Angelia Nedić

Distributed Gradient Methods for Convex Machine Learning Problems in Networks

Distributed optimization

Digital Object Identifier 10.1109/MSP.2020.2975210
Date of current version: 28 April 2020

This article provides an overview of distributed gradient methods for solving convex machine learning problems of the form $\min_{x\in\mathbb{R}^n}(1/m)\sum_{i=1}^{m} f_i(x)$ in a system consisting of m agents that are embedded in a communication network. Each agent i has a collection of data captured by its privately known objective function $f_i(x)$. The distributed algorithms considered here obey two simple rules: privately known agent functions $f_i(x)$ cannot be disclosed to any other agent in the network and every agent is aware of the local connectivity structure of the network, i.e., it knows its one-hop neighbors only. While obeying these two rules, the distributed algorithms that agents execute should find a solution to the overall system problem with the limited knowledge of the objective function and limited local communications. Given in this article is an overview of such algorithms that typically involve two update steps: a gradient step based on the agent local objective function and a mixing step that essentially diffuses relevant information from one to all other agents in the network.

Introduction

Artificial intelligence is emerging as the driving technology that will enable future automated transportation, smart cities, and smart power grids, as well as robots that will replace humans in hazardous workplaces and situations. At the core of this technology are the devices that can collect data (i.e., gather measurements from their surrounding), process information, and communicate to share estimates for optimal and reliable system performance. Machine learning is fundamental for data processing, while communication among the devices allows for collaborative learning from all of the data within the system without the need to share the data locally stored at the devices. As such, these systems are inherently distributed. The distributed computational models and coordination mechanisms relying on local communications are central for the development of these technologies.

One possible architecture that can support operations in such distributed systems is peer-to-peer, which can also serve as a base for building more complex hierarchical

architectures. In a peer-to-peer architecture, the consensus protocol has attracted a lot of interest and research in synchronization, formation control, learning, and coordination of multirobot tasks, to name a few. A common ground for addressing these problems is provided by distributed computational models (including machine learning in particular) that use a consensus mechanism as a virtual coordinator.

Artificial intelligence is emerging as the driving technology that will enable future automated transportation, smart cities, and smart power grids.

The goal of this article is to outline the framework of distributed consensus-based gradient methods for convex machine learning problems over static graphs, both directed and undirected. These methods will be discussed in relation to the standard machine learning problem of minimizing the average of the sum of functions

$$\min_{x\in\mathbb{R}^n}\frac{1}{p}\sum_{i=1}^{p} f_i(x), \tag{1}$$

where each function $f_i(x)$ represents the loss associated with a given data point. Specifically, given a collection $\{(z_i, y_i), i = 1, \ldots, p\}$ of data points, where $z_i \in \Re^n$ is a feature vector and $y_i \in \{-1, +1\}$ is its corresponding label, a nonlinear classification problem consists of solving the following minimization problem:

$$\min_{x\in\mathbb{R}^n}\left(c\rho(x)+\frac{1}{p}\sum_{i=1}^{p}\ell(x; z_i, y_i)\right), \tag{2}$$

where $c > 0$ is a regularization parameter, $x = (x_1, \ldots, x_n)$ is a vector, the regularizing function $\rho(x)$ is a strongly convex function (such as the Euclidean norm), and $\ell(x; z_i, y_i)$ is a loss function associated with the data point $(z_i, y_i) \in \mathbb{R}^{n+1}$. The loss function $\ell(x; z, y)$ is typically convex but not necessarily differentiable. For linear classifiers, a common choice is the logistic regression loss function, given by

$$\ell(x; z, y) = \log\left(1 + e^{-y\langle x, z\rangle}\right),$$

with $\langle x, z\rangle$ acting as the inner product of vectors x and z, i.e., $\langle x, z\rangle = \Sigma_{j=1}^{n} x_j z_j$. The resulting problem is known as the *logistic regression* (linear) classifier problem. Another common choice is the hinge-loss function,

$$\ell(x; z, y) = \max\{0, 1 - y\langle x, z\rangle\},$$

giving rise to the maximum margin (linear) classifier problem.

As the preceding discussion shows, the machine learning problem (2) falls under the general formulation in (1), where a function $f_i(x) = c\rho(x) + \ell(x; z_i, y_i)$ is associated with a data point (z_i, y_i). When the number of data points p is not too large, the problem in (1) can be solved using the iterative gradient method, as follows. Starting with some initial guess x^0, at iteration k, we have iterated x^k and compute the gradient

$$\frac{1}{p}\sum_{i=1}^{p}\nabla f_i(x^k) \tag{3}$$

of the objective function at the current iterate. Then, we obtain the new iterate x^{k+1} by moving away from point x^k along the opposite direction of the gradient with some positive step size α_k,

$$x^{k+1} = x^k - \frac{\alpha_k}{p}\sum_{i=1}^{p}\nabla f_i(x^k).$$

Convergence of this simple gradient method is known for the (positive) step-size sequence $\{\alpha_k\}$ satisfying the conditions $\Sigma_{k=0}^{\infty}\alpha_k = \infty$ and $\Sigma_{k=0}^{\infty}\alpha_k^2 < \infty$.

These conditions are satisfied, for example with step size $\alpha_k = 1/k$, or more generally with the step size of the form $\alpha_k = a/k^b$, where $a > 0$ is arbitrary while $b \in ((1/2), 1]$. When the objective function is convex, the gradient method with such a diminishing step size produces an iterate sequence $\{x^k\}$ converging to an optimal point x^* that solves the problem in (1), provided that an optimal point exists (e.g., see [1]). We note that, under some additional assumptions on the objective function, the gradient method with a suitably chosen constant step size $\alpha > 0$ (i.e., $\alpha_k = \alpha$ for all k) produces a sequence of iterates $\{x^k\}$ converging to the solution of the problem (see, for example, the textbook in [2]).

When the number p of the data points is large, computing the full gradient in (3) becomes expensive (time consuming). Alternative approaches have been developed, including the stochastic gradient method that uses $\nabla f_{i_k}(x^k)$ (with an index $i_k \in \{1, 2, \ldots, p\}$ selected uniformly at random) instead of the full gradient. To improve the convergence properties of this basic stochastic gradient method, variance reduced methods have been proposed, including the stochastic variance reduced gradient [3] and the stochastic approximate gradient algorithm [4].

The gradient method and the stochastic variants mentioned previously are centralized in the sense that they are applicable to the situations in which all of the data are located at a single location or the data can be accessed from a central location. In what follows, we will consider the machine learning problem in the case when the data are spatially distributed among several locations and there is no central entity that can access all of the data. Moreover, we assume that the data are private and cannot be shared among the data centers, a situation that arises in applications dealing with medical data, for example. Each data center is viewed as an agent capable of processing its data points locally and exchanging some estimates (extracted from its private data) with the other agents (data centers) over a communication graph. The agents communicate to coordinate their local computations with the other agents in the system, to jointly solve their aggregate problem (to be specified in the next section). In the rest of the article, the core references for the distributed methods over undirected graphs are [5]–[10], while the key references for the methods over directed graphs are [11]–[16].

Distributed gradient method for undirected graphs

The goal of this article is to outline the framework of distributed consensus-based gradient methods for convex machine learning problems over static graphs.

We consider the case when the data points pertinent to the same phenomena or a task are collected at m distinct centers that do not need to be colocated. We refer to these centers as agents labeled by $1, 2, \ldots, m$; we let agent i have the set S_i of data points. The learning problem corresponding to the aggregate data of the system of all agents has the form as in (2), where p is the total number of data points at all agents, i.e., $p = |S_1| + \cdots + |S_m|$, with $|S|$ denoting the cardinality of a finite set S. To account for the agent specific data, by aggregating the data per agent, the problem in (2) can be reformulated, as follows:

$$\min_{x \in \mathbb{R}^n} \frac{1}{m} \sum_{i=1}^{m} f_i(x), \tag{4}$$

where

$$f_i(x) = \frac{m}{p} \sum_{s \in S_i} \left(c\rho(x) + \ell(x; z_s, y_s) \right). \tag{5}$$

Note that (4) and (5) are equivalent to the problem in (2). We assume that the function $f_i : \mathbb{R}^n \to \mathbb{R}$ is the private loss function known only to agent i. Further, we will assume that the m agents are communicating over an undirected network represented by a graph $G = ([m], \mathcal{E})$, where $[m] = \{1, 2, \ldots, m\}$ denotes the set of agents and $\mathcal{E}$ denotes the set of undirected edges. We write $\{i, j\}$ to represent an edge connecting agents i and j.

The agents want to solve the problem in (4) collaboratively in the sense that they are willing to share some estimates with their immediate neighbors in the graph, but they are not willing (or not allowed) to share their data, which essentially means they do not reveal their loss functions f_i.

Given the communication graph $G = ([m], \mathcal{E}),$, the problem can be reformulated as follows:

$$\begin{aligned} \min_{x_i \in \mathbb{R}^n,\, i \in [m]} \quad & \frac{1}{m} \sum_{i=1}^{m} f_i(x_i) \\ \text{subject to} \quad & x_i = x_j \quad \text{for all } \{i, j\} \in \mathcal{E}. \end{aligned} \tag{6}$$

This reformulation is obtained from (4) by assigning a copy x_i of the decision variable x to each agent and by imposing the requirement that these copies should all be the same, i.e., $x_i = x$ for all agents $i \in [m]$. This system of equations is then replaced with an equivalent system of equations requiring that $x_i = x_j$ for all $i, j \in [m]$. When graph G is connected, the system of pairwise equations $x_i = x_j$ for all $i, j \in [m]$ is equivalent to the system of graph-compatible equations $x_i = x_j$ for all $\{i, j\} \in \mathcal{E}$. Thus, when graph G is connected, the problems in (4)–(6) are equivalent. Reformulating (6) serves as a departure point for discussing distributed first-order methods for machine learning problems that use a consensus-type update to enforce the equal-decision constraints in (6).

The objective function of (6) is decoupled, as each f_i depends on its own variable x_i, but these variables are coupled through edge-based constraints. The idea is to distribute the problem among the agents, by allowing each agent to know its neighbors in the graph; that is, every agent i is aware of agents j such that $\{i, j\} \in \mathcal{E}$, which constitute the set $\mathcal{N}_i$ of the neighbors of i in the graph, i.e., $\mathcal{N}_i = \{j \,|\, \{i, j\} \in \mathcal{E}\}$. Using the local agent knowledge of the graph and functions, every agent i can solve its own local part of the overall problem. However, for the agents to collectively solve the overall problem, each agent needs to align its variables with the variables x_j corresponding to its neighbors $j \in \mathcal{N}_i$.

As a mechanism for alignment of the variables, a consensus algorithm is used. The consensus algorithm is a distributed method that the agents can use to asymptotically agree on a decision vector. Specifically, each agent starts with an arbitrary vector $x_i(0)$. At iteration k, every agent sends its current iterate x_i^k to its neighbors $j \in \mathcal{N}_i$ and receives x_j^k from its neighbors $j \in \mathcal{N}_i$. Then, every agent i, executes the consensus update step

$$x_i^{k+1} = a_{ii} x_i^k + \sum_{j \in \mathcal{N}_i} a_{ij} x_j^k, \tag{7}$$

where $a_{ii} > 0$ and $a_{ij} > 0$ are such that $a_{ii} + \Sigma_{j \in \mathcal{N}_i} a_{ij} = 1$. The positive scalars a_{ij}, $j \in \mathcal{N}_i \cup \{i\}$ are referred to as *convex weights* and vector x_i^{k+1} is said to be a convex combination (or a weighted average) of points x_j, $j \in \mathcal{N}_i \cup \{i\}$. Note that the positive weights a_{ij}, $j \in \mathcal{N}_i \cup \{i\}$, are selected by agent i.

To have a more compact representation of the consensus method, let us define an $m \times m$ weight matrix A with entries

$$\begin{aligned} & a_{ii} > 0 \text{ and } a_{ij} > 0 \text{ with } a_{ii} + \sum_{j \in \mathcal{N}_i} a_{ij} = 1 \\ & \text{and} \\ & a_{ij} = 0 \text{ when } j \notin \mathcal{N}_i \cup \{i\}. \end{aligned} \tag{8}$$

Because the nonnegative matrix A has a positive entry in the ijth position only when $\{i, j\}$ is a link in graph G, we say that such a matrix A *is compatible with the structure of graph* G. Using the weights a_{ij} defined in (8), the consensus algorithm in (7) assumes the following form of for every $i \in [m]$,

$$x_i^{k+1} = \sum_{j=1}^{m} a_{ij} x_j^k. \tag{9}$$

The sum of the entries in each row of matrix A is equal to 1. Such a nonnegative matrix is referred to as *row stochastic*.

An illustration of a connected star graph with four agents is given in Figure 1. As an example of a row-stochastic matrix A that is compatible with the structure of the graph in Figure 1, consider the following:

$$A = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}.$$

This matrix is constructed using an equal-weight rule, whereby each agent i gives the same weight to itself and all of its neighbors $j \in \mathcal{N}_i$.

When the graph $G = ([m], \mathcal{E})$ is connected and matrix A is row stochastic and compatible with the graph structure [i.e., it satisfies the relations in (8)], the iterate sequences $\{x_i^k\}, i \in [m]$ generated by the consensus algorithm converge to a same limit point $\tilde{x}$. The limit point $\tilde{x}$, referred to as a *consensus* or *agreement point*, is given as a convex combination of the initial values $\{x_i^0, i \in [m]\}$, i.e.,

$$\tilde{x} = \sum_{i=1}^{m} \pi_i x_i^0.$$

The weight vector $\pi = [\pi_1, \ldots, \pi_m]$ is the unique (normalized) left eigenvector of matrix A corresponding to the eigenvalue $\lambda = 1$, i.e., $\pi A = \pi$. The convergence result is obtained by viewing matrix A as a one-step transition matrix of a Markov chain and employing the ergodicity theory for Markov chains. The detailed proof of the convergence of the consensus algorithm can be found in [5], where the consensus method was originally proposed. If matrix A is doubly stochastic, then $\pi_i = 1/m$ for all i and, consequently, the consensus point is the average of the initial values, i.e., $\tilde{x} = (1/m)\Sigma_{i=1}^m x_i^0$.

Using the consensus method as a mechanism to align the agent variables, a distributed optimization method with a local information exchange can be constructed by executing two steps, one mixing (consensus) (8) and one gradient based. At the beginning of a typical iteration k, every agent sends to its neighbors its current iterate x_i^k to its neighbors $j \in \mathcal{N}_i$ and receives x_j^k from its neighbors $j \in \mathcal{N}_i$. Then, every agent i executes the following two update steps:

$$\begin{aligned} v_i^k &= \sum_{j=1}^{m} a_{ij} x_j^k, \\ x_i^{k+1} &= v_i^k - \alpha_k \nabla f_i(v_i^k), \end{aligned} \tag{10}$$

where the weights a_{ij} satisfy the same relations as in the consensus method [i.e., the relations in (8)] and $\alpha_k > 0$ is a step size. We note that the vector v_i^k is a convex combination of the points x_j, $j \in \mathcal{N}_i \cup \{i\}$. After obtaining v_i^k through the mixing step, every agent i performs a gradient update step using v_i^k to obtain its new iterate x_i^{k+1}. The algorithm in (10) is distributed since every agent updates by using a gradient of its own private function and is local in the sense that it relies on local information exchange. The method is also often referred to as the *consensus-based gradient method*, due to its use of a mixing step that resembles the distributed consensus process.

The distributed algorithm (10) can be viewed as an extension of the gradient method, in which the mixing step is introduced to align the agents' iterates. This step plays the role of "virtual coordinator" of the agents iterates in a system that does not have a central coordinator. To shed more light on this issue, let us take a look at what is happening with the iterates of the distributed gradient method on the system level. Consider the iterate sequences $\{x_i^k\}, i \in [m]$, produced by the method at all agents in the system. By taking the average of these iterates [cf. (10)] across all of the agents, at any given instance, we have

$$\begin{aligned} \frac{1}{m}\sum_{i=1}^{m} x_i^{k+1} &= \frac{1}{m}\sum_{i=1}^{m} v_i^k - \frac{\alpha_k}{m}\sum_{i=1}^{m} \nabla f_i(v_i^k) \\ &= \frac{1}{m}\sum_{j=1}^{m}\Big(\sum_{i=1}^{m} a_{ij}\Big)x_j^k - \frac{\alpha_k}{m}\sum_{i=1}^{m} \nabla f_i(v_i^k), \end{aligned}$$

where the second equality is obtained by using the definition of v_i^k in (10) and by exchanging the order of the resulting two sums. When matrix A is doubly stochastic, we have $\Sigma_{i=1}^m a_{ij} = 1$ for all j, thus yielding

$$\frac{1}{m}\sum_{i=1}^{m} x_i^{k+1} = \frac{1}{m}\sum_{j=1}^{m} x_j^k - \frac{\alpha_k}{m}\sum_{i=1}^{m} \nabla f_i(v_i^k).$$

By letting $\bar{x}^k$ denote the iterate average across the agents at time k, i.e.,

$$\bar{x}^k = \frac{1}{m}\sum_{i=1}^{m} x_i^k,$$

the preceding relation can be written as

$$\bar{x}^{k+1} = \bar{x}^k - \frac{\alpha_k}{m}\sum_{i=1}^{m} \nabla f_i(v_i^k), \tag{11}$$

which is nearly the same as the (centralized) gradient method update for solving (6). The only difference is that the gradient of f_i is computed at point v_i^k instead of $\bar{x}^k$. More concretely, by adding and subtracting the correct gradients, for the averaged iterates we have

$$\bar{x}^{k+1} = \bar{x}^k - \frac{\alpha_k}{m}\sum_{i=1}^{m} \nabla f_i(\bar{x}^k) + \epsilon^k, \;\; \epsilon^k = \frac{\alpha_k}{m}\sum_{i=1}^{m} (\nabla f_i(\bar{x}^k) - \nabla f_i(v_i^k)).$$

The averaged iterates $\bar{x}^{k+1}$ will follow an erroneous gradient method in which the error is proportional to the average gradient difference $\nabla f_i(\bar{x}^k) - \nabla f_i(v_i^k)$, as long as we can ensure

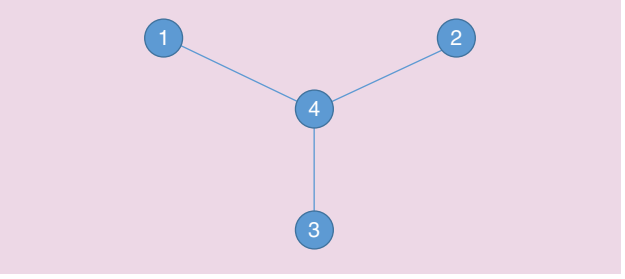

FIGURE 1. A star graph with four agents.

that the error ϵ_k remains small. In fact, under some additional assumptions regarding the functions and step size, the mixing step can ensure that the error ϵ_k does not blow up.

The role of the mixing matrix A is critical since it guarantees that the following properties hold under suitable conditions on the step size and the functions f_i (such as the convexity):

1) the averaged iterate sequence $\{\bar{x}^k\}$ converges to a solution x^* of (6)
2) the disagreement sequence $\{\|x_i^k - \bar{x}^k\|\}$ converges to zero for every agent i.

Thus, iterates $x_i^k, i \in [m]$ generated by the distributed method track the averaged process with increasing accuracy, while the average process converges to a solution. Consequently, the iterate sequences $\{x_i^k\}, i \in [m]$ all converge to the same point, which is a solution of (6).

From the view point of the consensus process in (9), the distributed gradient method in (10) can be interpreted as a consensus-based process that is affected by two forces: the consensus force represented by the mixing step (influenced by matrix A) and the agent-based gradient forces coming from the objective functions f_i. In this case, the gradients steer the limiting consensus point to a solution x^* of (6).

The consensus algorithm was originally proposed in [5] to model opinion dynamics, leading to an agreement in a team of individuals. It was later used for estimation, control, and optimization starting with works in [6], [17], and [18]. In contrast to the machine learning problem considered in this article, the optimization problem considered in [6] and [18] involves a nonseparable objective, i.e., the problem of the form

$$\min_{x=(x_1,\ldots,x_m)\in\mathbb{R}^n} f(x_1,\ldots,x_m),$$

where the decision vector $x \in \mathbb{R}^n$ is partitioned among m agents. The first variants of the distributed gradient methods that employ a mixing (consensus) step to solve the problem with a separable structure [i.e., in the form of a machine learning problem (4)] was pioneered in [7] and [8]. Subsequently, many distributed methods have been developed, such as those with a faster convergence [19], [20] and with emphasis on the communication and computation tradeoffs [21], [22]. A recent survey [10] thoroughly reviews the consensus problem and variations and extensions of the distributed method that account for time-varying graphs, delays and noisy (or quantized) communication links, noisy (stochastic) gradient evaluations, as well as some asynchronous methods.

The mixing step of the distributed method in (10) is also referred to as a *diffusion* since it allows for the local agent information to diffuse over the entire network. The mixing and gradient update steps in (10) can be exchanged, leading to the following alternative variant of the distributed method:

$$v_i^k = x_i^k - \alpha_k \nabla f_i(x_i^k),$$
$$x_i^{k+1} = \sum_{j=1}^{m} a_{ij} v_i^k. \tag{12}$$

We conclude this section by summarizing the basic advantages and limitations of the distributed method (10).

Viewing the gradient update step as an adaptation, the distributed method in (10) is also referred to as *combine-then-adapt* diffusion strategy, while the method in (12) is referred to as the *adapt-then-combine* diffusion strategy. One can find an elegant exposure of these strategies in [9] and follow the references therein for technical details of their analysis.

The convergence rate of the distributed method in (10) is of the order of $O(\log k/\sqrt{k})$ in the number k of iterations, which is dictated by the use of the diminishing step size satisfying the typical conditions in gradient methods, namely $\Sigma_{k=0}^{\infty} \alpha_k = \infty$ and $\Sigma_{k=0}^{\infty} \alpha_k^2 < \infty$. A centralized gradient method for a convex minimization problem that uses such a diminishing step size α_k converges in the order of $O\left(1/\sqrt{k}\right)$. Thus, the distributed method is slower due to the $\log k$ term, which in turn is due to the extra mixing step in (10) that copes with the distributed knowledge of the overall objective function. Moreover, the hidden constant factor in $O(\log k/\sqrt{k})$ depends critically on graph G's connectivity structure and the spectral properties of the mixing matrix A. A more in-depth discussion on how the graph topology affects the performance of the method can be found in a recent survey [23].

We conclude this section by summarizing the basic advantages and limitations of the distributed method (10). The method is considered to be slow since its convergence rate is in the order of $O(\log k/\sqrt{k})$, resulting from the use of a diminishing step size. However, the method is suitable for situations in which noise or errors are present due to noisy or failure-prone links, or due to the use of stochastic gradients. In such situations, the diminishing step size is beneficial, as it attenuates the noise.

There are two major limitations of the method (10).

1) It requires the use of a doubly stochastic matrix A to solve the problem of minimizing the average sum $(1/m)\Sigma_{i=1}^{m} f_i(x)$ of the agent objectives. The construction of such a matrix in a distributed fashion is a nontrivial problem when the underlying communication graph G is directed, as noted in [24]. To avoid this obstacle, one can consider a distributed gradient method that employs an alternative consensus process in the mixing step and does not require a doubly stochastic matrix, as discussed in the "Distributed Gradient Method for Directed Graphs" section.
2) In general, the method does not converge with a constant step size (i.e., $\alpha_k = \alpha$ for all k), which has been observed in [25]. Therefore, it cannot match the performance of the centralized gradient methods that use such step sizes and have a convergence rate faster than $O(1/\sqrt{k})$. A remedy for this drawback is to allow the agents to exchange and mix both the decision vectors and some suitably defined estimates of the gradient directions, as seen in the "Gradient-Tracking Algorithms" section.

Distributed gradient method for directed graphs

When the underlying communication graph G is directed, the distributed method of the "Distributed Gradient Method for

Undirected Graphs" section, which requires a doubly stochastic matrix A, cannot be easily implemented since the construction of matrix A in a distributed way becomes computationally expensive. To deal with directed graphs, one can resort to a different consensus algorithm that makes use of a column stochastic weight matrix A and combine it with a gradient method.

These algorithms are not readily rendering accelerated variants that can perform nearly as quickly as the centralized gradient method.

Specifically, the consensus algorithm is built using a column stochastic matrix A that is compatible with the given directed graph G, i.e., for all $j \in [m]$,

$$a_{jj} > 0 \text{ and } a_{ij} > 0 \text{ with } a_{jj} + \sum_{i \in \mathcal{N}_j^{\text{out}}} a_{ij} = 1 \text{ and}$$
$$a_{ij} = 0 \text{ when } i \notin \mathcal{N}_j^{\text{out}} \cup \{j\}, \tag{13}$$

where $\mathcal{N}_j^{\text{out}}$ is the set of all out-neighbors of agent j, i.e., the agents i that can be reached from j by a link $(j, i) \in \mathcal{E}$); formally, $\mathcal{N}_j^{\text{out}} = \{i \,|\, (j, i) \in \mathcal{E}\}$. In this case, agent j chooses the weights in the jth column of A and ensures that its entries sum to 1. A common choice is to let all of the values $a_{ij}, i \in \mathcal{N}_j^{\text{out}} \cup \{j\}$ be the same and equal to the cardinality of the set $\mathcal{N}_j^{\text{out}} \cup \{j\}$.

Having such a matrix A, a consensus method that has two variables is constructed. In particular, each agent i maintains variables x_i^k and y_i^k at time k. At time $k+1$, each agent j sends $a_{ij}x_j^k$ and $a_{ij}y_j^k$ to all of its out-neighbors $i \in \mathcal{N}_j^{\text{out}}$. After that, every agent i updates by simply summing the x and y variables it has received from its in-neighbors, i.e.,

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{in}} \cup \{i\}} a_{ij}x_j^k,$$
$$y_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{in}} \cup \{i\}} a_{ij}y_j^k, \tag{14}$$

where $\mathcal{N}_i^{\text{in}}$ is the set of in-neighbors of agent i, i.e.,

$$\mathcal{N}_i^{\text{in}} = \{j \,|\, (j, i) \in \mathcal{E}\}.$$

The algorithm in (14) is known as the *push–sum method*, since every agent pushes some information along its outgoing links, while it sums the incoming information. The algorithm will lead to consensus, as long as graph G is strongly connected and the y variables are initiated with $y_i^0 = 1$ for all i. The x variables can be vectors, while y variables are always scalars. The variable that reaches a consensus is in fact a ratio of x and y variables at each agent; that is, the variables

$$z_i^k = \frac{x_i^k}{y_i^k} \text{ for all } i \in [m]$$

converge to a common consensual point as the number k of the iterations increases to infinity. For this reason, the method is also referred to as a *ratio consensus*. The first push–sum algorithm has been proposed in [11] for a randomized gossip-type implementation. Subsequently, it has been further investigated in [12] and [26]–[28] for networks with a time-varying communication structure including the case of unreliable (packet-dropping) links.

One can use the push–sum consensus to construct a distributed gradient method for solving machine learning problem (6), where each function f_i is a private local objective function of agent i. Starting with arbitrary variables $x_i^0, i \in [m]$ and $y_i^0 = 1$ for all i, the algorithm proceeds as follows. Each agent j sends $a_{ji}x_j^k$ and $a_{ji}y_j^k$ to its out neighbors [i.e., the agents i that can be reached by a link $(j, i) \in \mathcal{E}$)]. After this communication step, all agents update their x and y variables as follows:

$$v_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{in}} \cup \{i\}} a_{ij}x_j^k,$$
$$y_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{in}} \cup \{i\}} a_{ij}y_j^k,$$
$$z_i^{k+1} = \frac{v_i^{k+1}}{y_i^{k+1}},$$
$$x_i^{k+1} = v_i^{k+1} - \alpha_k \nabla f(z_i^{k+1}). \tag{15}$$

This method will be referred to as the *gradient–push method*. To gain the intuition behind this algorithm, consider dividing the update relation for x_i^{k+1} by y_i^{k+1}, which in view of the definition of z_i^{k+1} in (15) yields

$$\frac{x_i^{k+1}}{y_i^{k+1}} = z_i^{k+1} - \gamma_i^k \nabla f(z_i^{k+1}),$$

where $\gamma_i^k = \left(\alpha_k \big/ y_i^{k+1}\right) > 0$. As the z variables converge to a common point $\tilde{x}$ and the step size α_k is diminishing, under suitable conditions on the gradients of the function f_i (such as boundedness), the ratio $x_i^{k+1} \big/ y_i^{k+1}$ also converges to point $\tilde{x}$. Additionally, the convexity of the functions f_i and the correct selection of the step size α_k are important to ensure that the consensual point $\tilde{x}$ is in fact a solution to the aggregate-agent learning problem of minimizing the average sum $(1/m)\sum_{i=1}^m f_i(x)$ of their local objective functions [i.e., the problem in (4)]. While the intuition behind the gradient–push method in (15) is simple, its formal analysis is quite involved.

The work in [13] and [14] is the first to provide a distributed method that makes use of the push–sum consensus protocol to cope with directed graphs. There, the dual-averaging method of Nesterov has been used in the construction and its convergence properties have been established for a static directed graph. Subsequently, in [15] a simple (sub)gradient method using the push–sum consensus has been proposed for a general sequence of time-varying directed graphs and is later extended to a stochastic method in [16]. The work in [15] and [16] establishes the convergence rate of $O\left(\log k \big/ \sqrt{k}\right)$ and $O(\log k \big/ k)$ for the case when the functions f_i are convex and strongly convex, respectively. We note that while the work in [15] and [16] uses the special weight matrix A_k (equal-out-neighbor weights), it is straightforward to extend it to the case of an arbitrary sequence

of column stochastic matrices that are compliant with the topology of the underlying graphs, and have nonzero entries that are uniformly lower-bounded away from zero. A more comprehensive review of the literature and the results for the push–pull consensus-based methods is given in a recent survey article [23].

The gradient–push method in (15) suffers from the same slow convergence as that of the distributed gradient method (10) due to their use of the diminishing step size. The advantages of the gradient–push method are the same as those of the distributed gradient method, mainly in noisy settings. If the step size is constant, both distributed gradient (10) and gradient–push (15) methods may fail to converge, while the constant step is desirable for a faster convergence. Thus, these algorithms are not readily rendering accelerated variants that can perform nearly as quickly as the centralized gradient method.

Despite the prolific research on distributed methods over the past decade, there still are many directions for further exploration.

Another explanation for the inability of these methods to be faster can be observed from their somewhat greedy (local) view of the objective function. In particular, in both of the distributed methods [cf. (10) and (15)], the agents use mixing (with their neighbors) for the decision variables, but the decision variables are updated at every agent i using only a local objective function gradient ∇f_i. Thus, with respective to the global objective function, which is proportional to $\Sigma_{i=1}^m f_i(x)$, the agents perform a greedy update. To enhance the performance of these methods, the agents need to be less greedy, as we will see in the next section.

Gradient-tracking algorithms

In this section, we assume that the agents are aware that they comprise a system, while they need not be aware of the global connectivity structure nor the global objective function $\Sigma_{i=1}^m f_i(x)$. However, by being aware that there is a system objective, in addition to mixing their decision variables, they also use and mix directions that serve as estimates of the global objective gradient. This extra mixing step of directions must be properly constructed to track the gradients of the system objective.

Suppose that graph G is undirected and the mixing matrix is compatible with the graph [see (8)]. Then, ideally, if the agents had access to all of the objective functions, each agent would implement the distributed method in (10) by using the gradient of the system objective, i.e., every agent i would execute the following two steps:

$$v_i^k = \sum_{j=1}^m a_{ij} x_j^k,$$
$$x_i^{k+1} = v_i^k - \frac{\alpha}{m} \sum_{i=1}^m \nabla f_i(v_i^k).$$

The step size here is fixed since we are interested in faster gradient methods that do not use a diminishing step. In the absence of the knowledge of all functions f_i, every agent can track the aggregate function gradients, by employing a direction d_i^k and tracking the average of these directions. In particular, every agent will update as follows. At time $k+1$, the agents have x_i^k and d_i^k, and exchange these vectors with their neighbors in the graph. Then, they will compute x_i^{k+1} and d_i^{k+1}, as follows:

$$x_i^{k+1} = \sum_{j=1}^m a_{ij} x_j^k - \alpha d_i^k,$$
$$d_i^{k+1} = \sum_{j=1}^m a_{ij} d_j^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k). \tag{16}$$

We note that each agent still computes the gradients of its own local objective function f_i, but through the mixing of the directions in (IV) it tracks the sum of the gradients of the overall system function objective. Subtracting the past gradient $\nabla f_i(x_i^k)$ in (16) is used to ensure that only new gradient information, captured by the difference $\nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)$, influences the direction update.

To ensure correct gradient tracking, the method has to be initialized with $d_i^0 = \nabla f_i(x_i^0)$ for all agents $i \in [m]$, while x_i^0 can be arbitrary. With such an initialization, when matrix A is column stochastic, the directions d_i^{k+1} satisfy

$$\sum_{i=1}^m d_i^{k+1} = \sum_{i=1}^m \nabla f_i(x_i^{k+1}),$$

which can be shown by using the mathematical induction on k.

Thus, the average iterates $\bar{x}^k = (1/m)\Sigma_{i=1}^m x_i^k$ satisfy $\bar{x}^{k+1} = \bar{x}^k - (\alpha/m)\Sigma_{i=1}^m \nabla f_i(x_i^k)$, which can be interpreted as generated by an erroneous centralized gradient method

$$\bar{x}^{k+1} = \bar{x}^k - (\alpha/m)\Sigma_{i=1}^m \nabla f_i(\bar{x}^k) + \mathbf{e}_k,$$
with error $\mathbf{e}_k = (\alpha/m)\Sigma_{i=1}^m (\nabla f_i(\bar{x}^k) - \nabla f_i(x_i^k))$.

These errors are well behaved, even for the constant step, due to an additional alignment of the directions in (16). In particular, work in [29] studies the method for smooth convex functions with Lipschitz continuous gradients and shows the convergence of the method in the order of $O(1/k)$ when a constant step size α is suitably chosen. (A function f has Lipschitz continuous gradients with a constant L if $\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\|$ for all $x, y \in \mathbb{R}^n$.)

Under additional requirement that each f_i is strongly convex, in [30] it is shown that the method (with a carefully selected step size α) produces iterates x_i^k that converge to the point x^* minimizing the global objective function $(1/m)\Sigma_{i=1}^m f_i(x)$ with a geometric rate, i.e., $\|x_i^k - x^*\| \le q^k C$ for all agents i and all $k \ge 0$, where $q \in (0,1)$ and C are some constants that depend on the graph connectivity topology, the entries of matrix A, the step size α, and the parameters characterizing the function f_i (such as the strong convexity and Lipschitz gradient constants). Thus, the distributed method with the gradient-tracking mechanism

matches the fastest convergence rate of the centralized gradient method (which is geometric).

When graph G is directed, the gradient–push method can also be augmented to include a gradient tracking mechanism. In particular, the gradient–push method in (15) assumes the form

$$\begin{aligned} v_i^{k+1} &= \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} (a_{ij}v_j^k - \alpha d_j^k),\\ y_i^{k+1} &= \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}y_j^k,\\ x_i^{k+1} &= \frac{v_i^{k+1}}{y_i^{k+1}},\\ d_i^{k+1} &= \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}d_j^k + \nabla f(x_i^{k+1}) - \nabla f(x_i^k). \end{aligned} \tag{17}$$

For a column stochastic matrix A and under the conditions of strong convexity and Lipschitz continuous gradients of the functions f_i, similar to that of the distributed gradient-tracking method in (16). This algorithm also enjoys a geometric convergence rate, which is analysis of these distributed gradient-tracking methods shown in [30] for a more general case in which the underlying communication graphs and matrix sequence are time varying.

Yet another general direction of research is the development and investigation of asynchronous implementations of the distributed methods.

Simultaneously and independently of [29] and [30], the idea of tracking the gradients has been employed in [31], where a class of distributed algorithms have been developed for solving nonconvex problems in networks. In summary, the distributed methods with gradient tracking in (16) and (17) can match the best known convergence rates of centralized gradient methods. The main distinction between the two methods is that the method in (16) works on undirected graphs, where the weight matrix A is required to be doubly stochastic while the method in (17) works on directed graphs, where the weight matrix A is required to be column stochastic.

We note that each of the two tracking methods employs the same weight matrix A when mixing the x variables and the direction related d variables. We next consider an interesting gradient-tracking method that makes use of different matrices to mix the x and d variables. In particular, the method works for a directed strongly connected graph G. At iteration k, every agent i has two vectors x_i^k and d_i^k. At the time of iteration $k+1$, each i agent obtains (pulls) $x_j^k - \alpha_j d_j^k$ from its in-neighbors $j \in \mathcal{N}_i^{\text{in}}$, and sends (pushes) $b_{\ell i}d_i^k$ to each of its out-neighbors $\ell \in \mathcal{N}_i^{\text{out}}$. Upon this information exchange, the agents update as follows:

$$\begin{aligned} x_i^{k+1} &= \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}(x_j^k - \alpha_j d_j^k) \qquad (A\text{-pull}),\\ d_i^{k+1} &= \sum_{\ell\in\mathcal{N}_i^{\text{out}}\cup\{i\}} b_{\ell i}d_j^k + \nabla f(x_i^{k+1}) - \nabla f(x_i^k) \qquad (B\text{-push}). \end{aligned} \tag{18}$$

Note that in this method, every agent uses its own step size α_i, which need not be coordinated with the step sizes of the other agents. For the algorithm to work properly, the initial directions are set to $d_i^0 = \nabla f_i(x_i^0)$ for all i, while $x_i^0 \in \mathbb{R}^n$, $i \in [m]$ are arbitrary initial decision vectors.

Agent i chooses the positive weights a_{ij}, $j \in \mathcal{N}_i^{\text{in}} \cup \{i\}$ for its in-neighbors involved in the pull step and also the positive weights $b_{\ell i}$, $\ell \in \mathcal{N}_i^{\text{out}} \cup \{i\}$ for its out-neighbors involved in the push step. By defining these weights to be zero for all other pairs (i, j), we obtain $m \times m$ matrices compatible with the structure of graph G. Specifically, it is

$$\begin{aligned} &a_{ij} > 0 \text{ for all } j \in \mathcal{N}_i^{\text{in}} \cup \{i\}, \text{ and } a_{ij} = 0 \text{ otherwise},\\ &b_{\ell i} > 0 \text{ for all } \ell \in \mathcal{N}_i^{\text{out}} \cup \{i\}, \text{ and } b_{\ell i} = 0 \text{ otherwise}. \end{aligned}$$

The interesting aspect of the *A*pull–*B*push method is that the matrix $A = [a_{ij}]$ is row stochastic while the matrix $B = [b_{ij}]$ is column stochastic, i.e.,

$$\sum_{j=1}^m a_{ij} = 1, \quad \sum_{\ell=1}^m b_{\ell i} = 1 \quad \text{for all } i \in [m].$$

Here, we have coined the name of the algorithm to be *A*pull–*B*push to fairly capture an independent development of two closely related methods, namely a push–pull method [cf. (18)] as given in [32] and a variant proposed and analyzed in [33], which performs slightly different updates. In particular, the algorithm in [33] is given by

$$\begin{aligned} x_i^{k+1} &= \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}x_j^k - \eta d_i^k,\\ d_i^{k+1} &= \sum_{\ell\in\mathcal{N}_i^{\text{out}}\cup\{i\}} b_{ij}(d_j^k + \nabla f_j(x_j^{k+1}) - \nabla f_j(x_j^k)). \end{aligned} \tag{19}$$

Both algorithms use a row stochastic matrix A in updating the x variables and a column stochastic matrix $B = [b_{ij}]$ in the updates of the directions. Also, with suitably selected step sizes, both algorithms converge linearly to the solution of the problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^m f_i(x)$$

when all of the functions f_i are strongly convex and have Lipschitz continuous gradients (see [32] and [33] for details).

In terms of communication requirements and the size of the variables exchanged among the agents, the distributed method for undirected graphs (see the "Distributed Gradient Method for Undirected Graphs" section) requires $2|\mathcal{E}|$ communications per iteration since each link in the graph is used twice in every update time; the x variables exchanged are of size n. The distributed method for directed graphs (see the "Distributed Gradient Method for Directed Graphs" section), requires $|\mathcal{E}|$ communications per iteration since each link in the graph is used once in

every update time; the variables exchanged are of size $n + 1$ (x variable has size n and the auxiliary y variable is scalar).

The distributed algorithm in (16) requires exchanging x and d variables with the total size of $2n$ and $2|E|$ communications among the agents per iteration (since the graph is undirected). The algorithm in (17) requires exchanging x and d variables, and an additional scalar y variable, yielding a total size of $2n + 1$. It requires $|E|$ communications among the agents per iteration since the graph is directed. Finally, the algorithm in (19) requires exchanging x and d variables with the total size of $2n$ and requires a subset of links forming a spanning tree: about 2 m communications per update (see [32] for precise assumptions). Overall, the communication requirements for the algorithms in this section are similar to those for the algorithms of the "Distributed Gradient Method for Undirected Graphs" and "Distributed Gradient Method for Directed Graphs" sections. However, in every communication, directed or not, the algorithms of this section require twice the larger size of the variables exchanged than those of the algorithms of the "Distributed Gradient Method for Undirected Graphs" and "Distributed Gradient Method for Directed Graphs" sections.

Some random gossip- and broadcast-gossip methods have been studied in the literature, but those studies are somewhat limited.

Some open directions for further research

Some of the basic distributed methods for solving machine learning problems are discussed in this article. The algorithms obey two main principles: the local knowledge of the graph structure (reflected in communications with one-hop neighbors only) and (private) knowledge of its own objective function only. To obey with these principles, the distributed algorithms considered here must use some form of the mixing or diffusion step that propagates the required information from one to all of the agents in the network. This mixing step combined with some basic gradient method is the basic idea in all of the distributed methods discussed here. Other variants of distributed method can be constructed, such as those based on the alternating direction method of multipliers approach. (See the recent survey article [23] for relevant literature.)

In practice, several issues arise in implementing the distributed methods due to the use of a communication network, such as quantization and link delays. These two aspects are currently of great interest as they are the bottleneck for the efficiency of the methods. Studies of the effects of quantization in distributed methods have been somewhat limited in the past decade [34]–[36] and very recently in [37]. The effects of communication delays for distributed computational systems have been considered in [38], while [39] considers the impact of delays in the weighted-average consensus method. Very recently, a distributed algorithm that factors in link delays in communications has been proposed and investigated in [40]. The method identifies and communicates more frequently over critical links and saves communication time by using other links less frequently. Such communication efficient approaches have not yet been studied for the recently developed methods that employ a gradient-tracking mechanism (i.e., the methods discussed in the "Gradient-Tracking Algorithms" section).

Despite the prolific research on distributed methods over the past decade, there still are many directions for further exploration. For example, the gradient–push method in (15) and its variant with gradient-tracking mechanism in (17) have been analyzed for solving an unconstrained machine learning problem (4) over time-varying undirected graphs [15], [30]. It is interesting to see how these methods should be modified and analyzed for a constrained learning problem in which the constraint set X may itself be distributed. That is, set X has the form $X = \cap_{i=1}^{m} X_i$ and each X_i is known to one agent only, where X_i is either simple (easy to be projected onto) or is given as an intersection of a collection of convex (algebraic) functional inequalities.

Another direction of research is posed by the *A*pull–*B*push method in (18), which has been analyzed for a static (directed) communication graph only [32], [33]. It would be desirable to see its performance over time-varying directed graphs, which would extend the method to a more realistic scenarios when the graph connectivity structure may change.

Yet another general direction of research is the development and investigation of asynchronous implementations of the distributed methods, where the agents' information exchange and updates do not need to be synchronous. Some random gossip- and broadcast-gossip methods have been studied in the literature, but those studies are somewhat limited. (See [23] for more detailed account of such randomized methods.)

Acknowledgments

This work was partially funded by the National Science Foundation under grant CCF-1717391 and the U.S. Navy under award N000141612245.

Author

Angelia Nedić (Angelia.Nedich@asu.edu) received her Ph.D. degrees from Moscow State University, Russia, in computational mathematics and mathematical physics in 1994 and from the Massachusetts Institute of Technology, Cambridge, in electrical and computer science engineering in 2002. She is a faculty member with the School of Electrical, Computer, and Energy Engineering at Arizona State University, Tempe. Prior to joining Arizona State University, she was a Willard Scholar faculty member with the University of Illinois at Urbana-Champaign. She received Best Paper Awards (jointly with coauthors) at the Winter Simulation Conference 2013 and International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks 2015. Her general

research interests are in optimization, large-scale complex systems dynamics, equilibrium, and variational inequality problems. She is an associate editor of *SIAM Journal on Optimization*, *Operations Research*, and *IEEE Transactions on Signal and Information Processing Over Networks*. She is a Member of the IEEE.

References

[1] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

[2] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. New York: Springer-Verlag, 2013.

[3] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. 26th Int. Conf. Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds. 2013, pp. 315–323. doi: 10.5555/2999611.2999647.

[4] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. 27th Int. Conf. Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Red Hook, NY: Curran Associates, 2014, pp. 1646–1654. doi: 10.5555/2968826.2969010. [Online]. Available: http://papers.nips.cc/paper/5258-saga-a-fast-incremental-gradient-method-with-support-for-non-strongly-convex-composite-objectives.pdf

[5] M. DeGroot, "Reaching a consensus," *J. Amer. Stat. Assoc.*, vol. 69, no. 345, pp. 118–121, 1974. doi: 10.1080/01621459.1974.10480137.

[6] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Autom. Control*, vol. 31, no. 9, pp. 803–812, 1986. doi: 10.1109/TAC.1986.1104412.

[7] A. H. Sayed and C. G. Lopes, "Adaptive processing over distributed networks," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E90-A, no. 8, pp. 1504–1510, 2007. doi: 10.1093/ietfec/e90-a.8.1504.

[8] A. Nedic´ and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, 2009. doi: 10.1109/TAC.2008.2009515.

[9] A. Sayed, *Adaptation, Learning, and Optimization Over Networks* (Foundations and Trends in Machine Learning), vol. 7. Hanover, MA: Now Publishers, 2014.

[10] A. Nedic´ and J. Liu, "Distributed optimization for control," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, no. 1, pp. 77–103, 2018. doi: 10.1146/annurev-control-060117-105131.

[11] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. 44th Annu. IEEE Symp. Foundations of Computer Science*, 2003, pp. 482–491. doi: 10.1109/SFCS.2003.1238221.

[12] A. Dominguez-Garcia and C. Hadjicostis, "Distributed strategies for average consensus in directed graphs," in *Proc. 50th IEEE Conf. Decision and Control and European Control*, Dec. 2011, pp. 2124–2129. doi: 10.1109/CDC.2011.6160462.

[13] K. Tsianos, S. Lawlor, and M. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *Proc. IEEE Conf. Decision and Control*, 2012, pp. 5453–5458. doi: 10.1109/CDC.2012.6426375.

[14] K. Tsianos, "The role of the network in distributed optimization algorithms: Convergence rates, scalability, communication/computation tradeoffs and communication delays," Ph.D. dissertation, Dept. Electrical Computer Engineering, McGill Univ., Montreal, 2013.

[15] A. Nedic´ and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 601–615, 2015. doi: 10.1109/TAC.2014.2364096.

[16] A. Nedic´ and A. Olshevsky, "Stochastic gradient–push for strongly convex functions on time-varying directed graphs," *IEEE Trans. Autom. Control*, vol. 61, no. 12, pp. 3936–3947, 2016. doi: 10.1109/TAC.2016.2529285.

[17] V. Borkar and P. Varaiya, "Asymptotic agreement in distributed estimation," *IEEE Trans. Autom. Control*, vol. 27, no. 3, pp. 650–655, 1982. doi: 10.1109/TAC.1982.1102982.

[18] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Dept. Elect. Eng. Computer Science, Massachusetts Institute of Technology, Cambridge, 1984.

[19] A. I. Chen and A. Ozdaglar, "A fast distributed proximal-gradient method," in *Proc. IEEE 50th Annu. Allerton Conf. Communication, Control, and Computing*, 2012, pp. 601–608. doi: 10.1109/Allerton.2012.6483273.

[20] D. Jakovetic´, J. Xavier, and J. Moura, "Fast distributed gradient methods," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1131–1146, May 2014. doi: 10.1109/TAC.2014.2298712.

[21] K. Tsianos, S. Lawlor, and M. Rabbat, "Communication/computation tradeoffs in consensus-based distributed optimization," in *Proc. Advances in Neural Information Processing Systems*, Lake Tahoe, Dec. 2012, pp. 1943–1951. doi: 10.5555/2999325.2999352.

[22] A. Berahas, R. Bollapragada, N. Keskar, and E. Wei, "Balancing communication and computation in distributed optimization," *IEEE Trans. Automat. Control*, vol. 64, no. 8, pp. 3141–3155, 2018. doi: 10.1109/TAC.2018.2880407.

[23] A. Nedic´, A. Olshevsky, and M. Rabbat, "Network topology and communication–Computation tradeoffs in decentralized optimization," *Proc. IEEE*, vol. 106, no. 5, pp. 953–976, 2018. doi: 10.1109/JPROC.2018.2817461.

[24] B. Gharesifard and J. Cortés, "Distributed strategies for generating weight-balanced and doubly stochastic digraphs," *Eur. J. Control*, vol. 18, no. 6, pp. 539–557, 2012. doi: 10.3166/EJC.18.539-557.

[25] W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first order algorithm for decentralized consensus optimization," *SIAM J. Optim.*, vol. 25, no. 2, pp. 944–966, 2015. doi: 10.1137/14096668X.

[26] F. Benezit, V. Blondel, P. Thiran, J. Tsitsiklis, and M. Vetterli, "Weighted gossip: Distributed averaging using non-doubly stochastic matrices," in *Proc. 2010 IEEE Int. Symp. Information Theory*, June 2010, pp. 1753–1757. doi: 10.1109/ISIT.2010.5513273.

[27] C. N. Hadjicostis and T. Charalambous, "Average consensus in the presence of delays in directed graph topologies," *IEEE Trans. Autom. Control*, vol. 59, no. 3, pp. 763–768, 2014. doi: 10.1109/TAC.2013.2275669.

[28] C. N. Hadjicostis, N. H. Vaidya, and A. D. Domínguez-García, "Robust distributed average consensus via exchange of running sums," *IEEE Trans. Autom. Control*, vol. 61, no. 6, pp. 1492–1507, 2016. doi: 10.1109/TAC.2015.2471695.

[29] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 3, pp. 1245–1260, 2018. doi: 10.1109/TCNS.2017.2698261.

[30] A. Nedic´, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM J. Optim.*, vol. 27, no. 4, pp. 2597–2633, 2017. doi: 10.1137/16M1084316.

[31] P. Di Lorenzo and G. Scutari, "Next: In-network nonconvex optimization," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 2, no. 2, pp. 120–136, 2016. doi: 10.1109/TSIPN.2016.2524588.

[32] S. Pu, W. Shi, J. Xu, and A. Nedic´, Push-pull gradient methods for distributed optimization in networks. 2018. [Online]. Available: arXiv:1810.06653

[33] R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *IEEE Control Syst. Lett.*, vol. 2, no. 3, pp. 315–320, 2018. doi: 10.1109/LCSYS.2018.2834316.

[34] M. G. Rabbat and R. D. Nowak, "Quantized incremental algorithms for distributed optimization," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 4, pp. 798–808, 2005. doi: 10.1109/JSAC.2005.843546.

[35] A. Nedic´, A. Olshevsky, A. Ozdaglar, and J. N. Tsitsiklis, "Distributed subgradient methods and quantization effects," in *Proc. 47th IEEE Conf. Decision and Control*, 2008, pp. 4177–4184. doi: 10.1109/CDC.2008.4738860.

[36] A. Kashyap, T. Bas¸ar, and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, no. 7, pp. 1192–1203, 2007. doi: 10.1016/j.automatica.2007.01.002.

[37] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proc. Int. Conf. Machine Learning*, 2019, pp. 3478–3487.

[38] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 1989.

[39] A. Nedic´ and A. Ozdaglar, "Convergence rate for consensus with delays," *J. Global Optim.*, vol. 47, no. 3, pp. 437–456, 2010. doi: 10.1007/s10898-008-9370-2.

[40] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, MATCHA: Speeding up decentralized SGD via matching decomposition sampling. 2019. [Online]. Available: arXiv:1905.09435

SP