexity is

$$O\left(\frac{L}{\epsilon}\right). \tag{1.8}$$

Query Complexity. Here "query" refers to the number of queries of the data samples. GD needs to query all M samples in each iteration. Therefore, the query complexity can be computed from the iteration

[2] https://www.cs.rochester.edu/~jliu/CSC-576/class-note-6.pdf.

complexity by multiplying the number of queries in each iteration

$$O\left(\frac{LM}{\epsilon}\right). \tag{1.9}$$

Computation Complexity. Similarly, the computation complexity can be computed from the query complexity by multiplying the complexity of computing one sample gradient $F'_m(\mathbf{x})$. The typical complexity of computing one sample gradient is proportional to the dimension of the variable, which is d in our notation. To see the reason, let us imagine a naive linear regression with $F_m := \frac{1}{2}(\mathbf{a}_m^\top \mathbf{x} - b)^2$ and a sample gradient of $f'_m(\mathbf{x}) := \mathbf{a}_m(\mathbf{a}_m^\top \mathbf{x} - b)$. Therefore, the computation complexity of `GD` is

$$O\left(\frac{LMd}{\epsilon}\right).$$

It is worth pointing out that the computation complexity is usually proportional to the query complexity (no matter for what kinds of objective) if we consider and compare only first-order (or sample-gradient-based) methods. Therefore, in the remainder of this monograph, we compare only the query complexity and the iteration complexity.

1.2 Stochastic Gradient Descent

One disadvantage of `GD` is that it requires one to query all samples in an iteration, which could be overly expensive. To overcome this shortcoming, the stochastic gradient method `SGD` is widely used in machine learning training. Instead of computing a full gradient in each iteration, it is usual to compute only the gradient on a batch (or minibatch) of sampled data. In particular, people randomly sample an $m_t \in [M]$ independently each time and update the model by

$$(\texttt{SGD}) \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \gamma F'_{m_t}(\mathbf{x}_t), \tag{1.10}$$

where $m_t \in [M]$ denotes the index randomly selected at the tth iteration. $F'_m(\mathbf{x})$ (or $F'_{m_t}(\mathbf{x}_t)$) is called the stochastic gradient (at the tth iteration). We use $\mathbf{g}(\cdot) := F'_m(\cdot)$ (or $\mathbf{g}_t(\cdot) := F'_{m_t}(\cdot)$) to denote the stochastic gradient (or at the tth iteration) for short. An important property for the stochastic gradient is that its expectation is equal to the true

gradient, that is,

$$\mathbb{E}[\mathbf{g}(\mathbf{x})] = \mathbb{E}_m[F'_m(\mathbf{x})] = f'(\mathbf{x}) \quad \forall \mathbf{x}.$$

An immediate advantage of SGD is that the computational complexity reduces to $O(d)$ per iteration. It is worth pointing out that the SGD algorithm is NOT a descent algorithm[3] due to the randomness.

1.2.1 Convergence Rate

The next questions are whether it converges and, if it does, how quickly. We first make a typical assumption.

Assumption 2. We make the following assumption:

- (**Unbiased gradient**) The stochastic gradient is unbiased, that is,
$$\mathbb{E}_m[F'_m(\mathbf{x})] = f'(\mathbf{x}) \quad \forall \mathbf{x}.$$
- (**Bounded stochastic variance**) The stochastic gradient is with bounded variance, that is, there exists a constant σ satisfying
$$\mathbb{E}_m[\|F'_m(\mathbf{x}) - f'(\mathbf{x})\|^2] \leq \sigma^2 \quad \forall \mathbf{x}.$$

We first apply the Lipschitzian gradient property in Assumption 1:

$$\begin{aligned} f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) &\leq \langle f'(\mathbf{x}_t), \mathbf{x}_{t+1} - \mathbf{x}_t \rangle + \frac{L}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \\ &= -\gamma \langle f'(\mathbf{x}_t), \mathbf{g}_t(\mathbf{x}_t) \rangle + \frac{L\gamma^2}{2}\|\mathbf{g}_t(\mathbf{x}_t)\|^2. \end{aligned} \tag{1.11}$$

Note two important properties:

- $\mathbb{E}[\langle f'(\mathbf{x}_t), \mathbf{g}_t(\mathbf{x}_t) \rangle] = \langle f'(\mathbf{x}_t), \mathbb{E}[\mathbf{g}_t(\mathbf{x}_t)] \rangle = \|f'(\mathbf{x}_t)\|^2$
- $\mathbb{E}[\|\mathbf{g}_t(\mathbf{x}_t)\|^2] = \|f'(\mathbf{x}_t)\|^2 + \mathbb{E}[\|\mathbf{g}_t(\mathbf{x}_t) - f'(\mathbf{x}_t)\|^2] \leq \|f'(\mathbf{x}_t)\|^2 + \sigma^2$,

[3] A descent algorithm means $f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)$, that is, $\mathbf{x}_{t+1}$ is not always worse than $\mathbf{x}_t$ for any iterate t.

where the second property uses the property of variance, that is, any random variable vector ξ satisfies

$$\mathbb{E}[\|\xi\|^2] = \|\mathbb{E}[\xi]\|^2 + \mathbb{E}[\|\xi - \mathbb{E}[\xi]\|]^2. \tag{1.12}$$

Apply these two properties to (1.11) and take expectation on both sides:

$$\begin{aligned}
&\mathbb{E}[f(\mathbf{x}_{t+1})] - \mathbb{E}[f(\mathbf{x}_t)] \\
&\leq -\gamma \mathbb{E}[\|f'(\mathbf{x}_t)\|^2] + \frac{L\gamma^2}{2}(\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] + \sigma^2) \qquad (1.13)\\
&\leq -\gamma\left(1 - \frac{\gamma L}{2}\right)\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] + \frac{\gamma^2}{2}L\sigma^2. \qquad (1.14)
\end{aligned}$$

From (1.13), we can see that `SGD` does not guarantee "descent" in each iteration, unlike `GD`, but it does guarantee "descent" in the expectation sense in each iteration as long as γ is small enough and $\|f'(\mathbf{x}_t)\|^2 > 0$. This is because the first term in (1.13) is in the order of $O(\gamma)$ while the second term is in the order of $O(\gamma^2)$.

Next we summarize (1.13) from $t = 1$ to $t = T$ and obtain

$$\mathbb{E}[f(\mathbf{x}_{T+1})] - f(\mathbf{x}_1) \leq -\gamma\left(1 - \frac{\gamma L}{2}\right)\sum_{t=1}^{T}\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] + \frac{\gamma^2}{2}TL\sigma^2. \tag{1.15}$$

We choose the learning rate $\gamma = \frac{1}{L+\sigma\sqrt{TL}}$ which implies that $(1-\gamma L/2) > 1/2$. It follows

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] \\
&\lesssim \frac{f(\mathbf{x}_1) - \mathbb{E}[f(\mathbf{x}_{T+1})]}{T\gamma} + \gamma L\sigma^2 \\
&\lesssim \frac{f(\mathbf{x}_1) - f^\star}{T\gamma} + \gamma L\sigma^2 \\
&\lesssim \frac{(f(\mathbf{x}_1) - f^\star)L}{T} + \frac{(f(\mathbf{x}_1) - f^\star)\sqrt{L}\sigma}{\sqrt{T}}.
\end{aligned}$$

Therefore the convergence rate of SGD can be summarized into the following theorem.

Theorem 1.2. Under Assumptions 1 and 2, the SGD method admits the following convergence rate

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] \lesssim \frac{L}{T} + \frac{\sqrt{L}\sigma}{\sqrt{T}}$$

by choosing the learning rate $\gamma = \frac{1}{L+\sigma\sqrt{TL}}$. Here we treat $f(\mathbf{x}_1) - f^\star$ as a constant.

We highlight the following observations from Theorem 1.2.

- (**Consistent with GD**) If $\sigma = 0$, the SGD algorithm reduces to GD and the convergence rate becomes $L(f(\mathbf{x}_1) - f^\star)/L$, which is consistent with the convergence rate for GD proven in Theorem 1.1.
- (**Asymptotic convergence rate**) The convergence rate of SGD achieves $O(1/\sqrt{T})$.

1.2.2 Iteration/Query Complexity

Using a similar analysis as Subsection 1.1.3, we can obtain the iteration complexity of SGD, which is also the query complexity (since there is only one query per one sample gradient)

$$O\left(\frac{L}{\epsilon} + \frac{L\sigma^2}{\epsilon^2}\right).$$

It is worse than GD in terms of the iteration complexity in (1.8), which is not a surprising result. The comparison of query complexity makes more sense since it is more related to the physical running time or the computation complexity. From the detailed comparison in Table 1.2, we can see that

- SGD is superior to GD, if $\frac{\sigma^2}{M} \ll \epsilon$;
- SGD is inferior to GD, if $\frac{\sigma^2}{M} \gg \epsilon$.

Table 1.2: Complexity comparison among `GD`, `SGD`, and `mb-SGD`

Algorithms	Iteration Complexity	Query Complexity
`GD`	$O\left(\frac{L}{\epsilon}\right)$	$O\left(\frac{ML}{\epsilon}\right)$
`SGD`	$O\left(\frac{L}{\epsilon}+\frac{L\sigma^2}{\epsilon^2}\right)$	$O\left(\frac{L}{\epsilon}+\frac{L\sigma^2}{\epsilon^2}\right)$
`mb-SGD`	$O\left(\frac{L}{\epsilon}+\frac{L\sigma^2}{B\epsilon^2}\right)$	$O\left(\frac{LB}{\epsilon}+\frac{L\sigma^2}{\epsilon^2}\right)$

It is worth pointing out that when the number of samples M is huge and a low precision solution is satisfactory,[4] $\frac{\sigma}{M\sqrt{L}} \ll \epsilon$ usually holds. As a result, `SGD` is favored for solving big data problems.

1.2.3 Minibatch Stochastic Gradient Descent (`mb-SGD`)

A straightforward variant of the `GD` algorithm is to compute the gradient of a minibatch of samples (instead of a single sample) in each iteration, that is,

$$\mathbf{g}^{\mathcal{B}}(\mathbf{x}) = \frac{1}{B}\sum_{m\in\mathcal{B}} F'_m(\mathbf{x}), \tag{1.16}$$

where $B := |\mathcal{B}|$. The minibatch $\mathcal{B}$ is obtained by using i.i.d. samples with (or without) replacement. One can easily verify that

$$\mathbb{E}[\mathbf{g}^{\mathcal{B}}(\mathbf{x})] = f'(\mathbf{x}).$$

Sample "With" Replacement. The stochastic variance (for the "with" replacement case) can be bounded by

$$\begin{aligned}&\mathbb{E}[\|\mathbf{g}^{\mathcal{B}}(\mathbf{x}) - f'(\mathbf{x})\|^2] \\ &\quad= \mathbb{E}\left[\left\|\frac{1}{B}\sum_{m\in\mathcal{B}}\left(F'_m(\mathbf{x}) - f'(\mathbf{x})\right)\right\|^2\right]\end{aligned} \tag{1.17}$$

[4] A low precision solution is satisfactory in many application scenarios, since a high precision solution may cause an unwanted overfitting issue.

$$= \frac{1}{B} \sum_{m \in \mathcal{B}} \mathbb{E}\left[\left\| F'_m(\mathbf{x}) - f'(\mathbf{x}) \right\|^2 \right]$$
$$\leq \frac{\sigma^2}{B} \quad \text{(from Assumption 2)}.$$

Sample "Without" Replacement. The stochastic variance for the "without" replacement is even smaller, but it involves a bit more complicated derivation. We essentially need the following key lemma.

Lemma 1.3. Give a set including $M \geq 2$ real numbers $\{a_1, a_2, \ldots, a_M\}$. Define a random variable

$$\bar{\xi}_{[B]} := \frac{1}{B} \sum_{m=1}^{B} \xi_m,$$

where $\xi_1, \ldots, \xi_B$ are uniformly randomly sampled from the set "without" replacement, and $B (1 \leq B \leq M)$ is the batch size. Then the following equality holds

$$\mathbf{Var}[\bar{\xi}] = \left(\frac{M - B}{M - 1} \right) \frac{\mathbf{Var}[\xi_1]}{B}.$$

Proof. First, it is not hard to see that the marginal distributions of ξ_m's are identical. For simplicity of notation, we assume that $\mathbb{E}[\xi_m] = 0$ without the loss of generality. Therefore, we have $\mathbf{Var}[\xi_m] = \mathbb{E}[\xi_m^2]$ for all k.

Next we have the following derivation:

$$\begin{aligned} \mathbf{Var}[B\bar{\xi}_{[B]}] &= \mathbb{E}[(B\bar{\xi}_{[B]})^2] \quad \text{(due to } \mathbb{E}[\bar{\xi}_{[B]}] = 0) \\ &= \sum_{m=1}^{B} \mathbb{E}[\xi_m^2] + \sum_{k \neq l} \mathbb{E}[\xi_m \xi_l] \\ &= B\mathbf{Var}[\xi_1] + B(B-1)\mathbb{E}[\xi_1 \xi_2], \end{aligned} \tag{1.18}$$

where the last equality uses the fact $\mathbb{E}[\xi_m^2] = \mathbf{Var}[\xi_k] = \mathbf{Var}[\xi_1]$ for any k and $E[\xi_k \xi_l] = E[\xi_1 \xi_2]$ for any $k \neq l$. Note that $\mathbf{Var}[M\bar{\xi}_{[M]}] = 0$, since

it has only one possible combination for $\{\xi_1, \xi_2, \ldots, \xi_M\}$. Then letting $B = M$ obtains the following dependence from (1.18)

$$\mathbb{E}[\xi_1 \xi_2] = \frac{-1}{M-1} \mathbf{Var}[\xi_1].$$

Plug this result into (1.18)

$$\mathbf{Var}[B\bar{\xi}_{[B]}] = B\left(\frac{M-B}{M-1}\right)\mathbf{Var}[\xi_1],$$

which implies the claimed result. □

If a_m's are vectors and satisfy $\frac{1}{M}\sum_{m=1}^{M} \xi_m = 0$, from Lemma 1.3 one can easily verify

$$\mathbb{E}[\|\bar{\xi}\|^2] = B\left(\frac{M-B}{M-1}\right)\mathbb{E}[\|\xi_1\|^2]. \tag{1.19}$$

Now we are ready to compute the stochastic variance for the "without" replacement sampling strategy. Let $\mathcal{B}$ be a batch of samples "without" replacement. Then we let $\xi_m := F'_m(\mathbf{x}) - f'(\mathbf{x})$ and from (1.19) obtain

$$\begin{aligned}
\mathbb{E}[\|\mathbf{g}^{\mathcal{B}}(\mathbf{x}) - f'(\mathbf{x})]\|^2] &= \mathbb{E}[\|\bar{\xi}\|^2] \\
&= \left(\frac{M-B}{M-1}\right)\frac{\mathbb{E}[\|\xi_1\|^2]}{B} \\
&\leq \left(\frac{M-B}{M-1}\right)\frac{\sigma^2}{B} \\
&\leq \frac{\sigma^2}{B}.
\end{aligned}$$

To sum up, we have the stochastic variance bounded by $\frac{\sigma^2}{B}$ no matter "with" or "without" replacement sampling.

We can observe that the effect of using a minibatch stochastic gradient is nothing but reduced variance. All remaining analysis for the convergence rate remains the same. Therefore, it is quite easy to obtain the convergence rate of `mb-SGD`

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|f'(\mathbf{x}_t)\|^2] \lesssim \frac{L}{T} + \frac{\sqrt{L}\sigma}{\sqrt{TB}}. \tag{1.20}$$

The iteration complexity and the query complexity are reported in Table 1.2.

1.3 A Simplified Distributed Communication Model

When scaling up the stochastic gradient descent (SGD) algorithm to a distributed setting, one often needs to develop *system relaxations* techniques to achieve better performance and scalability. In this monograph, we describe multiple popular system relaxation techniques that have been developed in recent years. In this subsection, we introduce a simple performance model of a distributed system, which will be used in later sections to reason about the performance impact of different relaxation techniques.

From a mathematical optimization perspective, all of the system relaxations that we will describe *do not make the convergence (loss vs. # iterations/epochs) faster*.[5] Then *why do we even want to introduce these relaxations into our system in the first place?*

One common theme of the techniques we cover in this monograph is that their goal is not to improve the convergence rate in terms of # iterations/epochs; rather, their goal is to make each iteration finish faster in terms of wall-clock time. As a result, to reason about each system relaxation technique in this monograph, we need to first agree on a *performance model* of the underlying distributed system. In this subsection, we introduce a very simple performance model — it ignores many (if not most) important system characteristics, but it is just informative enough for readers to understand why each system relaxation technique in this monograph actually makes a system faster.

1.3.1 Assumptions

In practice, it is often the case that the bandwidth or latency of each worker's network connection is the dominating bottleneck in the communication cost. As a result, in this monograph we focus on the following simplified communication model.

Figure 1.2 illustrates our communication model. Each worker (blue rectangle) corresponds to *one* computation device (worker), and all

[5] The reason that we emphasize the "mathematical optimization" perspective is that some researchers find that certain system relaxations can actually lead to better generalization performance. We do not consider generalization in this monograph.

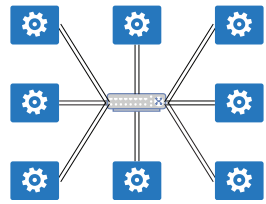

Figure 1.2: An illustration of the distributed communication model we use in this monograph. We assume that all devices (worker, machine) are connected via a "logical switch" whose property is defined in Subsection 1.3.

workers are connected via a "logical switch" that has the following property.

1. The switch has infinitely large bandwidth. We make this simplifying assumption to reflect the observation that, in practice, the bottleneck is often the bandwidth or latency of each worker's network connection.

2. For each message that "passes through" the switch (sent by worker w_i and received by worker w_j), the switch adds a constant delay t_{latency} independently of the number of concurrent messages that this switch is serving. This delay is the timestamp difference between the sender sending out the first bit and the receiver receiving the first bit.

For each worker, we also assume the following properties.

1. Each worker can only send one message at the same time.

2. Each worker can only receive one message at the same time.

3. Each worker can concurrently receive one message and send one message at the same time.

4. Each worker has a fixed bandwidth, i.e., to send/receive one unit (e.g., MB) amount of data, it requires $t_{\text{transfer1MB}}$ seconds.

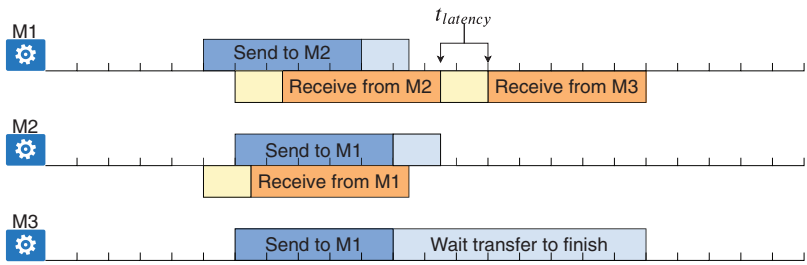

Figure 1.3: Illustration of the communication pattern of Example 1.2.

Example 1.1. Under the above communication model, consider the following three events:

```
time      event
0:05      M1 send 1MB to M2
0:06      M2 send 1MB to M1
0:06      M3 send 1MB to M2
```

We assume that the latency added by the switch t_{latency} is 1.5 units of time and it took 5 units of time to transfer 1 MB of data. Figure 1.3 illustrates the timeline on three machines under our communication model. The yellow block corresponds to the *latency* added by the "logical switch". We also see that the machine `M1` can concurrently send (blue block) and receive (orange block) data at the same time; however, when the machine `M3` tries to send data to `M1`, because the machine `M2` is already sending data to `M1`, `M3` needs to wait (the shallow blue block of `M3`).

Example 1.2. Figure 1.4 illustrates a hypothetical scenario in which all data sent in Example 1.2 are "magically" compressed by 2× at the sender. As we will see in later sections, this is similar to what would happen if one were to compress the gradient by 2× during training.

We make multiple observations from Figure 1.4.

1. First, compressing data does make the "system" faster. Without compression, all three events finish in 14 units of time (Figure 1.3) whereas it finishes in 9 units of time after compression. This is

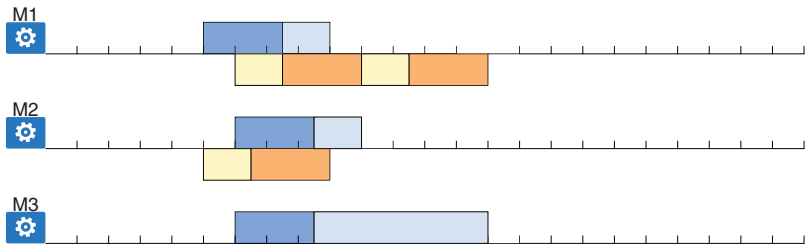

Figure 1.4: Illustration of the communication pattern of Example 1.2, with 2× data compression.

because the time used to *transfer* the data is decreased by half in our communication model.

2. Second, even if the data are compressed by 2×, the speedup of the system is smaller than that; in fact, it is only $14/9 = 1.55\times$. This is because, even though the transfer time is cut by half, the communication latency does not decrease as a result of the data compression.

We now use the above communication model to describe the *communication patterns* of three popular ways to implement distributed stochastic gradient descent. These implementations will often serve as the baseline from which we apply different system relaxations to remove certain system bottlenecks that arise in different configurations of $(t_{\text{latency}}, t_{\text{transfer}})$ together with the relative computational cost on each machine.

Workloads. We focus on one of the core building blocks to implement a distributed SGD system — each worker M_i holds a parameter vector w_i, and they communicate to compute the sum of all parameter vectors: $S = \sum_i w_i$. At the end of communication, each worker holds one copy of S.

1.3.2 Synchronous Parameter Server

The parameter server is not only one of the most popular system architectures for distributed stochastic gradient descent; it is also one

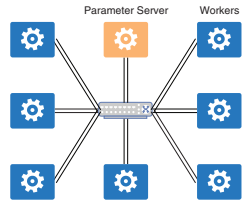

Figure 1.5: Illustration of the parameter server architecture with a single dedicated parameter server.

of the most popular communication models that researchers have in mind when they conduct theoretical analysis. In a parameter server architecture, one or more machines serve as the *parameter server(s)* and other machines serve as the workers processing data. Periodically, workers send updates to the parameters to the parameter servers and the parameter servers send back the updated parameters. Figure 1.5 illustrates this architecture: the orange machine is the parameter server and the blue machines are the workers.

A real-world implementation of a parameter server architecture usually involves many system optimizations to speed up the communication. In this subsection, we build our abstraction using the simplest implementation with only a single machine serving as the parameter server. We also scope ourselves and only focus on the synchronous communication case.

When using this simplified parameter server architecture to calculate the sum S, each worker M_i sends their local parameter vector w_i to the parameter server, and the parameter server collects all these local copies, sums them up, and sends back to each worker. In a simple example with three workers and one parameter server, the series of communication events looks like this:

```
Time=0      Worker1 send w1 to PS
Time=0      Worker2 send w2 to PS
Time=0      Worker3 send w3 to PS
Time=T      PS send S to Worker1
Time=T      PS send S to Worker2
Time=T      PS send S to Worker3
```

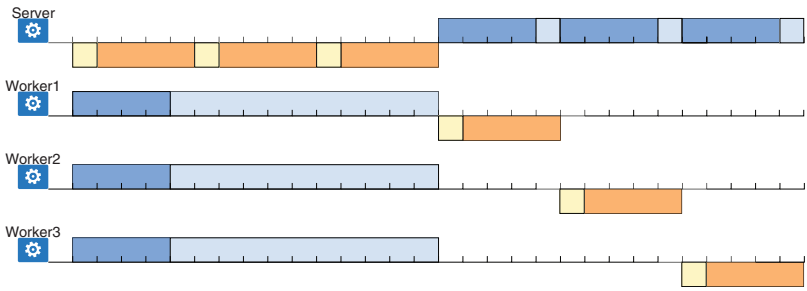

Figure 1.6: Illustration of the communication pattern of the parameter server architecture with a single dedicated parameter server.

Figure 1.6 illustrates the communication timeline of these events. We see that, in the first phase, all workers send their local parameter vectors to the parameter server at the same time. Because, in our communication model, the parameter server can receive data from only one worker at a time, it took $3(t_{\text{latency}} + t_{\text{transfer}})$ for the aggregation phase to finish. In the broadcast phase, because, in our communication model, the parameter server can send data to only one worker at a time, it took $3(t_{\text{latency}} + t_{\text{transfer}})$ for the broadcast phase to finish.

In general, when there are N workers and 1 parameter server, *as under our communication model*, a parameter server architecture in which all workers are *perfectly synchronized* takes

$$2n(t_{\text{latency}} + t_{\text{transfer}})$$

to compute and broadcast the sum S over all local copies $\{w_i\}$.

Discussions. As we see, the communication cost of the parameter server architecture grows linearly with respect to the total number of workers we have in the system. As a result, this architecture could be sensitive to both latency and transfer time. This motivates some system relaxations that could alleviate potential system bottlenecks.

1. When the network has small latency t_{latency} compared with the transfer time t_{transfer}, one could conduct lossy compression (e.g., via quantization, sparsification, or both) to decrease the transfer time. Usually, this approach can lead to a linear speedup with

respect to the compression rate, up to a point that t_{latency} starts to dominate.

2. When the network has large latency t_{latency}, compression on its own won't be the solution. In this case, one could adopt a decentralized communication pattern, as we will discuss later in this monograph.

1.3.3 AllReduce

Calculating the sum over distributed workers is a very common operator used in distributed computing and high performance computing systems. In many communication frameworks, it can be achieved using the `AllReduce` operator. Optimizing and implementing the `AllReduce` operator has been studied by the HPC community for decades, and the implementation is usually different for different numbers of machines, different sizes of messages, and different physical communication topologies.

In this monograph, we focus on the simplest case, in which all workers form a *logical* ring and communicate only to their neighbors (all communications still go through the single switch all workers are connected to). We also assume that the local parameter vector is large enough.

Under these assumptions, we can implement an `AllReduce` operator in the following way. Each worker w_n partitions their local parameter vectors into N partitions (N is the number of workers): w_n^k is the kth partition of the local model w_n. The communication happens in two phases.

1. **Phase 1.** At the first iteration of Phase 1, each machine n sends w_n^n to its "next" worker in the logical ring, i.e., w_j where $j = n + 1 \bmod N$. Once machine j receives a partition k, it sums up the received partition with its local partition, and sends the aggregated partition to the next worker in the next iteration. After $N - 1$ communication iterations, different workers now have the sum of different partitions.

2. **Phase 2.** Phase 2 is similar to Phase 1, with the difference that when machine n receives a partition k, it replaces its local copy

with the received partition, and passes it onto the next machine in the next iteration.

At the end of communication, all workers have the sum S of all partitions.

Example 1.3. We walk through an example with four workers $M_1, \ldots, M_4$. The communication pattern of the above implementation is as follows. We use `w{Mi,j}` to denote the jth partition on machine `Mi`.

```
# For the first partition w{M1 (worker id), 1 (partition id)}
Time= 0   M1 sends w{M1,1}                                  to M2
Time= t   M2 sends w{M1,1} + w{M2,1}                        to M3
Time=2t   M3 sends w{M1,1} + w{M2,1} + w{M3,1}              to M4
Time=3t   M4 sends w{M1,1} + w{M2,1} + w{M3,1} + w{M4,1}    to M1
Time=4t   M1 sends w{M1,1} + w{M2,1} + w{M3,1} + w{M4,1}    to M2
Time=5t   M2 sends w{M1,1} + w{M2,1} + w{M3,1} + w{M4,1}    to M3

# For the second partition w{M2 (worker id), 2 (partition id)}
Time= 0   M2 sends w{M2,2}                                  to M3
Time= t   M3 sends w{M2,2} + w{M3,2}                        to M4
Time=2t   M4 sends w{M2,2} + w{M3,2} + w{M4,2}              to M1
Time=3t   M1 sends w{M2,2} + w{M3,2} + w{M4,2} + w{M1,2}    to M2
Time=4t   M2 sends w{M2,2} + w{M3,2} + w{M4,2} + w{M1,2}    to M3
Time=5t   M3 sends w{M2,2} + w{M3,2} + w{M4,2} + w{M1,2}    to M4

# For the third partition w{M3 (worker id), 3 (partition id)}
Time= 0   M3 sends w{M3,3}                                  to M4
Time= t   M4 sends w{M3,3} + w{M4,3}                        to M1
Time=2t   M1 sends w{M3,3} + w{M4,3} + w{M1,3}              to M2
Time=3t   M2 sends w{M3,3} + w{M4,3} + w{M1,3} + w{M2,3}    to M3
Time=4t   M3 sends w{M3,3} + w{M4,3} + w{M1,3} + w{M2,3}    to M4
Time=5t   M4 sends w{M3,3} + w{M4,3} + w{M1,3} + w{M2,3}    to M1

# For the fourth partition w{M4 (worker id), 4 (partition id)}
Time= 0   M4 sends w{M4,4}                                  to M1
Time= t   M1 sends w{M4,4} + w{M1,4}                        to M2
Time=2t   M2 sends w{M4,4} + w{M1,4} + w{M2,4}              to M3
Time=3t   M3 sends w{M4,4} + w{M1,4} + w{M2,4} + w{M3,4}    to M4
Time=4t   M4 sends w{M4,4} + w{M1,4} + w{M2,4} + w{M3,4}    to M1
Time=5t   M1 sends w{M4,4} + w{M1,4} + w{M2,4} + w{M3,4}    to M2
```

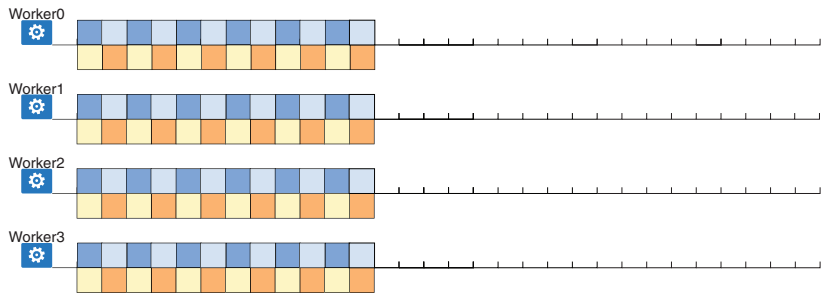

Figure 1.7: Illustration of the communication pattern of the AllReduce architecture with ring topology.

From the above pattern, it is not hard to see why, at the end, each worker has a copy of $S = \sum_{n=1}^{N} w_n$.

One interesting property of the above way of implementing the `AllReduce` operator is that, at any timestep, each machine concurrently sends and receives one partition of the data, which is possible in our communication model. Figure 1.7 illustrates the communication timeline.

We make multiple observations.

1. Compared with a parameter server architecture with a single parameter server (Figure 1.6), the total amount of data that each worker sends and receives is the same in both cases — in both cases, the amount of data sent and received by each machine is equal to the size of the parameter vector.

2. At any given time, each worker sends and receives data concurrently. At any given time, only the left neighbor w_n sends data to $w_{n+1 \bmod N}$ and $w_{n+1 \bmod N}$ only sends data to $w_{n+2 \bmod N}$. This allows the system to take advantage of the *aggregated* bandwidth of N machines (which grows linearly with respect to N) instead of being bounded by the bandwidth of a single central parameter server.

In general, when there are $N + 1$ workers, *as under our communication model* and assuming that the computation cost to sum up

parameter vectors is negligible, an `AllReduce` operator in which all workers are *perfectly synchronized* took

$$2Nt_{\text{latency}} + 2t_{\text{transfer}}$$

to compute and broadcast the sum S over all local copies $\{w_n\}$.

Discussions. As we see, the latency of an `AllReduce` operator grows linearly with respect to the total number of workers we have in the system. As a result, this architecture could be sensitive to network latency. This motivates some system relaxations that could alleviate potential system bottlenecks.

1. When the network has large latency t_{latency}, compression on its own won't be the solution. In this case, one could adopt a decentralized communication pattern, as we will discuss later in this monograph.

2. When the network has small latency t_{latency} and the parameter vector is very large, the transfer time t_{transfer} can still become the bottleneck. In this case, one could conduct lossy compression (e.g., via quantization, sparsification, or both) to decrease the transfer time. Usually, this approach can lead to a linear speedup with respect to the compression rate, up to a point that t_{latency} starts to dominate.

Caveats. We will discuss the case of asynchronous communication later in this monograph. Although it is quite natural to come up with an asynchronous parameter server architecture, making the `AllReduce` operator run in an asynchronous fashion is less natural. As a result, when there are stragglers in the system (e.g., one worker is significantly slower than all other workers), `AllReduce` can make it more difficult to implement a straggler avoidance strategy if one simply uses the off-the-shelf implementation.

Why Do We Partition the Parameter Vector? One interesting design choice in implementing the `AllReduce` operator is setting each local

parameter vector to be partitioned into N partitions. This decision is important if you want to fully take advantage of the aggregated bandwidth of *all* workers. Take the same four-worker example and assume that we do not partition the model. In this case, the series of communication events will look like this:

```
Time= 0   M1 sends w{M1}                              to M2
Time= t   M2 sends w{M1} + w{M2}                      to M3
Time=2t   M3 sends w{M1} + w{M2} + w{M3}              to M4
Time=3t   M4 sends w{M1} + w{M2} + w{M3} + w{M4}      to M1
Time=4t   M1 sends w{M1} + w{M2} + w{M3} + w{M4}      to M2
Time=5t   M2 sends w{M1} + w{M2} + w{M3} + w{M4}      to M3
```

In general, with $N + 1$ workers, the communication cost without partitioning becomes

$$2N(t_{\text{latency}} + t_{\text{transfer}}).$$

Comparing this with the $2Nt_{\text{latency}} + 2t_{\text{transfer}}$ cost of `AllReduce` with model partition, we see that model partition is the key reason for taking advantage of the full aggregated bandwidth provided by all machines.

1.3.4 Multi-Machine Parameter Server

One can extend the single-server parameter server architecture and use multiple machines serving as parameter servers instead. In this monograph, we focus on the scenario in which each worker also serves as a parameter server.

Under this assumption, we can implement a multi-server parameter server architecture in the following way. Each worker w_n partitions their local parameter vectors into N partitions (N is the number of workers): w_n^k. The communication happens in two phases.

1. Phase 1. All workers send their nth partition to worker w_n. Worker w_n aggregates all messages and calculates the nth partition of the sum S.

2. Phase 2. Worker w_n sends the nth partition of the sum S to all other workers.

With careful arrangement of communication events, we can also take advantage of the full aggregated bandwidth in this architecture, as illustrated in the following example.

Example 1.4. We walk through an example with four workers $M_1, \ldots, M_4$. The communication pattern of the above implementation is as follows:

```
# First partition: w{M1 (worker id), 1 (partition id)}
# First partition of the result: S{1}
Time= 0   M2 sends w{M2,1} to M1
Time= t   M3 sends w{M3,1} to M1
Time=2t   M4 sends w{M4,1} to M1
Time=3t   M1 sends S{1}    to M2
Time=4t   M1 sends S{1}    to M3
Time=5t   M1 sends S{1}    to M4

# Second partition: w{M2 (worker id), 2 (partition id)}
# Second partition of the result: S{2}
Time= 0   M1 sends w{M1,2} to M2
Time= t   M4 sends w{M4,2} to M2
Time=2t   M3 sends w{M3,2} to M2
Time=3t   M2 sends S{2}    to M3
Time=4t   M2 sends S{2}    to M4
Time=5t   M2 sends S{2}    to M1

# Third partition w{M3 (worker id), 3 (partition id)}
# Third partition of the result: S{3}
Time= 0   M4 sends w{M4,3} to M3
Time= t   M1 sends w{M1,3} to M3
Time=2t   M2 sends w{M2,3} to M3
Time=3t   M3 sends S{3}    to M4
Time=4t   M3 sends S{3}    to M1
Time=5t   M3 sends S{3}    to M2

# Fourth partition w{M4 (worker id), 4 (partition id)}
# Fourth partition of the result: S{4}
```

```
Time= 0   M3 sends w{M3,4} to M4
Time= t   M2 sends w{M2,4} to M4
Time=2t   M1 sends w{M1,4} to M4
Time=3t   M4 sends S{4}    to M1
Time=4t   M4 sends S{4}    to M2
Time=5t   M4 sends S{4}    to M3
```

For the above communication events, it is not hard to see that, at the end, each machine has access to the sum $S = \sum_{n=1}^{N} w_n$. In terms of the communication pattern, under our communication model, the multi-server parameter server architecture has the same pattern as `AllReduce`, illustrated in Figure 1.7.

In general, when there are $N + 1$ workers, *as under our communication model* and assuming that the computation cost to sum up parameter vectors is negligible, a multi-server parameter server architecture in which all workers are *perfectly synchronized* took

$$2Nt_{\text{latency}} + 2t_{\text{transfer}}$$

to compute and broadcast the sum S over all local copies $\{w_n\}$.

References

Acharya, J., C. De Sa, D. Foster, and K. Sridharan (2019). "Distributed learning with sublinear communication". *International Conference on Machine Learning*.

Agarwal, A. and J. C. Duchi (2012). "Distributed delayed stochastic optimization". *IEEE Conference on Decision and Control*.

Agarwal, N., A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan (2018). "cpSGD: Communication-efficient and differentially-private distributed SGD". *Annual Conference on Neural Information Processing Systems*. 7575–7586.

Alistarh, D., Z. Allen-Zhu, and J. Li (2018a). "Byzantine stochastic gradient descent". *Annual Conference on Neural Information Processing Systems*. 4618–4628.

Alistarh, D., D. Grubic, J. Li, R. Tomioka, and M. Vojnovic (2017). "QSGD: Communication-efficient SGD via gradient quantization and encoding". *Annual Conference on Neural Information Processing Systems*. 1707–1718.

Alistarh, D., T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli (2018b). "The convergence of sparsified gradient methods". *Annual Conference on Neural Information Processing Systems*. 5977–5987.

Allen-Zhu, Z. and E. Hazan (2016). "Variance reduction for faster non-convex optimization". *International Conference on Machine Learning*.

Allen-Zhu, Z. and Y. Yuan (2016). "Improved SVRG for non-strongly-convex or sum-of-non-convex objectives". *International Conference on Machine Learning*.

Arjevani, Y. (2017). "Limitations on variance-reduction and acceleration schemes for finite sums optimization". *Annual Conference on Neural Information Processing Systems*.

Assran, M., N. Loizou, N. Ballas, and M. Rabbat (2019). "Stochastic gradient push for distributed deep learning". *International Conference on Machine Learning*.

Aybat, N., Z. Wang, and G. Iyengar (2015). "An asynchronous distributed proximal gradient method for composite convex optimization". *International Conference on Machine Learning*.

Banner, R., I. Hubara, E. Hoffer, and D. Soudry (2018). "Scalable methods for 8-bit training of neural networks". *Annual Conference on Neural Information Processing Systems*. 5151–5159.

Ben-Nun, T. and T. Hoefler (2018). "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis". *arXiv: 1802.09941*.

Bernstein, J., Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar (2018). "signSGD: Compressed optimisation for non-convex problems". *International Conference on Machine Learning*.

Beznosikov, A., S. Horváth, P. Richtárik, and M. Safaryan (2020). "On biased compression for distributed learning". *arXiv:2002.12410*.

Bietti, A. and J. Mairal (2017). "Stochastic optimization with variance reduction for infinite datasets with finite sum structure". *Annual Conference on Neural Information Processing Systems*. 1622–1632.

Blanchard, P., E. M. El Mhamdi, R. Guerraoui, and J. Stainer (2017). "Machine learning with adversaries: Byzantine tolerant gradient descent". *Annual Conference on Neural Information Processing Systems*. 118–128.

Bottou, L., F. E. Curtis, and J. Nocedal (2016). "Optimization methods for large-scale machine learning". *arXiv:1606.04838*.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Chaturapruek, S., J. C. Duchi, and C. Ré (2015). "Asynchronous stochastic convex optimization: The noise is in the noise and SGD don't care". *Annual Conference on Neural Information Processing Systems*. 1531–1539.

Chen, C., N. Ding, C. Li, Y. Zhang, and L. Carin (2016). "Stochastic gradient MCMC with stale gradients". *Annual Conference on Neural Information Processing Systems*. 2945–2953.

Chen, L., H. Wang, Z. Charles, and D. Papailiopoulos (2018). "DRACO: Byzantine-resilient distributed training via redundant gradients". *International Conference on Machine Learning*.

Chen, X., M. R. Lyu, and I. King (2017). "Toward efficient and accurate covariance matrix estimation on compressed data". *International Conference on Machine Learning*.

Cho, M., U. Finkler, D. Kung, and H. Hunter (2019). "BlueConnect: Decomposing AllReduce for deep learning on heterogeneous network hierarchy". *Conference on Systems and Machine Learning*.

Cohen, M., J. Diakonikolas, and L. Orecchia (2018). "On acceleration with noise-corrupted gradients". *International Conference on Machine Learning*.

Colin, I., A. Bellet, J. Salmon, and S. Clémen**c**con (2016). "Gossip dual averaging for decentralized optimization of pairwise functions". *International Conference on Machine Learning*.

Cutkosky, A. and R. Busa-Fekete (2018). "Distributed stochastic optimization via adaptive SGD". *Annual Conference on Neural Information Processing Systems*. 1914–1923.

Damaskinos, G., E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki (2018). "Asynchronous Byzantine machine learning (the case of SGD)". *International Conference on Machine Learning*.

De Sa, C. M., C. Zhang, K. Olukotun, and C. Ré (2015). "Taming the wild: A unified analysis of Hogwild-style algorithms". *Annual Conference on Neural Information Processing Systems*.

Dünner, C., T. Parnell, D. Sarigiannis, N. Ioannou, A. Anghel, G. Ravi, M. Kandasamy, and H. Pozidis (2018). "Snap ML: A hierarchical framework for machine learning". *Annual Conference on Neural Information Processing Systems.* 250–260.

Elgohary, A., M. Boehm, P. J. Haas, F. R. Reiss, and B. Reinwald (2016). "Compressed linear algebra for large-scale machine learning". *Proceedings of the VLDB Endowment.* 9(12): 960–971.

Garber, D., O. Shamir, and N. Srebro (2017). "Communication-efficient algorithms for distributed stochastic principal component analysis". *International Conference on Machine Learning.*

Ghadimi, S., G. Lan, and H. Zhang (2013). "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization". *arXiv:1308.6594.*

Gopal, S. (2016). "Adaptive sampling for SGD by exploiting side information". *International Conference on Machine Learning.*

Goyal, P., P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He (2017). "Accurate, large minibatch SGD: Training ImageNet in 1 hour". *arXiv:1706.02677.*

Gu, B., Z. Huo, C. Deng, and H. Huang (2018). "Faster derivative-free stochastic algorithm for shared memory machines". *International Conference on Machine Learning.*

Gupta, S., A. Agrawal, K. Gopalakrishnan, and P. Narayanan (2015). "Deep learning with limited numerical precision". *International Conference on Machine Learning*: 1737–1746.

Hanzely, F., K. Mishchenko, and P. Richtarik (2018). "SEGA: Variance reduction via gradient sketching". *Annual Conference on Neural Information Processing Systems.* 2086–2097.

Haochen, J. and S. Sra (2019). "Random shuffling beats SGD after finite epochs". *International Conference on Machine Learning.* 2624–2633.

Hashemi, S. H., S. A. Jyothi, and R. H. Campbell (2019). "TicTac: Accelerating distributed deep learning with communication scheduling". *Conference on Systems and Machine Learning.*

Hazan, E. and H. Luo (2016). "Variance-reduced and projection-free stochastic optimization". *International Conference on Machine Learning.*

He, C., C. Tan, H. Tang, S. Qiu, and J. Liu (2019). "Central server free federated learning over single-sided trust social networks". *arXiv:1910.04956*.

He, L., A. Bian, and M. Jaggi (2018). "COLA: Decentralized linear learning". *Annual Conference on Neural Information Processing Systems*. 4541–4551.

Hofmann, T., A. Lucchi, S. Lacoste-Julien, and B. McWilliams (2015). "Variance reduced stochastic gradient descent with neighbors". *Annual Conference on Neural Information Processing Systems*. 2305–2313.

Horváth, S. and P. Richtarik (2019). "Nonconvex variance reduced optimization with arbitrary sampling". *International Conference on Machine Learning*.

Hsieh, C.-J., H.-F. Yu, and I. Dhillon (2015). "PASSCoDe: Parallel ASynchronous stochastic dual co-ordinate descent". *International Conference on Machine Learning*.

Hsieh, K., A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu (2017). "Gaia: Geo-distributed machine learning approaching LAN speeds". *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*.

Jayarajan, A., J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko (2019). "Priority-based parameter propagation for distributed DNN training". *Conference on Systems and Machine Learning*.

Ji, K., Z. Wang, Y. Zhou, and Y. Liang (2019). "Improved zeroth-order variance reduced algorithms and analysis for nonconvex optimization". *International Conference on Machine Learning*.

Jia, Z., M. Zaharia, and A. Aiken (2019). "Beyond data and model parallelism for deep neural networks". *Conference on Systems and Machine Learning*.

Jiang, J., B. Cui, C. Zhang, and L. Yu (2017). "Heterogeneity-aware distributed parameter servers". *International Conference on Management of Data*.

Jiang, J., F. Fu, T. Yang, and B. Cui (2018). "SketchML: Accelerating distributed machine learning with data sketches". *Proceedings of the 2018 International Conference on Management of Data. SIGMOD'18.* 1269–1284.

Jiang, P. and G. Agrawal (2018). "A linear speedup analysis of distributed deep learning with sparse and quantized communication". *Annual Conference on Neural Information Processing Systems.* 2530–2541.

Johnson, T. B. and C. Guestrin (2018). "Training deep models faster with robust, approximate importance sampling". *Annual Conference on Neural Information Processing Systems.* 7276–7286.

Jothimurugesan, E., A. Tahmasbi, P. Gibbons, and S. Tirthapura (2018). "Variance-reduced stochastic gradient descent on streaming data". *Annual Conference on Neural Information Processing Systems.* 9928–9937.

Kamp, M., M. Boley, O. Missura, and T. Gärtner (2017). "Effective parallelisation for machine learning". *Annual Conference on Neural Information Processing Systems.* 6480–6491.

Kaoudi, Z., J.-A. Quiane-Ruiz, S. Thirumuruganathan, S. Chawla, and D. Agrawal (2017). "A cost-based optimizer for gradient descent optimization". *International Conference on Management of Data.*

Kara, K., K. Eguro, C. Zhang, and G. Alonso (2018). "ColumnML: Column-store machine learning with on-the-fly data transformation". *Proceedings of the VLDB Endowment.* 12(4): 348–361.

Karakus, C., Y. Sun, S. Diggavi, and W. Yin (2017). "Straggler mitigation in distributed optimization through data encoding". *Annual Conference on Neural Information Processing Systems.* 5440–5448.

Katharopoulos, A. and F. Fleuret (2018). "Not all samples are created equal: Deep learning with importance sampling". *International Conference on Machine Learning.*

Koloskova, A., S. Stich, and M. Jaggi (2019). "Decentralized stochastic optimization and Gossip algorithms with compressed communication". *International Conference on Machine Learning.*

Li, X., T. Zhao, R. Arora, H. Liu, and J. Haupt (2016). "Stochastic variance reduced optimization for nonconvex sparse learning". *International Conference on Machine Learning.*

Li, Y. and M. Yan (2019). "On linear convergence of two decentralized algorithms". *arXiv:1906.07225*.

Li, Y., M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing (2018). "Pipe-SGD: A decentralized pipelined SGD framework for distributed deep net training". *Annual Conference on Neural Information Processing Systems*. 8056–8067.

Li, Z., W. Shi, and M. Yan (2019). "A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates". *IEEE Transactions on Signal Processing*. 67(17): 4494–4506.

Li, Z. and M. Yan (2017). "A primal-dual algorithm with optimal stepsizes and its application in decentralized consensus optimization". *arXiv:1711.06785*.

Lian, X., Y. Huang, Y. Li, and J. Liu (2015). "Asynchronous parallel stochastic gradient for nonconvex optimization". *Annual Conference on Neural Information Processing Systems*. 2737–2745.

Lian, X., C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu (2017). "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent". *Annual Conference on Neural Information Processing Systems*. 5336–5346.

Lian, X., H. Zhang, C.-J. Hsieh, Y. Huang, and J. Liu (2016). "A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order". *Annual Conference on Neural Information Processing Systems*. 3062–3070.

Lian, X., W. Zhang, C. Zhang, and J. Liu (2019). "Asynchronous decentralized parallel stochastic gradient descent". *International Conference on Machine Learning*.

Lim, H., D. G. Andersen, and M. Kaminsky (2019). "3LC: Lightweight and effective traffic compression for distributed machine learning". *Conference on Systems and Machine Learning*.

Ling, Q. and A. Ribeiro (2013). "Decentralized dynamic optimization through the alternating direction method of multipliers". *IEEE Transactions on Signal Processing*. 62(5): 1185–1197.

Liu, J. and S. J. Wright (2015). "Asynchronous stochastic coordinate descent: Parallelism and convergence properties". *SIAM on Optimization*.

Liu, J., S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar (2014). "An asynchronous parallel stochastic coordinate descent algorithm". *International Conference on Machine Learning*.

Liu, S., B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, and L. Amini (2018). "Zeroth-order stochastic variance reduction for nonconvex optimization". *Annual Conference on Neural Information Processing Systems*. 3731–3741.

Liu, X., Y. Li, J. Tang, and M. Yan (2019). "A double residual compression algorithm for efficient distributed learning". *arXiv:1910.07561*.

Ma, C., V. Smith, M. Jaggi, M. Jordan, P. Richtarik, and M. Takac (2015). "Adding vs. averaging in distributed primal-dual optimization". *International Conference on Machine Learning*.

Mahajan, D., J. K. Kim, J. Sacks, A. Ardalan, A. Kumar, and H. Esmaeilzadeh (2018). "In-RDBMS hardware acceleration of advanced analytics". *Proceedings of the VLDB Endowment*. 11(11): 1317–1331.

Nagaraj, D., P. Jain, and P. Netrapalli (2019). "SGD without replacement: Sharper rates for general smooth convex functions". *International Conference on Machine Learning*.

Namkoong, H., A. Sinha, S. Yadlowsky, and J. C. Duchi (2017). "Adaptive sampling probabilities for non-smooth optimization". *International Conference on Machine Learning*.

Nedić, A. and A. Ozdaglar (2009). "Distributed subgradient methods for multi-agent optimization". *IEEE Transactions on Automatic Control*. 54(1): 48–61.

Nguyen, L., P. H. A. Nguyen, M. van Dijk, P. Richtarik, K. Scheinberg, and M. Takac (2018). "SGD and Hogwild! Convergence without the bounded gradients assumption". *International Conference on Machine Learning*.

Palaniappan, B. and F. Bach (2016). "Stochastic variance reduction methods for saddle-point problems". *Annual Conference on Neural Information Processing Systems*. 1416–1424.

Pan, X., M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, and C. Ré (2016). "Cyclades: Conflict-free asynchronous machine learning". *Annual Conference on Neural Information Processing Systems*. 2576–2584.

Peng, H., S. Zhe, X. Zhang, and Y. Qi (2017). "Asynchronous distributed variational Gaussian process for regression". *International Conference on Machine Learning*.

Qu, Z., P. Richtarik, and T. Zhang (2015). "Quartz: Randomized dual coordinate ascent with arbitrary sampling". *Annual Conference on Neural Information Processing Systems*. 865–873.

Raviv, N., R. Tandon, A. Dimakis, and I. Tamo (2018). "Gradient coding from cyclic MDS codes and expander graphs". *International Conference on Machine Learning*.

Reddi, S. J., A. Hefny, S. Sra, B. Poczos, and A. Smola (2016). "Stochastic variance reduction for nonconvex optimization". *International Conference on Machine Learning*.

Shamir, O. (2016). "Without-replacement sampling for stochastic gradient methods". *Annual Conference on Neural Information Processing Systems*. 46–54.

Shi, W., Q. Ling, G. Wu, and W. Yin (2015a). "A proximal gradient algorithm for decentralized composite optimization". *IEEE Transactions on Signal Processing*. 63(22): 6013–6023.

Shi, W., Q. Ling, G. Wu, and W. Yin (2015b). "Extra: An exact first-order algorithm for decentralized consensus optimization". *SIAM Journal on Optimization*. 25(2): 944–966.

Simonetto, A., A. Koppel, A. Mokhtari, G. Leus, and A. Ribeiro (2017). "Decentralized prediction-correction methods for networked time-varying convex optimization". *IEEE Transactions on Automatic Control*. 62(11): 5724–5738.

Simsekli, U., C. Yildiz, T. H. Nguyen, T. Cemgil, and G. Richard (2018). "Asynchronous stochastic quasi-Newton MCMC for non-convex optimization". *International Conference on Machine Learning*.

Sridhar, S., V. Bittorf, J. Liu, C. Zhang, C. Ré, and S. J. Wright (2013). "An approximate efficient solver for LP rounding". *Annual Conference on Neural Information Processing Systems*. 2895–2903.

Stich, S. U., J.-B. Cordonnier, and M. Jaggi (2018). "Sparsified SGD with memory". *Annual Conference on Neural Information Processing Systems*. 4452–4463.

Sun, T., R. Hannah, and W. Yin (2017). "Asynchronous coordinate descent under more realistic assumptions". *Annual Conference on Neural Information Processing Systems*.

Tandon, R., Q. Lei, A. G. Dimakis, and N. Karampatziakis (2017). "Gradient coding: Avoiding stragglers in distributed learning". *International Conference on Machine Learning*.

Tang, H., S. Gan, C. Zhang, T. Zhang, and J. Liu (2018a). "Communication compression for decentralized training". *Annual Conference on Neural Information Processing Systems*. 7663–7673.

Tang, H., X. Lian, S. Qiu, L. Yuan, C. Zhang, T. Zhang, and J. Liu (2019a). "DeepSqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression". *arXiv:1907.07346*.

Tang, H., X. Lian, M. Yan, C. Zhang, and J. Liu (2018b). "D^2: Decentralized training over decentralized data". *International Conference on Machine Learning*.

Tang, H., X. Lian, T. Zhang, and J. Liu (2019b). "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression". *International Conference on Machine Learning*.

Wang, H., S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright (2018a). "ATOMO: Communication-efficient learning via atomic sparsification". *Annual Conference on Neural Information Processing Systems*. 9872–9883.

Wang, H., S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright (2018b). "Atomo: Communication-efficient learning via atomic sparsification". *Annual Conference on Neural Information Processing Systems*. 9850–9861.

Wang, J., M. Kolar, N. Srebro, and T. Zhang (2017). "Efficient distributed learning with sparsity". *International Conference on Machine Learning*.

Wang, J. and G. Joshi (2019). "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD". *Conference on Systems and Machine Learning*.

Wang, N., J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan (2018c). "Training deep neural networks with 8-bit floating point numbers". *Annual Conference on Neural Information Processing Systems*. 7686–7695.

Wang, S., D. Li, Y. Cheng, J. Geng, Y. Wang, S. Wang, S.-T. Xia, and J. Wu (2018d). "BML: A high-performance, low-cost gradient synchronization algorithm for DML training". *Annual Conference on Neural Information Processing Systems*. 4243–4253.

Wang, Z., K. Kara, H. Zhang, G. Alonso, O. Mutlu, and C. Zhang (2019). "Accelerating generalized linear models with MLWeaving: A one-size-fits-all system for any-precision learning". *Proceedings of the VLDB Endowment*. 12(7): 807–821.

Wangni, J., J. Wang, J. Liu, and T. Zhang (2018). "Gradient sparsification for communication-efficient distributed optimization". *Annual Conference on Neural Information Processing Systems*. 1306–1316.

Wen, W., C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li (2017). "TernGrad: Ternary gradients to reduce communication in distributed deep learning". *Annual Conference on Neural Information Processing Systems*. 1508–1518.

Wu, J., W. Huang, J. Huang, and T. Zhang (2018). "Error compensated quantized SGD and its applications to large-scale distributed optimization". *International Conference on Machine Learning*.

Xie, C., S. Koyejo, and I. Gupta (2019). "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance". *International Conference on Machine Learning*.

Xin, D., S. Macke, L. Ma, J. Liu, S. Song, and A. Parameswaran (2018). "HELIX: Holistic optimization for accelerating iterative machine learning". *Proceedings of the VLDB Endowment*. 12(4): 446–460.

Ye, M. and E. Abbe (2018). "Communication-computation efficient gradient coding". *International Conference on Machine Learning*.

Yin, D., Y. Chen, R. Kannan, and P. Bartlett (2018). "Byzantine-robust distributed learning: Towards optimal statistical rates". *International Conference on Machine Learning*.

You, Y., J. Li, J. Hseu, X. Song, J. Demmel, and C. Hsieh (2020). "Reducing BERT pre-training time from 3 days to 76 minutes". *arXiv:1904.00962*.

You, Y., X. Lian, J. Liu, H.-F. Yu, I. S. Dhillon, J. Demmel, and C.-J. Hsieh (2016). "Asynchronous parallel greedy coordinate descent". *Annual Conference on Neural Information Processing Systems.* 4689–4697.

Yu, C., H. Tang, C. Renggli, S. Kassing, A. Singla, D. Alistarh, C. Zhang, and J. Liu (2019). "Distributed learning over unreliable networks". *International Conference on Machine Learning.*

Yu, M., Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr (2018). "GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training". *Annual Conference on Neural Information Processing Systems.*

Yuan, K., Q. Ling, and W. Yin (2016). "On the convergence of decentralized gradient descent". *SIAM Journal on Optimization.* 26(3): 1835–1854.

Zhang, C., P. Patras, and H. Haddadi (2018). "Deep learning in mobile and wireless networking: A survey". *arXiv:1803.04311.*

Zhang, H., J. Li, K. Kara, D. Alistarh, J. Liu, and C. Zhang (2017). "ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning". *International Conference on Machine Learning.*

Zhang, L., T. Yang, R. Jin, Y. Xiao, and Z.-H. Zhou (2016). "Online stochastic linear optimization under one-bit feedback". *International Conference on Machine Learning.*

Zhang, S., A. E. Choromanska, and Y. LeCun (2015). "Deep learning with elastic averaging SGD". *Annual Conference on Neural Information Processing Systems.* 685–693.

Zhao, P. and T. Zhang (2015). "Stochastic optimization with importance sampling for regularized loss minimization". *International Conference on Machine Learning.*

Zheng, S., Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu (2017). "Asynchronous stochastic gradient descent with delay compensation". *International Conference on Machine Learning.*

Zhou, D., P. Xu, and Q. Gu (2018a). "Stochastic nested variance reduced gradient descent for nonconvex optimization". *Annual Conference on Neural Information Processing Systems.*

Zhou, D., P. Xu, and Q. Gu (2018b). "Stochastic variance-reduced cubic regularized Newton methods". *International Conference on Machine Learning*.

Zhou, K., F. Shang, and J. Cheng (2018c). "A simple stochastic variance reduced algorithm with fast convergence rates". *International Conference on Machine Learning*.

Zhou, Z., P. Mertikopoulos, N. Bambos, P. Glynn, Y. Ye, L.-J. Li, and L. Fei-Fei (2018d). "Distributed asynchronous optimization with unbounded delays: How slow can you go?" *International Conference on Machine Learning*.

Zhu, R. (2016). "Gradient-based sampling: An adaptive importance sampling for least-squares". *Annual Conference on Neural Information Processing Systems*. 406–414.

Zhu, R. and Q. Gu (2015). "Towards a lower sample complexity for robust one-bit compressed sensing". *International Conference on Machine Learning*.

Zhu, Y. and J. Lafferty (2018). "Distributed nonparametric regression under communication constraints". *International Conference on Machine Learning*.

Zou, D., P. Xu, and Q. Gu (2018). "Stochastic variance-reduced Hamilton Monte Carlo methods". *International Conference on Machine Learning*.