PROCEEDINGS OF THE IEEE

Distributed Principal Component Analysis

Journal:	*Proceedings of the IEEE*
Manuscript ID	Draft
Manuscript Categories:	Special Issue Paper
Date Submitted by the Author:	n/a
Complete List of Authors:	WU, Xiaoxiao; Shenzhen University, Wai, Hoi-To; Arizona State University, Li, Lin; Lincoln Laboratory, Massachusetts Institute of Technology Human Language Technology Group Scaglione, Anna; Arizona State University
Keyword:	

SCHOLARONE™
Manuscripts
COPY

Distributed Principal Component Analysis

Sissi Xiaoxiao Wu, Hoi-To Wai Lin Li and Anna Scaglione

***Abstract*—Principal Component Analysis (PCA) is a fundamental primitive of many data analysis, array processing and machine learning methods. In applications where extremely large arrays of data are involved, particularly in distributed data acquisition systems, distributed PCA algorithms can harness local communications and network connectivity to overcome the need of communicating and accessing the entire array locally. A key feature of distributed PCA algorithm is that they defy the conventional notion that the first step towards computing the principal vectors is to form a sample covariance. This paper is a survey of the methodologies to perform distributed PCA on different datasets, their performance and of their applications in the context of distributed data acquisition systems.**

I. Overview

Distributed algorithms have a long history. In recent years, they have gained prominence in light of the end of Moore's law scaling and the seemingly exponential growth in data to analyze, due to the latest incarnation of networked technologies from social mobile media to the Internet of Things. This backdrop has sparked significant advances over the last decades on multi-agent signal processing algorithms. In contrast to their centralized counterparts, these algorithms requires the participating agents, *i.e.,* nodes in the network, to make use of their local processing power and the ability to communicate with each other by message passing, with the goal of tackling a common optimization problem.

In this context, a pervasive primitive in machine learning and sensor array processing is the computation of the *principal components* from the covariance matrix of several data streams. The goal of this paper is to survey *distributed algorithms* in Principal Component Analysis (PCA), which has wide applications in areas of communications, network, data mining and machine learning [1]–[3]. Generally speaking, PCA is a statistical procedure that convert a set of high dimensional samples into a set of features that represent the data in a lower dimensional space spanned by the principal components. Since its first appearance in the seminal 1901 paper by Karl Pearson [4], PCA has evolved into various forms that fit different applications. The distributed PCA algorithms we review in this paper leave a residual relative to the original data set that is minimum in the least-square sense, *i.e.,* with the minimum Euclidean norm. This is the most common version of PCA employed in signal processing and data science, and it is associated with a number of tools in linear algebra such as the Karhunen-Loéve transform (KLT), the singular value decomposition (SVD), the eigenvalue decomposition (EVD), the Orthogonal-triangular (QR) factorization, etc.

S. X. Wu is with the Department of Communication and Information Engineering, Shenzhen University, China. H.-T. Wai and A. Scaglione are with the Ira A. Fulton School of Electrical Computer and Energy Engineering in Arizona State University, United States. Lin Li is with Lincoln Laboratory, Massachusetts Institute of Technology Human Language Technology Group, United States. E-mail: xxwu.eesissi@szu.edu.cn, {htwai, Anna.Scaglione}@asu.edu, lin.li@ll.mit.edu

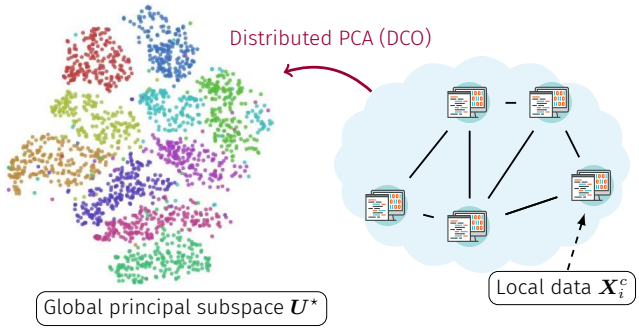

Fig. 1. **Example of Distributed PCA with DCO data**. In this illustration, each agent is a computer server gathering data locally from the users that it serves, organized through partitions by the columns.

The solution found by the PCA problem(s) is widely applied in dimensionality reduction and to cluster large amount of data into groups via spectral clustering [5], to classify word documents [6], for beam-forming in array processing [1], [2]. We refer the reader to [3] for a comprehensive survey of PCA applications.

The main goal of this review is to explain how distributed storage and computation systems can implement PCA. Next, we classify these algorithms based on the way the data are partitioned and the communications are structured in the network.

A. The family of distributed PCA algorithms

The distributed PCA algorithms in the literature can be broadly divided into different classes, and they are often designed according to: 1) how the data are divided in the network; 2) how the communication and computations among the different agents (or workers) are structured, namely a hierarchical architecture versus a totally flat architecture based on message passing.

1) Data partitioning: The design of distributed PCA algorithms depend heavily on the way that data are partitioned and stored in the agents on the network. Below, we survey two of the main classes of data partitions.

In the first class of data partition, each agent has access to a different subset of samples of the data set. This type of partition occurs in applications where a large amount of high dimensional data is stored across different sites in a network. In this case, distributed PCA allows to learn the important features from these high dimensional data that help compress, summarize, classify or rank them, without sharing the data directly. A relevant instance for this setting occurs in document

REVIEW COPY

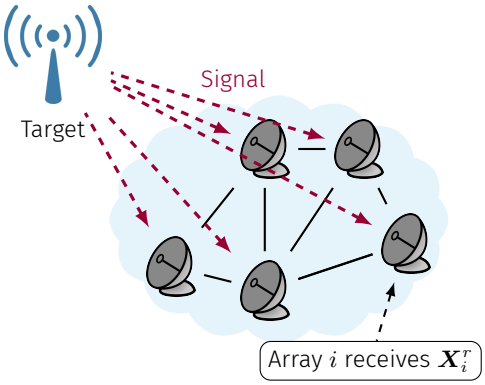

Fig. 2. **Example of Distributed PCA with DRO data.** In this example, each array receives its own copy of signal which is partitioned by rows and all the arrays estimate the subspace of received signals for target tracking.

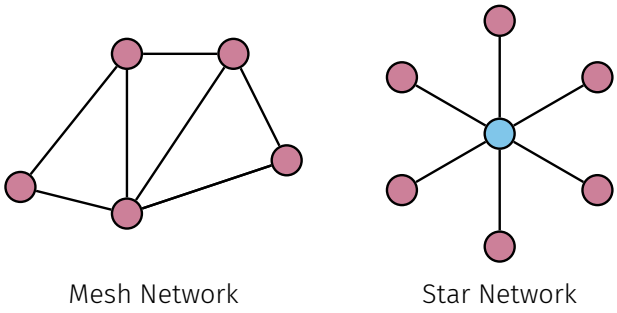

Fig. 3. **Mesh and Star Networks**. For star networks, the nodes are divided into slave nodes and a master node, where the latter receives data from all the slave nodes.

classification [cf. Fig. 1], where the data dimensions are the frequency of a specific word in the document, and each sample corresponds to a different document. As the documents are scattered across various servers, each agent in the network only possesses a subset of the outcomes. Applying distributed PCA in this case provides a way to de-noise the unlabeled data (that can be interpreted as topics) by retaining only the useful subspaces and that can then be used to classify the documents.

Prior work on this class of the distributed PCA algorithms is prevalent in the machine learning community, e.g., [7]–[17]. Specifically, [7], [14] proposed to compute a set of local principal component vectors at the agents and then have a central coordinator fuse these results into the desired order principal sub-space; the performance of this approach (and its variants) have been studied in e.g. [8]–[11]. In [12], a multi-round distributed PCA algorithm was proposed to better balance communication and computation costs. In [13], the principal components are found by performing a sequence of local QR decomposition at the agents. In [15], the principal components are computed using the aggregated eigenvectors and eigenvalues of the local covariance matrix. It is worthwhile to point out that except for [17], the work cited above considers a star network architecture, with a hub that fuses the results of several servers iteratively. This is different from the setting where the servers communicate through a meshed network, which is the primary focus of this paper review in Section IV.

The second class of data partition corresponds to having a multi-dimensional time series and having the entries of each sample distributed across the agents, on an entry by entry (or a block by block) basis. This situation naturally arises in distributed sensors deployments where the sensors collect samples simultaneously of a continuous field that evolves in time and space. For example, the field could the signal emitted or backscattered by a moving target [cf. Fig. 2]. What motivates PCA is the underlying assumption that the field has few active signal components, therefore spanning an unknown but low dimensional signal subspace. Each sensor collects one (or a subset) of the projections of such vector field at a given time, *i.e.,* a *spatial* sample of the field.

This class of PCA problems has inspired a number of distributed PCA methods developed by the signal processing community [18]–[27] which we review in Section V. For instance, [18]–[20] considered extending the classical power method [28] to the distributed setting. The proposed distributed power method was later analyzed in [21] and extended to an asynchronous setting in [22]. Note that the power method is a batch processing method with fast convergence, yet the method is non-adaptive and has high latency. Since the observations are evolving with time, it is natural to seek adaptive solutions; in the centralized setting this is achieved by the Oja's method [29], and the distributed setting can be found in [23], [25] which developed the decentralized subspace tracking algorithms. Interestingly, these class of algorithms are all for meshed networks.

Before we delve into the mathematical details in Section II, we shall fix some terminology by describing how the data samples are arranged. In particular, we define a data matrix which takes each sample of data as a column. In this way, the two main situations described above correspond, respectively, to what we shall refer as the Distributed Columns Observations (DCO) and the Distributed Rows Observations (DRO), respectively. In both cases, the PCA problem is aimed at retrieving the principal subspace for the *column span* of the data matrix, yet in the DRO case each node computes only a subset of the coordinates (one entry or one block) of the principal subspace vectors, while in the DCO case all nodes achieve consensus on the entire principal subspace basis. The way the knowledge about the principal subspace is shared in the DCO and DRO cases affects how the distributed PCA primitive can be used for different applications.

2) Communication and computation architecture: The design of distributed PCA algorithms also differ in terms of the type of communications they require: one variant uses a hierarchical master-slave type architecture and one uses a flat message passing architecture [cf. Fig. 3].

The master-slave approach for distributed PCA conforms to a hierarchical division of tasks, and employs a central coordinator that also acts as a fusion node [7]–[16]. In this case, the agents form a *star network* topology, where the computation tasks of the master node at the center are different from those of the servers. These models conform to the typical architecture for parallel computation in multi-core processors,

IEEE Proof

where the aim of applying distributed PCA algorithms is to accelerate PCA computation by utilizing local computations and also local memory resources. It is also worthwhile to point out that the algorithms above fall into the category of the DCO type distributed PCA.

The algorithms developed for this architecture usually consists of two stages — a local stage and a global stage. In the local stage, each agent/slave performs its own local optimization, e.g., by solving a local PCA problem, and send the results to the central coordinator. In the global stage, the central coordinator then computes the global PCA from the aggregated data.

The second class of distributed PCA algorithms works on arbitrarily meshed networks and conforms to the type of parallel processing that is performed in Graphic Processing Units (GPUs), or distributed storage systems. They have the following features: 1) all nodes and links must perform the same function and run the same procedures; 2) the nodes exchange partial computations but not the data, thus privacy is respected. This architecture can be seen as a generalization of the master-slave one, where the agents form a topology described by a general graph, and it arises from applications involving computer or wireless sensor networks. In particular, while the agents are still connected to each other, transmitting information from one agent to another may require multi-hop communications. In addition to accelerating PCA computations, this class of algorithms aims at offering better resiliencies in hostile environments such as random failures of agents. Examples of these algorithms can be found in [18]–[25]. To adapt to the mesh network setting, these algorithms are often developed by re-interpreting classical numerical methods for PCA such as power and Oja's method as a sequence of computation steps for averaging a set of local values across the agents. The distributed averaging can then be realized by resorting to instantiations of the *average consensus (AC)* subroutine; see [30], [31]. Their accuracy rests on the performance of the AC subroutine, which is known to converge exponentially at rate that depends on the algebraic connectivity of the communication network; see Section III-A.

We observe that the majority of distributed PCA algorithms belong to either one of the two types — (i) DCO data on master-slave architectures and (ii) DRO data on arbitrary mesh networks. Such dichotomy is due to the different types of applications that are prevalent in the machine learning and signal processing communities respectively. Throughout this paper, we focus on surveying algorithms for the settings with *arbitrarily meshed networks*, though some of the representative work for the master-slave architecture will also be summarized in Section IV-A. Furthermore, we shall emphasize that the tailor-made master-slave distributed PCA algorithms can be more effective in handling specific *big-data* problems, and they are preferable in contexts where one can choose to build such an architecture.

Notation. We use boldfaced lower-case letters (e.g. $\boldsymbol{x}$) to denote vectors and boldfaced upper-case letters (e.g. $\boldsymbol{X}$) to denote matrices. While this convention is prevalent throughout, occasionally roman capitals $\mathbf{X}$ will be used to denote the random vectors whose outcomes are denoted with the lower-case boldface notation $\boldsymbol{x}$ used for deterministic vectors. For a square matrix $\boldsymbol{R}$, $\lambda_i(\boldsymbol{R})$ denotes its ith largest eigenvalue; while for a rectangular matrix $\boldsymbol{X}$, $\sigma_i(\boldsymbol{X})$ is its ith largest singular value. The operator $(\cdot)^{\mathsf{H}}$ (*resp.* $(\cdot)^{\top}$) denotes the Hermitian transpose (*resp.* the standard transpose). Unless otherwise specified, $\|\boldsymbol{x}\|$ is the standard Euclidean norm for the vector $\boldsymbol{x}$.

B. Paper organization

In the next section we formulate mathematically the distributed PCA problem and its two instantiations, for the DRO and DCO scenarios, respectively. Then, in Section III, we provide background on the building blocks that are at the basis of the distributed PCA algorithms: specifically, the AC subroutine, the power method and the Oja's method. Section IV and V review the distributed PCA methods for the DRO setting and for the DCO setting, respectively; within each of them, some salient applications are also reviewed. Possible extensions of the work cited and the related problems are presented in Section VI, which is followed by the conclusions in Section VII.

II. The PCA Problem & Its Distributed Forms

We consider a set of observations given as an $N \times T$ (potentially complex) matrix $\boldsymbol{X}$, *i.e.,*

$$\boldsymbol{X} := \big(\boldsymbol{x}(1)\ \boldsymbol{x}(2)\ \cdots\ \boldsymbol{x}(T)\big) \in \mathbb{C}^{N\times T}\ , \tag{1}$$

The rows and columns of $\boldsymbol{X}$ represent the feature/spatial dimension and the sample/time dimension, respectively. The singular value decomposition (SVD) of $\boldsymbol{X}$ is denoted by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{H}}$ where $\boldsymbol{U}, \boldsymbol{V}$ are unitary matrices and $\boldsymbol{D} = \mathrm{Diag}([\sigma_1; \ldots; \sigma_R])$ is a diagonal matrix of the singular values with $R = \mathrm{rank}(\boldsymbol{X})$. We make the canonical assumption that all the singular values have multiplicity of one, *i.e.,* $\sigma_1 > \ldots > \sigma_R > 0$[1].

In many practical scenarios, the data samples $\boldsymbol{x}(t)$ are correlated with each other. When the dataset is large with $N, T \gg 1$ an option to reduce the size of the data is to project $\boldsymbol{X}$ onto its p-dimensional (p-D) principal subspace, which can be represented by an orthogonal transformation $\boldsymbol{U}_p$ — a sub-matrix of the unitary matrix $\boldsymbol{U}$ consisting of only its left p column vectors. In particular, given the orthogonal projection vector $\boldsymbol{z}(t) = \boldsymbol{U}_p^{\mathsf{H}}\boldsymbol{x}(t) \in \mathbb{C}^p$ the corresponding low-dimensional approximation of $\boldsymbol{x}(t)$ is given by $\hat{\boldsymbol{x}}(t) = \boldsymbol{U}_p\boldsymbol{z}(t)$. It is also convenient to write the tth sample as:

$$\boldsymbol{x}(t) = \hat{\boldsymbol{x}}(t) + \boldsymbol{e}(t) = \boldsymbol{U}_p\boldsymbol{z}(t) + \boldsymbol{e}(t)\ , \tag{2}$$

where $\boldsymbol{e}(t)$ represents the modeling error. PCA is effective when the Frobenious norm of $\boldsymbol{e}(t)$ is small. Based on (2), the PCA problem amounts to learning the orthogonal transformation $\boldsymbol{U}_p$ from the data $\boldsymbol{X}$ that would allow a *lossy* mapping of $\boldsymbol{x}(t) \mapsto \boldsymbol{z}(t)$. Under the standard assumption that

[1]This assumption can be relaxed, e.g., when one is interested only finding the top p principal subspaces, then only $\sigma_1 \geq \cdots \geq \sigma_p > \sigma_{p+1} \geq \cdots \geq \sigma_R \geq 0$ is needed for the algorithms in this paper.

PREVIEW COPY

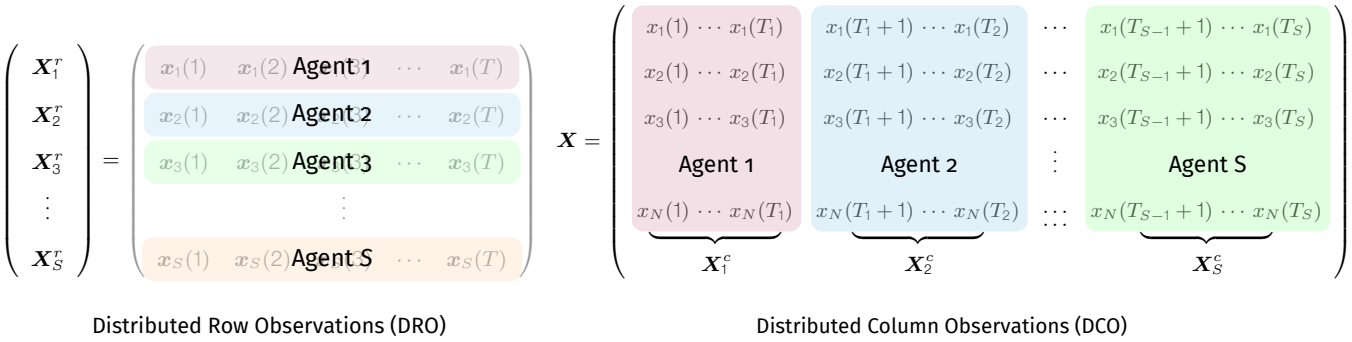

Fig. 4. **Types of Data Partition employed in Distributed PCA.** Each data partition type shall involve a different solution technique for distributed computation of the PCA.

$T \geq p$, the PCA problem can be represented as the following optimization:

$$\begin{aligned} \boldsymbol{U}^\star \in \arg \min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \quad & \|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{X}\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I} \,. \end{aligned} \tag{3}$$

As seen, the solution of (3) minimizes the residual for the signal reconstructed from $\{\boldsymbol{z}(t)\}_{t=1}^T$ in the mean square sense. The problem may also be extended to a stochastic and dynamic setting as follows:

$$\begin{aligned} \boldsymbol{U}^\star(t) \in \arg \min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \quad & \mathbb{E}\big[\|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\mathbf{X}(t)\|^2\big] \\ \text{s.t.} \quad & \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I} \,, \end{aligned} \tag{4}$$

where we have denoted $\mathbf{X}(t) \in \mathbb{C}^N$ as a vector-valued random process and the expectation is taken with respect to the distribution of $\mathbf{X}(t)$. Note that the realization of $\mathbf{X}(t)$ is denoted as $\boldsymbol{x}(t)$. To account for the possible non-stationarity in $\mathbf{X}(t)$, the solution of (4) depends on the sample/time index t. We shall refer to (3) as the *batch/static* PCA problem, and (4) as the *dynamic/stochastic* PCA problem. Note that the solution to the PCA problems is intrinsically ambiguous subject to rotations, as we observe by first defining the equivalence class $[\boldsymbol{U}]$ with $\boldsymbol{U} \in \mathbb{C}^{N\times p}$:

$$[\boldsymbol{U}] := \{\hat{\boldsymbol{U}} \in \mathbb{C}^{N\times p} |\ \hat{\boldsymbol{U}} = \boldsymbol{U}\boldsymbol{Q},\ \boldsymbol{Q} \in \mathbb{C}^{p\times p} \text{ is unitary}\} \,. \tag{5}$$

Importantly, any matrix in $[\boldsymbol{U}^\star]$ (*resp.* $[\boldsymbol{U}^\star(t)]$) will also be an optimal solution to (3) (*resp.* (4)).

In the *batch* PCA setting, a *centralized* PCA algorithm [32] solves (3) by performing an SVD on $\boldsymbol{X}$, where we can simply set the optimal solution as $\boldsymbol{U}^\star = \boldsymbol{S}_p$. Note that this gives the optimal objective value of $\sum_{r=p+1}^R \sigma_r^2$. In a *distributed* or *dynamic* setting, the entries of $\boldsymbol{X}$, instead of being processed at a central machine, are scattered on S different machines/sensors, which we shall refer to as *agents* later on. We further provide the following definitions to distinguish the types of distributed algorithms:

- **Distributed Columns Observations (DCO)**: The DCO setting assumes that each agent observes a subset of *columns* of $\boldsymbol{X}$. We partition $\boldsymbol{X}$ by its columns such that

$$\boldsymbol{X} = (\boldsymbol{X}_1^c\ \boldsymbol{X}_2^c\ \cdots\ \boldsymbol{X}_S^c) \,, \tag{6}$$

  where $\boldsymbol{X}_i^c \in \mathbb{R}^{N\times T_i}$ is the column-partitioned sub-matrix kept by agent i and $T = \sum_{i=1}^S T_i$. Alternatively, the data available at agent i can also be represented as $\boldsymbol{x}(t) \in \mathbb{C}^N$ for $t \in \mathcal{T}_i$ with $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_S = \{1,...,T\}$, $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$, and $|\mathcal{T}_i| = T_i$.
- **Distributed Rows Observations (DRO)**: The DRO setting assumes that each agent observes only a subset of *rows* of $\boldsymbol{X}$. We partition $\boldsymbol{X}$ by its rows such that

$$\boldsymbol{X} = ((\boldsymbol{X}_1^r)^\top\ (\boldsymbol{X}_2^r)^\top\ \cdots\ (\boldsymbol{X}_S^r)^\top)^\top \,, \tag{7}$$

  where agent i keeps the row-partitioned sub-matrix $\boldsymbol{X}_i^r \in \mathbb{R}^{N_i\times T}$ with $N = \sum_{i=1}^S N_i$. Alternatively, the data available at agent i can also be represented as $\boldsymbol{x}_i(t) \in \mathbb{C}^{N_i}$ for $t = 1,...,T$.

See Fig. 4 for an illustration on the types of data structure considered. A key feature that distinguishes distributed PCA methods from its centralized counterpart is that the agents have to solve (3) by *cooperating* with their neighbors in the network, given that each agent has only access to the partial observation matrix. Based on the settings we just discussed, next we describe the goals of the distributed PCA methods.

The *DCO* setting is encountered primarily in *big-data* mining applications. Each agent in this case can be a computer server that gathers data samples from a set of users that it is serving. For instance, the ith agent obtains samples $\{\boldsymbol{x}(t)\}_{t\in\mathcal{T}_i}$ from a group of users. In this case, our goal is:

> *Goal (DCO):* Agent i learns a common p-D principal subspace of $\boldsymbol{X}$ shared by the other agents, *i.e.,* to learn $\boldsymbol{U}^\star$, in a distributed fashion.

Notice that the goal here is similar to solving a *consensus* problem requiring the agents to agree with each other. Using the data structure in the DCO setting, we observe that (3) can be written as:

$$\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \ \sum_{i=1}^S \underbrace{\|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{X}_i^c\|_{\mathrm{F}}^2}_{:=f_i(\boldsymbol{U})} \ \text{ s.t. } \ \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I} \,, \tag{8}$$

which has a separable objective function similar to the consensus optimization problem tackled in [33]. In other words, our aim is to find a *common* dictionary based on all the data accrued across the network. Furthermore, we remark that the DCO data structure is only relevant in the static/batch PCA setting.

REVIEW COPY

As mentioned before, the *DRO* setting is typically encountered in sensor networks where each agent, depending on its siting, captures components of the vector field that corresponds to the block $\boldsymbol{x}_i(t)$. For instance, the ith agent may have access to the ith antenna that received the signal $x_i(t)$. In this case, we define our goal as:

> *Goal (DRO):* Agent i learns the ith partition of the p-D principal subspace of $\boldsymbol{X}$, *i.e.,* to learn $\boldsymbol{U}_i^{r,\star}$ in the partition $\boldsymbol{U}^\star = (\boldsymbol{U}_1^{r,\star}; \ldots; \boldsymbol{U}_S^{r,\star})$, in a distributed fashion.

This is a reasonable setting as $\boldsymbol{U}_i^{r,\star}$ keeps the components of $\boldsymbol{U}^\star$ that are related to the observations made at the ith agent. A typical example for DRO is the subspace estimation and tracking in a sensor or radar network. The main setting is an information fusion, which consists of many nodes, that could be sensor nodes, arrays, radars. Each node can be seen as a one-dimension unit for receiving the signal, aiming at cooperatively estimating and tracking the principal subspace of the signals. Once the nodes have retrieved the PCA, the distributed projection on the principal subspace can also be distributed and all nodes will achieve a consensus on the coordinates of the PCA approximation $\boldsymbol{z}(t)$.

The important observation here is that in the DRO setting the PCA problem in not separable like it is for the DCO setting, as seen in (8), and there is no consensus condition that ties the results the agents obtain.

Prior to reviewing the DCO and DRO algorithms in Sections IV and V respectively, in the next section we briefly introduce some of the building blocks these algorithms rely on.

III. Building Blocks

In this section, we describe the important algorithms for network consensus and PCA, which will then serve as the building blocks for the distributed PCA methods surveyed.

A. Average Consensus Algorithm (a.k.a. Gossip Algorithm)

In this section we provide a brief introduction to the so called *average consensus algorithm* (also known as the gossip algorithm) [30], [31], which is key computation method used in distributed PCA methods. As its name has suggested, the average consensus algorithm describes a procedure for computing the *average* of a set of values stored at the agents through *local information exchanges*.

To proceed, it is necessary to define some terminology to refer to the communication network that connects the agents. As explained in the Introduction, we focus on the setting where the agents are connected on an arbitrarily *meshed network*. Specifically, the network used can be mapped onto a connected, undirected and simple graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., S\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of communication links between the agents. See Fig. 5 for examples of a few common models to generate graph topologies.

Suppose that the ith agent holds a certain vector/matrix $\boldsymbol{Y}_i$ and he/she wishes to compute the global *average* $\boldsymbol{Y}_{\text{avg}} := S^{-1}\sum_{i=1}^S \boldsymbol{Y}_i$. A distributed algorithm suitable for the task is the classical average consensus algorithm [30], [31], [34], also known as the gossip algorithm. In particular, for all $\ell \geq 1$, let $\boldsymbol{W}[\ell]$ be a symmetric and doubly stochastic matrix (i.e. such that $\boldsymbol{W}[\ell]\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T\boldsymbol{W}[\ell] = \mathbf{1}^T$) whose sparsity matches that of adjacency matrix for the graph G, that is $W_{ij}[\ell] = W_{ji}[\ell] > 0$ only if $(i,j) \in \mathcal{E}$, and $\boldsymbol{W}[\ell]\mathbf{1} = \boldsymbol{W}^\top[\ell]\mathbf{1} = \mathbf{1}$. We can compute $\boldsymbol{Y}_{\text{avg}}$ distributively as follows:

> **Average Consensus (AC) algorithm**:
> 1) Initialize as $\boldsymbol{Z}_i[0] = \boldsymbol{Y}_i$ for all i.
> 2) For all agent $i \in \{1, ..., S\}$, perform the recursion:
>
> $$\boldsymbol{Z}_i[\ell] = \sum_{j=1}^S W_{ij}[\ell]\boldsymbol{Z}[\ell-1]\ , \tag{9}$$
>
> for all $\ell \geq 1$ and we terminate after $\ell \geq L$. The collection $\{\boldsymbol{Z}_i[L]\}_{i=1}^S$ is retrieved as the output of the AC subroutine.

For simplicity, let us denote $\boldsymbol{Z}_i[L]$, *i.e.,* the variable after the Lth update, as output of the above subroutine:

$$\{\boldsymbol{Z}_i[L]\}_{i=1}^S := \mathsf{AC}\big(\{\boldsymbol{Y}_i\}_{i=1}^S; L\big)\ , \tag{10}$$

where the first argument to the subroutine are the initialization given to the network and the second argument specifies the number of average consensus updates required. We also use $\boldsymbol{Z}_i[L] := \mathsf{AC}_i\big(\{\boldsymbol{Y}_i\}_{i=1}^S; L\big)$ to denote the output of the AC subroutine stored at the ith agent as indicated by the subscript. We remark that the AC subroutine defined in the above is applicable for computing the averages of scalar, vector and matrices.

Above, we described the most general case where $\boldsymbol{W}[\ell]$ changes over time, e.g., it models the scenario when some links in the network may be inactive at times. Formally, for every $\ell \geq 1$, this matrix is drawn from a distribution with $\mathbb{E}[\boldsymbol{W}[\ell]] = \overline{\boldsymbol{W}}$ with

$$\lambda_{\text{conn.}} := \max\{\lambda_2(\overline{\boldsymbol{W}}), -\lambda_S(\overline{\boldsymbol{W}})\} < 1\ . \tag{11}$$

An example is represented by the *pairwise gossiping protocol* introduced by [34], where an edge (i_ℓ, j_ℓ) is selected from $\mathcal{E}$ uniformly at random and the agents compute a convex combination of their current state values which then becomes their new state. In this case:

$$\boldsymbol{W}[\ell] = \boldsymbol{I} - \frac{1}{2}(\boldsymbol{e}_{i_\ell} - \boldsymbol{e}_{j_\ell})(\boldsymbol{e}_{i_\ell} - \boldsymbol{e}_{j_\ell})^\top \tag{12}$$

and the condition (11) will be satisfied as long as G is a connected graph.

An important feature of the average consensus algorithm is that its convergence is exponentially fast, *i.e.,* we have $\mathbb{E}[\|\boldsymbol{Z}_i[L] - \boldsymbol{Y}_{\text{avg}}\|] = \mathcal{O}(\lambda_{\text{conn.}}^L)$ for all i where the expectation is taken with respect to the realizations of $\boldsymbol{W}[\ell]$. To compute an ϵ-accurate average, one only requires $\Theta(\log \epsilon^{-1})$ average consensus updates.

B. Power Method for PCA

Motivated by the fact that an optimal solution to the *static* PCA problem (3) can be obtained from evaluating the top (left) singular vectors of the data matrix $\boldsymbol{X}$ (or equivalently the top eigenvectors of the auto-correlation $\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}}$), a natural idea

IEEE REVIEW COPY

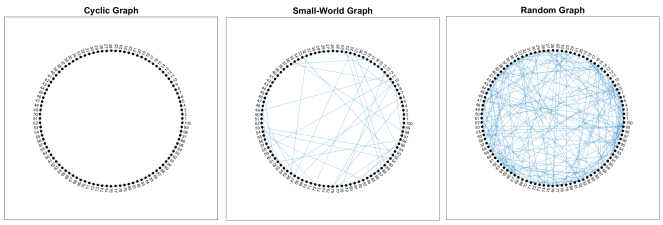

Fig. 5. **Examples of graph topology.** Left: a cyclic graph with $S = 100$ nodes and node degree 4; Middle: a small-world graph with $S = 100$, average degree 4 and re-wiring probability 0.2. Right: a random graph with $S = 100$ nodes and node degree 4.

is to apply the power method for solving the PCA problem, which is a well known numerical method [28] for computing the top eigenvectors of a symmetric matrix. In light of this, here we briefly review the power method and provide insights that will be instrumental to the development of distributed PCA methods. Consider the PCA problem (3) and observe that an optimal solution can be found by retrieving the top-p eigenvectors corresponding to the largest p eigenvalues of the sampled correlation matrix:

$$\hat{\boldsymbol{R}}_x := \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\ . \tag{13}$$

Note that $\boldsymbol{x}(t)$ is zero mean, then $\hat{\boldsymbol{R}}_x$ is also the sampled covariance. Alternatively, one could remove the mean in the data by a simple pre-processing step. To compute the *top* eigenvector of $\hat{\boldsymbol{R}}_x$, the power method is initialized by $\boldsymbol{u}_1[1] \sim \mathcal{CN}(0, \boldsymbol{I})$, and it adopts the following recursion:

$$\overline{\boldsymbol{u}}_1[k] = \frac{\boldsymbol{u}_1[k]}{\|\boldsymbol{u}_1[k]\|},\ \boldsymbol{u}_1[k+1] = \hat{\boldsymbol{R}}_x\overline{\boldsymbol{u}}_1[k],\ \forall\ k \geq 1\ . \tag{14}$$

It can be shown that $\overline{\boldsymbol{u}}_1[k]$ converges to $\boldsymbol{u}_1^\star$ as $k \to \infty$, where $\boldsymbol{u}_1^\star$ is the *top* eigenvector of $\hat{\boldsymbol{R}}_x$.

From an optimization perspective, the power method can be seen as a *fixed-point iteration* method which solves the non-linear system arising from the optimality condition of (3). Its rate of convergence is exponential: it computes an ϵ-accurate eigenvector with $k = \Omega((\log(\sigma_1(\hat{\boldsymbol{R}}_x)/\sigma_2(\hat{\boldsymbol{R}}_x)))^{-1}\log(1/\epsilon))$ power iterations [28][2]. Note that the ratio $\sigma_1(\hat{\boldsymbol{R}}_x)/\sigma_2(\hat{\boldsymbol{R}}_x)$ is known as the *spectral gap* of $\hat{\boldsymbol{R}}_x$ and it is a key factor determining the convergence speed of the power method. Moreover, as we shall reveal later, the computations above can be performed distributively with the help of the average consensus subroutine, by exploiting the relationship of $\hat{\boldsymbol{R}}_x$ with the data.

To find the *second* eigenvector of $\hat{\boldsymbol{R}}_x$, denoted by $\boldsymbol{u}_2^\star$, we observe that $(\boldsymbol{I} - \boldsymbol{u}_1^\star(\boldsymbol{u}_1^\star)^{\mathsf{H}})\hat{\boldsymbol{R}}_x$ is also Hermitian and positive semidefinite, and this matrix's top eigenvector is the sought $\boldsymbol{u}_2^\star$. Naturally, we can apply the same power method procedure to compute $\boldsymbol{u}_2^\star$. Repeating the same procedures we can find $\boldsymbol{u}_3^\star, ..., \boldsymbol{u}_p^\star$, to complete the principal subspace $\boldsymbol{U}^\star = (\boldsymbol{u}_1^\star, ..., \boldsymbol{u}_p^\star)$.

[2]In the sense that $\sqrt{1 - |\overline{\boldsymbol{u}}_1[k]^{\mathsf{H}}\boldsymbol{u}_1^\star|^2} \leq \epsilon$.

C. Oja-based Methods for PCA

Originally proposed by Oja *et al.* [29] in 1985, the Oja's method for PCA was developed from a different philosophy than the power method. In particular, the method focuses on tackling the optimization problem (4) using a stochastic gradient descent (SGD) method. Notice that (4) is a non-convex and stochastic optimization problem which can be difficult to handle. As a remedy, we consider the special case of $p = 1$ and the following form of (3):

$$\boldsymbol{u}^\star(t) \in \arg\max_{\boldsymbol{u}\in\mathbb{C}^N}\ f_t(\boldsymbol{u}) := \frac{\boldsymbol{u}^{\mathsf{H}}\mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)]\boldsymbol{u}}{\|\boldsymbol{u}\|^2}\ , \tag{15}$$

where the objective function is also known as the Rayleigh coefficient of the correlation matrix $\boldsymbol{R}_x(t) := \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)]$. The following function is a stochastic approximation of $f_t(\boldsymbol{u})$:

$$\hat{f}(\boldsymbol{u}; \{t\}) := \frac{\boldsymbol{u}^{\mathsf{H}}\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}}{\|\boldsymbol{u}\|^2} \approx f_t(\boldsymbol{u})\ , \tag{16}$$

since only one sample is used in the above, we say that the batch size used is 1. We observe the approximation is unbiased as $\mathbb{E}[\hat{f}(\boldsymbol{u}; \{t\})] = f_t(\boldsymbol{u})$. Consequently, its gradient is also unbiased since $\mathbb{E}[\nabla\hat{f}(\boldsymbol{u}; \{t\})] = \nabla f_t(\boldsymbol{u})$.

The Oja's method in [29] is essentially an SGD method for (15) with a batch size of 1, *i.e.,* let $\boldsymbol{u}^{\mathsf{Oja}}(t) \in \mathbb{C}^N$ be the estimated principal component at iteration t, we have

$$\begin{aligned}\boldsymbol{u}^{\mathsf{Oja}}(t+1) &= \boldsymbol{u}^{\mathsf{Oja}}(t) + \tilde{\gamma}_t\nabla\hat{f}(\boldsymbol{u}^{\mathsf{Oja}}(t); \{t\}) \\ &= \boldsymbol{u}^{\mathsf{Oja}}(t) + \gamma_t\Big(\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t) - \frac{|\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)|^2}{\|\boldsymbol{u}^{\mathsf{Oja}}(t)\|^2}\boldsymbol{I}\Big)\boldsymbol{u}^{\mathsf{Oja}}(t)\ ,\end{aligned} \tag{17}$$

where $\gamma_t > 0$ is a step size and $\gamma_t := 2\tilde{\gamma}_t/\|\boldsymbol{u}^{\mathsf{Oja}}(t)\|^2$. It is worth noting that, due to the non-convex nature of (15), the global convergence of Oja's learning rule has remained elusive. In fact, most of the available results are focused on the special cases with stationary $\boldsymbol{x}(t)$. For example, the authors in [35] proved that when $p = 1$ and $\gamma_t = c/t$, then (17) returns the top eigenvector of $\boldsymbol{R}_x$ as $t \to \infty$ at a sub-linear rate of $\mathcal{O}(1/t)$; [29, Theorem 2] proved if $\sum_t \gamma_t = \infty$, $\sum_t \gamma_t^2 < \infty$, then (17) converges almost surely to the principal subspace, yet the convergence rate is not given. Nevertheless, the Oja's learning rule is often used even when the process $\boldsymbol{x}(t)$ is non-stationary. To avoid getting stuck at a solution equal to the previous subspaces, an effective heuristic is to set γ_t to be a small *constant* such that the newly observed samples are sufficiently represented.

Various forms of Oja-based method have also been proposed. In general, their philosophy is to apply different relaxations to the PCA problem (4) and then to apply SGD on the relaxed problems. Examples are NOja and NOOja studied in [36]. Their convergence has been studied in [37].

IV. Distributed PCA Methods for DCO

In this section we survey some of the representative algorithms for distributed PCA in the DCO setting. As mentioned before, a distinguishing feature of the DCO scenario is that each agent keeps a (sub)set of the observed samples $\{\boldsymbol{x}(t)\}_{t\in\mathcal{T}_i}$, grouped into an $N \times T_i$ matrix $\boldsymbol{X}_i^c$. When $\boldsymbol{x}(t)$ are

REVIEW COPY

generated i.i.d. and T_i is large, each agent possesses sufficient data to compute the PCA locally. As such, we focus on the more restrictive case when T_i is not large and discuss the strategies in which the agents can leverage on additional information from other agents in the same network.

A. Strategies for Master-slave Architectures

A number of papers [7]–[17] tackled the distributed PCA problem (with DCO data) when the agents are organized according to a *master-slave* architecture, or equivalently a star network. As mentioned in the Introduction, these works keep the sets of data samples at the different servers sites and focus on the issue of reducing the *overall* computation complexity and communication cost required by the distributed methods.

As an example, Qu *et al.* [7] proposed to compute a *local PCA* from the partial data by having agent i perform an SVD of its own local subset of data matrix, deprived of their mean, *i.e.,* :

$$\boldsymbol{X}_i^c(\boldsymbol{I} - |\mathcal{T}_i|^{-1}\mathbf{1}\mathbf{1}^\top) = \boldsymbol{U}_i\boldsymbol{\Lambda}_i\boldsymbol{V}_i^{\mathsf{H}} . \tag{18}$$

Let $\tilde{\boldsymbol{U}}_i \in \mathbb{C}^{N\times p_i}$ be the matrix for the top p_i singular vector and $\tilde{\boldsymbol{\Lambda}}_i$ be the principal $p_i \times p_i$ submatrix of $\boldsymbol{\Lambda}$, where $p_i \geq p$. Agent i then transmits $\tilde{\boldsymbol{U}}_i\tilde{\boldsymbol{\Lambda}}_i^2\tilde{\boldsymbol{U}}_i^{\mathsf{H}}$ and the local mean $\bar{\boldsymbol{x}}_i := |\mathcal{T}_i|^{-1}\boldsymbol{X}_i^c\mathbf{1}\mathbf{1}^\top$ to the central server, which estimates the global PCA, $\hat{\boldsymbol{U}} \in \mathbb{C}^{N\times p}$, from:

$$\tilde{\boldsymbol{S}} = \sum_{i=1}^{S}\tilde{\boldsymbol{U}}_i\tilde{\boldsymbol{\Lambda}}_i^2\tilde{\boldsymbol{U}}_i^{\mathsf{H}} + \sum_{i=1}^{S}|\mathcal{T}_i|(\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}})(\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}})^{\mathsf{H}} . \tag{19}$$

We observe that this algorithm can be completed in one communication and computation round and its accuracy depends on the choice of p_i. On the other hand, it is clear that $\hat{\boldsymbol{U}} \neq \boldsymbol{U}^\star$ as $\tilde{\boldsymbol{S}}$ does not share the principal subspaces as $\boldsymbol{S}$ in general. In fact, the accuracy of the algorithm above can only guaranteed when the data observed are *homogeneous*, e.g., for all i, each column of $\boldsymbol{X}_i^c$ is generated from (2) with independently distributed modeling error $\boldsymbol{e}(t)$. The merits of these approaches rest on the savings in computation cost when implemented in a parallel computer system. Similar approach can be found in [8] which combines distributed PCA method with distributed K-means algorithms for further processing of the dimension reduced data.

B. Strategies for Mesh Networks

From now on we focus on the mesh networks case. As pointed out in [17], a naïve distributed PCA method can be obtained by simply approximating the global correlation matrix via the AC subroutine, *i.e.,* for some $L \geq 1$, we have

$$\hat{\boldsymbol{R}}_{x,i} = \frac{S}{T}\cdot\mathsf{AC}_i\big(\{\boldsymbol{X}_j^c(\boldsymbol{X}_j^c)^{\mathsf{H}}\}_{j=1}^S; L\big) \approx \hat{\boldsymbol{R}}_x . \tag{20}$$

In other words, each agent obtains an approximate of the global correlation matrix and the desired PCA can be then computed from $\hat{\boldsymbol{R}}_{x,i}$ using, for example, the power method. The drawback, however, lies on the communication and computation cost entailed in this approach, which is particularly onerous when considering high-dimensional data with $N \gg 1$. This is because (20) requires computing the average of an $N \times N$ matrix.

As an improvement over the correlation averaging method, next we describe a natural extension of the power method in (14) to the distributed setting with DCO data. In this setting, the distributed power method (DistPM) was introduced as a subroutine in [38], [39]. Observe that the first expression in (14) can be written as:

$$\begin{aligned}\boldsymbol{u}_1[k] &= \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}}\overline{\boldsymbol{u}}_1[k-1] \\ &= \frac{1}{T}\sum_{i=1}^{S}\Big(\boldsymbol{X}_i^c(\boldsymbol{X}_i^c)^{\mathsf{H}}\overline{\boldsymbol{u}}_1[k-1]\Big) ,\end{aligned} \tag{21}$$

If a copy of $\overline{\boldsymbol{u}}_1[k-1]$ is also known to the agents, computing each of the terms requires data known to the ith agent and, therefore, $\boldsymbol{u}_1[k]$ can simply be computed adding up the S terms — a computation that can be accomplished using the average consensus subroutine. This would be the case if the nodes start with the same initial vector $\boldsymbol{u}_1[0]$ and achieve asymptotic convergence to the average. In reality both conditions can be relaxed.

Specifically, for the DistPM with DCO data, we denote $\boldsymbol{u}_1^i[k] \in \mathbb{C}^N$ (*resp.* $\overline{\boldsymbol{u}}_1^i[k]$) as the estimate of the first (*resp.* normalized) eigenvector of $\boldsymbol{R}_x$ at the kth power iteration, kept by the ith agent. The DistPM proceeds by:

Distributed Power Method (DistPM) for DCO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_1^i[0] \sim \mathcal{CN}(\mathbf{0}, \boldsymbol{I})$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:
$$\boldsymbol{u}_1^i[k] = S\cdot\mathsf{AC}_i\big(\{\boldsymbol{X}_j^c(\boldsymbol{X}_j^c)^{\mathsf{H}}\overline{\boldsymbol{u}}_1^j[k-1]\}_{j=1}^S; L\big) ,$$
$$\overline{\boldsymbol{u}}_1^i[k] = \boldsymbol{u}_1^i[k]/\|\boldsymbol{u}_1^i[k]\| ,$$
for all $k \geq 1$ and we terminate after $k \geq K$.
3) Denote $\hat{\boldsymbol{u}}_1^i := \boldsymbol{u}_1^i[K]$ as the solution kept by the ith agent for all $i \in \{1, ..., S\}$.

The subsequent eigenvectors $\boldsymbol{u}_2^\star, ..., \boldsymbol{u}_p^\star$ can be found by using a similar strategy as in the centralized power method.

Theoretical Guarantee: The convergence of DistPM (DCO) has been analyzed in [39] for real data matrices and static network between the agents, *i.e.,* $\boldsymbol{W}[\ell] = \overline{\boldsymbol{W}}$ for all ℓ. Again, for simplicity, we only present the result for $p = 1$ where the top eigenvector of $\hat{\boldsymbol{R}}_x$ is sought. We have

Theorem IV.1 *Consider the DistPM (DCO) method for real data matrices and $p = 1$. Fix $1/2 > \epsilon > 0$ be the desirable accuracy, and $c > 0$ be the failure probability. If the DistPM parameters satisfy:*

$$\begin{aligned}L &= \Omega\left(\frac{\log\epsilon^{-1} + \log(\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x))^{-1}}{\log(\lambda_{\text{conn.}}^{-1})}\right) \\ K &= \Omega\left(\frac{\lambda_1(\hat{\boldsymbol{R}}_x)}{\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x)}\cdot\log\left(\frac{N}{c\cdot\epsilon}\right)\right),\end{aligned} \tag{22}$$

and the initialization satisfies $(\boldsymbol{u}_1^\star)^\top\boldsymbol{u}_1^i[0] \neq 0$, then with probability at least $1 - Sc$, we have:

$$((\boldsymbol{u}_1^\star)^\top\hat{\boldsymbol{u}}_1^i)^2 \geq 1 - \epsilon^2, \quad \min_{j=2,...,N}((\boldsymbol{u}_j^\star)^\top\hat{\boldsymbol{u}}_1^i)^2 \leq \epsilon^2 , \tag{23}$$

REVIEW COPY

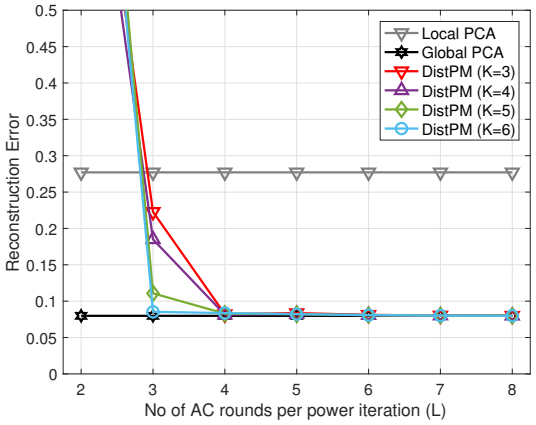

Fig. 6. **Applying DistPM (DCO) on the dimensionality reduction problem**. The reconstruction error is shown for agent $i = 1$ on the testing data, which is generated according to a similar model as (25). In the legend, 'local PCA' (*resp.* 'global PCA') refers to using only the local data (*resp.* all available data) to estimate the principal subspace.

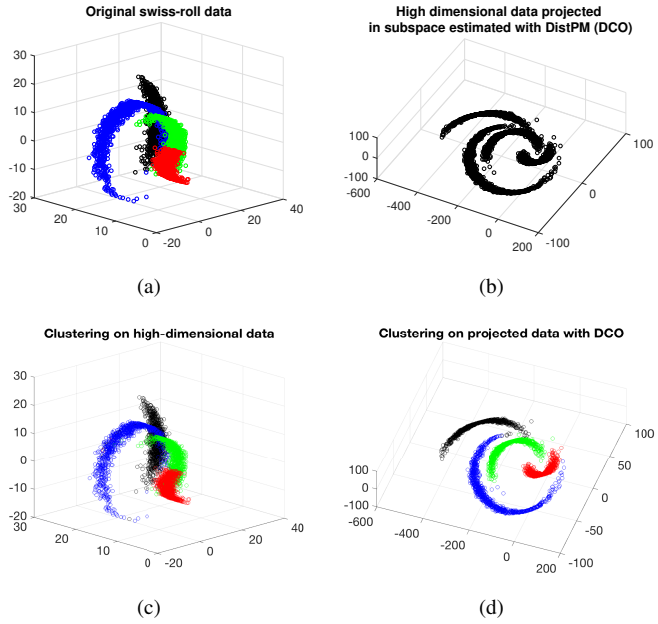

Fig. 7. **Clustering on the swiss roll data** (a) Original swiss roll data before high-dimensional embedding; (b) High-dimensional data projected in 3-dimensional subspace estimated with DistPM (DCO); (c) Results of hierarchical clustering on the original high-dimensional data; and, (d) Results of hierarchical clustering on the projected data with DistPM (DCO).

for all $i \in \{1, ..., S\}$. *Also, the eigenvectors found are in consensus:*

$$\|\hat{\boldsymbol{u}}_1^i(\hat{\boldsymbol{u}}_1^i)^\top - \hat{\boldsymbol{u}}_1^{i'}(\hat{\boldsymbol{u}}_1^{i'})^\top\| = \mathcal{O}(\epsilon),\ \forall\ i, i' \in \{1, ..., S\}\ . \quad (24)$$

Here, the randomness in the algorithm is due to the random initializations. The theorem above is taken from [39, Proposition 1] and was partially inspired by [40]. Theorem IV.1 gives a *non-asymptotic* bound on the accuracy achieved by the DistPM (DCO) as it does not require $P \to \infty$.

C. Application Examples

We consider applying PCA to dimensionality reduction and clustering in a distributed setting. Particularly, we assume that the ith machine/agent observes a dataset which can be described as:

$$\boldsymbol{X}_i^c = \boldsymbol{A}\boldsymbol{Z}_i^c + \boldsymbol{E}_i \in \mathbb{R}^{N \times T_i}\ , \quad (25)$$

where each row of $\boldsymbol{X}_i^c$ represents a *feature* and each column is a data point. In the above, $\boldsymbol{A} \in \mathbb{R}^{N \times p}$ is a common 'dictionary' with $p \ll N$, $\boldsymbol{Z}_i^c \in \mathbb{R}^{p \times T_i}$ is the latent parameters for the observations and $\boldsymbol{E}_i$ represents the modeling error. Furthermore, $\boldsymbol{X}_i^c$ constitutes a set of *training data*, and our aim is to learn the subspace spanned by the columns of $\boldsymbol{A}$ so that we can compress some unseen data, $\boldsymbol{X}_{\text{test}}$, which are also generated by the same model, using their p-D latent parameters. Specifically, if $\boldsymbol{U}^\star$ is the learnt p-D subspace from the collection of training data $\{\boldsymbol{X}_i^c\}_{i=1}^S$, then $\boldsymbol{X}_{\text{test}}$ can be compressed as $\boldsymbol{Z}_{\text{test}} := (\boldsymbol{U}^\star)^{\mathsf{H}}\boldsymbol{X}_{\text{test}}$. We assume that the modeling errors $\boldsymbol{E}_i$ are independent, and the dimensionality reduction problem can now be solved as a static PCA problem (3). Furthermore, our target problem corresponds to the DCO setting considered in (8) as the data points are collected locally by the machines.

To illustrate the performance of the distributed PCA method (DistPM), we consider the data model of (25) with $N = 1000$, $p = 2$ and $T_i = 2$; while the SNR is fixed at 25dB. We consider a network with $S = 100$ nodes, connected through a mean degree-10 small world graph with rewiring probability of 0.2. Fig. 6 shows the reconstruction error in different settings for the DistPM (DCO) with different number of AC rounds.

The development of distributed dimensionality reduction techniques provide both efficient storage and preprocessing (denoising) of the data for machine learning. Fig. 7 shows an application of dimensionality reduction in data clustering. We consider a set of high-dimensional data distributed over $S = 100$ nodes. Specifically, the data consist of four separated clusters. Each cluster is a high-dimensional embedding of a swiss roll using the model in (25) with $N = 200$. Fig. 7(a) shows the original clusters of swiss roll data that are used to generate the high-dimensional data. Fig. 7(b) shows the result of dimensionality reduction using the estimated subspace with DistPM (DCO). We then perform Ward's hierarchical clustering algorithm [41] on both the original high-dimensional data and the projected data[3]. The normalized mutual information (NMI) is used to measure the clustering performance. Fig. 7(c) and 7(b) show the detected clusters with NMI = 0.9785 and 0.9797, respectively.

V. Distributed PCA Methods for DRO

In this section, we focus on the DRO setting and survey the available distributed PCA strategies in the literature. An interesting feature under this setting is that the goal of each agent is to compute a *portion* of the orthogonal transformation $\boldsymbol{U}^\star$ required. This requires one to design algorithms carefully in order to take advantage of these requirements while respecting the DRO data structure. We remark that most of the prior

[3]Here we let the nodes to broadcast their local projected data to the network.

REVIEW COPY

work for this data structure are designed for the mesh network scenario.

A. Distributed Power Method (DRO)

The distributed power method (DistPM) was developed in [18] for the DRO setting. In a nutshell, the method combines average consensus and power method to estimate the principal subspace distributively. Note that this is a *batch* method that applies to the *static* setting in (3).

We first derive the DistPM for just the top eigenvector of $\hat{\boldsymbol{R}}_x$, denoted as $\boldsymbol{u}_1^\star$. To set up the stage, we define $\boldsymbol{u}_{1,i}[k]$ as the local portion of the top eigenvector kept by the ith agent at iteration k and $\boldsymbol{u}_1[k] := (\boldsymbol{u}_{1,1}[k]; \ldots; \boldsymbol{u}_{1,S}[k])$ as the *global* eigenvector estimate. To initialize, each agent generates a random vector $\boldsymbol{u}_{1,i}[0] \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ independently.

Consider the kth power iteration in (14), we see that the normalization step may be relegated to the end of the iterations as it only involves a scalar multiplication. Instead, we focus on the second step in (14). Observe that the ith row partition in $\boldsymbol{u}_1[k+1]$ can be written as:

$$\begin{aligned} &\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}_i(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_1[k] \\ &= \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}_i(t) \left(\sum_{j=1}^{S} \boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_{1,j}[k] \right) , \end{aligned} \tag{26}$$

note that we are using the *unnormalized* vector $\boldsymbol{u}_{1,j}[k]$ on the right hand side of the equation. The vectors $\boldsymbol{x}_j(t)$ and $\boldsymbol{u}_{1,j}[k]$ are known to the jth agent. Therefore, the inner products inside the summation are *locally* computable by the jth agent. This inspires us to replace the summation with the previously introduced AC subroutine. Specifically, the required computation can be performed by the pseudo-code:

Distributed Power Method (DistPM) for DRO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_{1,i}[0] \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:

$$\boldsymbol{u}_{1,i}[k+1] = \frac{S}{T}\sum_{t=1}^{T} \boldsymbol{x}_i(t)\mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_{1,j}[k]\}_{j=1}^{S}; L\big)$$

for all $k \geq 0$ and we terminate after $k \geq K-1$.
3) Denote $\boldsymbol{u}_{1,i}[K]$ as the un-normalized top eigenvector kept by the ith agent for all $i \in \{1, ..., S\}$.

Note that we have replaced the inner product $\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_1[k]$ by its distributed average approximation $\mathsf{AC}_i(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_{1,j}[k]\}_{j=1}^{S}; L)$.

The procedure above finds the ith block of the *unnormalized* top eigenvector of $\hat{\boldsymbol{R}}_x$. As a final step, we shall take care of the normalization factor which we have ignored. Observe that

$$\|\boldsymbol{u}_1[K]\| = \sqrt{\textstyle\sum_{i=1}^{S} \|\boldsymbol{u}_{1,i}[K]\|^2} , \tag{27}$$

once again we observe that the terms inside the summation are merely the squared norm of the local estimate $\boldsymbol{u}_{1,i}[K]$, which is obviously known to the ith agent. This can be replaced by another AC subroutine. Let us define:

$$\mathsf{norm}_i := \sqrt{S \cdot \mathsf{AC}_i(\{\|\boldsymbol{u}_{1,j}[K]\|^2\}_{j=1}^{S}; L')} \approx \|\boldsymbol{u}_1[K]\| , \tag{28}$$

where we set $L' \gg L$. Finally, we denote the obtained partial eigenvector as $\hat{\boldsymbol{u}}_{1,i}$ such that

$$\hat{\boldsymbol{u}}_{1,i} := \big(\mathsf{norm}_i\big)^{-1} \boldsymbol{u}_{1,i}[K] , \tag{29}$$

and the concatenated version of it as $\hat{\boldsymbol{u}}_1 := (\hat{\boldsymbol{u}}_{1,1}; \ldots; \hat{\boldsymbol{u}}_{1,S})$. We can also compute the eigenvalue λ_1 associated with $\boldsymbol{u}_1^\star$ by observing:

$$\lambda_1 = \hat{\boldsymbol{u}}_1^{\mathsf{H}} \Big(\underbrace{\tfrac{1}{T}\textstyle\sum_{t=1}^{T} \boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\hat{\boldsymbol{u}}_1}_{\approx(\boldsymbol{u}_{1,1}[K];\ldots;\boldsymbol{u}_{1,S}[K])} \Big) . \tag{30}$$

The above can be approximated by:

$$\hat{\lambda}_{1,i} = S \cdot \mathsf{AC}_i\big(\{\hat{\boldsymbol{u}}_{1,j}^{\mathsf{H}}\boldsymbol{u}_{1,j}[K]\}_{j=1}^{S}; L\big) . \tag{31}$$

However, we note that this step is optional as the eigenvalue is not required in the computation of the second, third, etc. eigenvectors.

To compute the second eigenvector $\boldsymbol{u}_2^\star$, we observe that it is equivalent to computing the top eigenvector for $(\boldsymbol{I} - \boldsymbol{u}_1^\star(\boldsymbol{u}_1^\star)^{\mathsf{H}})\hat{\boldsymbol{R}}_x$. In light of this, the respective power iteration can be approximated as:

$$\boldsymbol{u}_2[k+1] = \frac{1}{T}\sum_{t=1}^{T} (\boldsymbol{I} - \hat{\boldsymbol{u}}_1\hat{\boldsymbol{u}}_1^{\mathsf{H}})\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k] \tag{32}$$

Similar to the previous derivations in (26), the ith row block of the above can be written as:

$$\frac{1}{T}\sum_{t=1}^{T} \Big(\boldsymbol{x}_i(t) - \hat{\boldsymbol{u}}_{1,i} \underbrace{\hat{\boldsymbol{u}}_1^{\mathsf{H}}\boldsymbol{x}(t)}_{\approx \mathsf{prod}_{1,i}} \Big) \big(\underbrace{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k]}_{\approx \mathsf{prod}_{2,i}(k)} \big) , \tag{33}$$

Importantly, the highlighted inner products are replaced respectively by:

$$\begin{aligned} \mathsf{prod}_{1,i} &:= S \cdot \mathsf{AC}_i\big(\{\hat{\boldsymbol{u}}_{1,j}^{\mathsf{H}}\boldsymbol{x}_j(t)\}_{j=1}^{S}; L\big) \\ &\approx \hat{\boldsymbol{u}}_1^{\mathsf{H}}\boldsymbol{x}(t) \\ \mathsf{prod}_{2,i}(k) &:= S \cdot \mathsf{AC}_i\big(\{(\boldsymbol{x}_j(t))^{\mathsf{H}}\boldsymbol{u}_{2,j}[k]\}_{j=1}^{S}; L\big) \\ &\approx \boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k] . \end{aligned} \tag{34}$$

where the right hand side can be computed distributively using the AC subroutine. Finally, the power iteration is performed by:

$$\boldsymbol{u}_{2,i}[k+1] = \frac{1}{T}\sum_{t=1}^{T} \Big(\boldsymbol{x}_i(t) - \hat{\boldsymbol{u}}_{1,i} \cdot \mathsf{prod}_{1,i} \Big) \mathsf{prod}_{2,i}[k] . \tag{35}$$

We can compute $\hat{\boldsymbol{u}}_{3,i}, ..., \hat{\boldsymbol{u}}_{p,i}$ using a similar procedure, again in a distributed fashion.

REVIEW COPY

Theoretical Guarantees: The convergence of DistPM (DRO) for the static network between the agents, *i.e.,* with $\boldsymbol{W}[\ell] = \overline{\boldsymbol{W}}$ for all ℓ, has been analyzed in [21] in the asymptotic regime as $P \to \infty$. Here, we only present a simplified version of their result for the special case of $p = 1$. Importantly, we observe that when $L \to \infty$ such that the AC subroutine computes the exact averages, the DistPM method will be equivalent to the original/centralized power method. As such, we expect that the performance of DistPM will improve with L, as shown in the following:

Theorem V.1 *Consider the DistPM (DRO) which terminates after P power iterations. If $P \to \infty$, we have*

$$\|\hat{\boldsymbol{u}}_1 - \boldsymbol{u}_1^\star\| = \mathcal{O}\left(\frac{\lambda_1(\hat{\boldsymbol{R}}_x)}{\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x)} \cdot \lambda_{\text{conn.}}^L\right) . \tag{36}$$

The above theorem is derived from [21, Theorem 3]. Theorem V.1 bounds the difference between $\boldsymbol{u}_1^\star$ and $\hat{\boldsymbol{u}}_1$. It confirms our expectation that the DistPM's performance depends on the number of AC iterations L and in particular, the performance improves exponentially with L. Moreover, the error also depends on the spectral gap in $\hat{\boldsymbol{R}}_x$ such that the error is reduced when $\lambda_1(\hat{\boldsymbol{R}}_x) \gg \lambda_2(\hat{\boldsymbol{R}}_x)$. It can be seen that the required number of AC steps, L, has the same scaling as in Theorem IV.1.

B. Distributed Oja's Method

As opposed to the distributed power method, the Oja's method can be applied to the *dynamic* PCA problem (4) which is a non-convex, time varying, stochastic optimization problem. A distributed version of the Oja's method, named D-Oja, was developed in [23] for the DRO setting. Here, we focus on the setting with $p = 1$. The first step is to approximate the update (17) as:

$$\begin{aligned}&\boldsymbol{u}^{\mathsf{Oja}}(t+1) = \boldsymbol{u}^{\mathsf{Oja}}(t) + \\ &\gamma_t\Big(\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t) - |\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)|^2\boldsymbol{I}\Big)\boldsymbol{u}^{\mathsf{Oja}}(t) .\end{aligned} \tag{37}$$

Similar to the development of DistPM, the D-Oja method relies on interpreting the Oja's updates as a combination of several computations that are replaceable by distributed algorithms. In particular, consider the ith block of $\boldsymbol{u}^{\mathsf{Oja}}(t+1)$ in (37):

$$\begin{aligned}&\boldsymbol{u}_i^{\mathsf{Oja}}(t+1) = \boldsymbol{u}_i^{\mathsf{Oja}}(t) + \\ &\gamma_t\Big(\boldsymbol{x}_i(t) \cdot \underline{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)} - \boldsymbol{u}_i^{\mathsf{Oja}}(t) \cdot \big|\underline{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)}\big|^2\Big) .\end{aligned} \tag{38}$$

In particular, for the ith agent, it suffices to compute the complex-valued inner product $\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)$ in order to complete the update. To distribute the computation, again we observe that this inner product is computed from a number of terms that are computed locally at S agents:

$$\begin{aligned}\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t) &= \textstyle\sum_{i=1}^S \boldsymbol{x}_i^{\mathsf{H}}(t)\boldsymbol{u}_i^{\mathsf{Oja}}(t) \\ &\approx S \cdot \mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_j^{\mathsf{Oja}}(t)\}_{j=1}^S; L\big) ,\end{aligned} \tag{39}$$

where L is a sufficiently large integer controlling the number of average consensus steps used. To summarize, we have the pseudo-code:

Distributed Oja's (D-Oja) Method for DRO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_i^{\mathsf{Oja}}(0) \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:

$$\begin{aligned}\mathsf{prod}_i(t) &:= S \cdot \mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_j^{\mathsf{Oja}}(t)\}_{j=1}^S; L\big) , \\ \boldsymbol{u}_i^{\mathsf{Oja}}(t+1) &= \boldsymbol{u}_i^{\mathsf{Oja}}(t) + \gamma_t\Big(\mathsf{prod}_i(t) \cdot \boldsymbol{x}_i(t) \\ &\qquad - |\mathsf{prod}_i(t)|^2 \cdot \boldsymbol{u}_i^{\mathsf{Oja}}(t)\Big) .\end{aligned} \tag{40}$$

for all $t \geq 0$.

For general $p > 1$, the authors in [23] proposed an alternative method that involves tackling a different objective function than (15). The developed method will be described in the next subsection.

Theoretical Guarantees: The asymptotic [*i.e.,* when $t \to \infty$] convergence of D-Oja method has been analyzed in [23] under the assumption that $\boldsymbol{x}(t)$ is stationary. For simplicity, we also consider the case of static network. We have

Theorem V.2 *Suppose that the step size γ_t is sufficiently small and L is sufficiently large, then the D-Oja method* (40) *can be approximated as an ODE. Furthermore, the ODE has a stable equilibrium $\hat{\boldsymbol{u}}_1$ that satisfies:*

$$\|\boldsymbol{u}_1^\star - \hat{\boldsymbol{u}}_1\| \leq 3S\lambda_1(\hat{\boldsymbol{R}}_x) \cdot \lambda_{\text{conn.}}^L + \mathcal{O}(S^2\lambda_{\text{conn.}}^{2L}) . \tag{41}$$

The proof can be found in [23, Lemma 4.2]. Similar to the results for DistPM (DRO), Theorem V.2 also shows an exponential dependence on the number of AC steps L in the error produced by the D-Oja method.

C. Power-Oja's method

As discussed earlier on, the Oja's method (as well as the D-Oja method) is known to converge slowly, *i.e.,* at most at the rate of $\mathcal{O}(1/t)$. To accelerate the convergence rate, the authors in [25] proposed a Power-Oja's (P-Oja) method that combines the fast convergence of power method with the adaptivity of the Oja's method. Notice that the power method is a *batch* method that requires the random process $\boldsymbol{x}(t)$ to be stationary. As a compromise, we assume *quasi-stationarity* for $\boldsymbol{x}(t)$ such that the process is stationary over the period $\mathcal{I}_k$ in which $\boldsymbol{x}(\tau)$ has the same distribution for all $\tau \in \mathcal{I}_k$. Furthermore, in contrast to the Oja's method, the P-Oja's method tackles the following dynamic PCA problem:

$$\boldsymbol{U}^\star(k) \in \arg\min_{\boldsymbol{U} \in \mathbb{C}^{N \times p}} f_\tau(\boldsymbol{U}) \text{ s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} , \tag{42}$$

where $\tau \in \mathcal{I}_k$ and

$$f_\tau(\boldsymbol{U}) := \mathbb{E}\Big[\mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\boldsymbol{x}(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau)\Big)\Big] . \tag{43}$$

It can be shown that the above problem is equivalent to (4).

We apply stochastic approximation for (42) to obtain a tractable objective function. Let $\mathcal{B}_{k,s} \subseteq \mathcal{I}_k$ be the sth batch in period $\mathcal{I}_k$, the objective function in the above can be approximated by:

$$\hat{f}(\boldsymbol{U}; \mathcal{B}_{k,s}) := \mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\hat{\boldsymbol{R}}(\mathcal{B}_{k,s})\Big) \tag{44}$$

REVIEW COPY

where

$$\hat{\boldsymbol{R}}(\mathcal{B}_{k,s}) := \frac{1}{|\mathcal{B}_{k,s}|}\sum_{\tau\in\mathcal{B}_{k,s}} \boldsymbol{x}(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau)\ , \tag{45}$$

which is a *mini-batch* sampled correlation matrix.

From here, one may apply the SGD method and update the principal subspaces by taking the gradient of $\hat{f}(\boldsymbol{U};\mathcal{B}_{k,s})$. Doing so results in a method similar to Oja's which may suffer from the same slow convergence rate. Instead, let P be a predefined integer constant, we consider a modified stochastic approximation function:

$$\hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s}) := \mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\Big)\ . \tag{46}$$

Importantly, the optimal solution set to the modified stochastic approximation function is the same as the original one:

$$\begin{aligned}\arg\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}}\ & \hat{f}(\boldsymbol{U};\mathcal{B}_{k,s}) = \arg\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}}\ \hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s})\\ \text{s.t. }& \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I} \qquad\qquad \text{s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I}\ .\end{aligned} \tag{47}$$

The benefit of considering $\hat{f}^{\mathsf{POja}}(\cdot)$ in lieu of $\hat{f}(\cdot)$ is that the former admits a better *spectral gap* for the sampled correlation since

$$\frac{\sigma_p(\hat{\boldsymbol{R}}^P) - \sigma_{p+1}(\hat{\boldsymbol{R}}^P)}{\sigma_p(\hat{\boldsymbol{R}}^P)} > \frac{\sigma_p(\hat{\boldsymbol{R}}) - \sigma_{p+1}(\hat{\boldsymbol{R}})}{\sigma_p(\hat{\boldsymbol{R}})}\ , \tag{48}$$

as we notice that $\sigma_p(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})) = \sigma_p(\hat{\boldsymbol{R}}(\mathcal{B}_{k,s}))^P$, where $\sigma_p(\boldsymbol{R})$ denotes the pth largest singular value of $\boldsymbol{R}$. As shown in the analysis of [35], the size of the spectral gap $\sigma_p(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})) - \sigma_{p+1}(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s}))$ is an important factor in determining the convergence speed of subspace estimation/tracking algorithms. Therefore, we anticipate that using $\hat{f}^{\mathsf{POja}}(\cdot)$ should yield a faster PCA algorithm than using $\hat{f}^{\mathsf{Oja}}(\cdot)$. Finally, observe that the gradient of the new stochastic approximation function is:

$$\begin{aligned}\nabla\hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s}) &= -2\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}\\ &\quad + \hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} + \boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}\ .\end{aligned} \tag{49}$$

Remarkably, $\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}$ is similar to running a power method for P steps with the columns of $\boldsymbol{U}$ as initialization vectors (an insight that motivated this distributed PCA approach). Lastly, relaxing the Grassmanian manifold constraint $\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I}$ yields the P-Oja method:

$$\boldsymbol{U}(k,s+1) = \boldsymbol{U}(k,s) - \gamma_s\nabla\hat{f}^{\mathsf{POja}}(\boldsymbol{U}(k,s);\mathcal{B}_{k,s})\ , \tag{50}$$

this is similar to the NOja's method considered in [36].

Note that P-Oja method requires a non-trivial batch size, $|\mathcal{B}_{k,s}| \geq 2$ to show its benefits, which depend on a sufficient increase in the spectral gap.

Distributed Method: The authors in [25] considered a distributed extension for the P-Oja method. Their philosophy is similar to that employed by the DistPM and D-Oja method, as we observe that the update of the ith block [cf. (49)] is:

$$\boldsymbol{U}_i(s) - \gamma_s\big[\nabla\hat{f}^{\mathsf{POja}}(\boldsymbol{U}(s);\mathcal{B}_s)\big]_i\ , \tag{51}$$

where we have dropped the index of k for simplicity and

$$\begin{aligned}&\big[\nabla\hat{f}^{\mathsf{POja}}(\boldsymbol{U}(s);\mathcal{B}_s)\big]_i :=\\ &-2\underbrace{[\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i}_{\approx\mathsf{prod}_{1,i}(s)} + \underbrace{[\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i}_{\approx\mathsf{prod}_{1,i}(s)}\underbrace{\boldsymbol{U}^{\mathsf{H}}(s)\boldsymbol{U}(s)}_{\approx\mathsf{prod}_{2,i}(s)}\\ &+\boldsymbol{U}_i(s)\underbrace{\boldsymbol{U}^{\mathsf{H}}(s)\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)}_{\approx\mathsf{prod}_{3,i}(s)}\ .\end{aligned} \tag{52}$$

The required computation boils down to evaluating the highlighted terms in the above. First, we observe that $\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)$ is the result of applying P power iterations with the sampled correlation matrix $\hat{\boldsymbol{R}}(\mathcal{B}_s)$, initialized on $\boldsymbol{U}(s)$. Each of the power iteration can be handled in a similar fashion as in the DistPM. In particular, let $\hat{\boldsymbol{U}}(s,q)$ be the intermediate variable at the qth power iteration, $\hat{\boldsymbol{U}}_i(s,q)$ be its ith row block, and $\hat{\boldsymbol{U}}(s,1) = \boldsymbol{U}(s)$, the power iteration can be approximated as

$$\begin{aligned}&\hat{\boldsymbol{U}}_i(s,q+1)\\ &= \frac{S}{|\mathcal{B}_s|}\sum_{\tau\in\mathcal{B}_s}\boldsymbol{x}_i(\tau)\cdot\mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(\tau)\hat{\boldsymbol{U}}_j(s,q)\}_{j=1}^S;L\big)\\ &\approx \frac{1}{|\mathcal{B}_s|}\sum_{\tau\in\mathcal{B}_s}\boldsymbol{x}_i(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau)\hat{\boldsymbol{U}}(s,q)\ .\end{aligned} \tag{53}$$

Repeating the above for P iterations yields

$$\mathsf{prod}_{1,i}(s) := \hat{\boldsymbol{U}}_i(s,P+1) \approx [\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i\ . \tag{54}$$

For the remaining terms, we can handle them easily by:

$$\begin{aligned}\mathsf{prod}_{2,i}(s) &:= S\cdot\mathsf{AC}_i\big(\{\boldsymbol{U}_j^{\mathsf{H}}(s)\boldsymbol{U}_j(s)\}_{j=1}^S;L\big)\\ &\approx \textstyle\sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)\boldsymbol{U}_j(s)\ ,\end{aligned} \tag{55}$$

$$\begin{aligned}\mathsf{prod}_{3,i}(s) &:= S\cdot\mathsf{AC}_i\big(\{\boldsymbol{U}_j^{\mathsf{H}}(s)\mathsf{prod}_{1,j}(s)\}_{j=1}^S;L\big)\\ &\approx \sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)\mathsf{prod}_{1,j}(s) \approx \sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)[\hat{\boldsymbol{R}}^P\boldsymbol{U}(s)]_j\ .\end{aligned} \tag{56}$$

Finally, the P-Oja update for agent i can be approximated as:

$$\begin{aligned}\boldsymbol{U}_i(s+1) = \boldsymbol{U}_i(s) - \gamma_s\Big(&\boldsymbol{U}_i(s)\mathsf{prod}_{3,i}(s)\\ &+\mathsf{prod}_{1,i}(s)\mathsf{prod}_{2,i}(s) - 2\mathsf{prod}_{1,i}(s)\Big)\ .\end{aligned} \tag{57}$$

This demonstrates that the P-Oja update can be replaced by a number of average consensus steps to be computed distributively. We remark that the P-Oja update can be reduced to the multiple subspaces-tracking D-Oja method (*i.e.,* when $p > 1$) by setting $P = 1$.

D. Application Examples

1) Distributed Distance Matrix Estimation: We demonstrate that the distance matrix estimation problem can be cast as a PCA problem, which naturally admits a DRO data structure when we consider a sensor network setting. Here, agent/sensor i gathers an d-dimensional observation $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1,...,S$. Our goal is to compute the pairwise distances $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ which are necessitated by clustering problems or dimensionality reduction with ISOMAP [20]. Let us observe that

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = \sqrt{\|\boldsymbol{x}_i\|^2 + \|\boldsymbol{x}_j\|^2 - 2\boldsymbol{x}_i^{\top}\boldsymbol{x}_j}\ . \tag{58}$$

REVIEW COPY

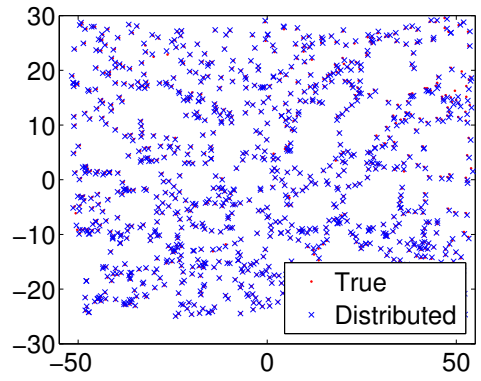

Fig. 8. **Distributed Distance Matrix Estimation**. Comparing the true and estimated distance matrix on a sensor network with $S = 1000$ sensors. We set $K = 50, L = 500$ for the DistPM and reconstructed the distance matrix through finding its rank-2 approximation.

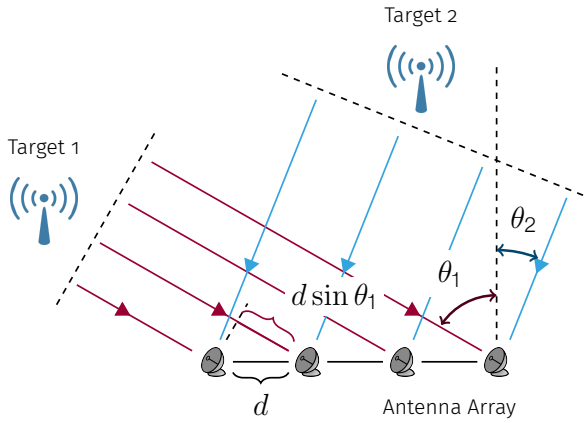

Fig. 9. **Direction of Arrival Estimation**. Our goal is to estimate the directions θ_1, θ_2, ... of the *targets* in the figure from the signals received at the *uniform linear array* (ULA).

Importantly, the squared norms and inner products in the above can be obtained from the following matrix:

$$\begin{aligned} \boldsymbol{A} &:= \boldsymbol{X}\boldsymbol{X}^\top \in \mathbb{R}^{S\times S}, \text{ where} \\ \boldsymbol{X} &:= (\boldsymbol{x}_1 \ \cdots \ \boldsymbol{x}_S)^\top \in \mathbb{R}^{S\times d}. \end{aligned} \tag{59}$$

We have $\boldsymbol{x}_i^\top \boldsymbol{x}_j = A_{ij}$ and therefore the pairwise distance can be computed through $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = \sqrt{A_{ii} + A_{jj} - 2A_{ij}}$. To estimate the pairwise distances, we can therefore compute the rank-p approximation of $\boldsymbol{A}$:

$$\hat{\boldsymbol{A}}_p = \textstyle\sum_{r=1}^p \lambda_r \boldsymbol{u}_r^\star (\boldsymbol{u}_r^\star)^\top \approx \boldsymbol{A}\ , \tag{60}$$

In particular, $\hat{\boldsymbol{A}}_p$ is computed from the eigenvalue-vector pairs $\{\lambda_r, \boldsymbol{u}_r^\star\}_{r=1}^p$, *i.e.,* it is a PCA problem. We notice that the data is structured in a DRO setting since each agent holds a row vector in $\boldsymbol{X}$. This is a static PCA problem as the data matrix does not change over time. Thus DistPM is a suitable algorithm. Fig. 8 compares the true and estimated distance matrix using DistPM (DRO) on a sensor network with $S = 1000$ sensors/agents, with the graph topology arranged according to a random geometric graph. The DistPM (DRO) finds the distance matrix accurately.

2) Distributed Direction-of-Arrival Tracking: We next illustrate how to track the direction-of-arrival (DoA) of multiple objects in a massive antenna array system using the distributed PCA methods. Consider a scenario where the (groups of) antennas are arranged in a *uniform linear array* (ULA) configuration, receiving signals from a far object. An example is depicted in Fig. 9. When there are p objects to be tracked, it is well known [1], [2] that the signal received at the antennas is:

$$\boldsymbol{x}(t) = \textstyle\sum_{q=1}^p \boldsymbol{a}(d, \theta_q)\boldsymbol{z}_q(t) + \boldsymbol{e}(t)\ , \tag{61}$$

where

$$\boldsymbol{a}(d, \theta_q) := (1,\ e^{-j\omega_c d \sin\theta_q/\lambda}, \ldots,\ e^{-j\omega_c (S-1) d \sin\theta_q/\lambda})^\top,$$

such that λ is the wavelength and ω_c is the carrier frequency, $\boldsymbol{z}_q(t)$ is the signal transmitted from the qth object, and $\boldsymbol{e}(t)$ is a white additive noise with variance σ^2. Importantly, the vectors $\{\boldsymbol{a}(d, \theta_q)\}_{q=1}^p$ are mutually orthogonal when the number of antennas is large, provided that $\theta_q \neq \theta_{q'}$, thus the p-D principal subspace of the signal $\boldsymbol{x}(t)$ is given by $\boldsymbol{A} := (\boldsymbol{a}(d, \theta_1) \ \cdots \ \boldsymbol{a}(d, \theta_p))$ and the model fits into the one described by (2). The direction-of-arrival (DoA) for multiple objects can thus be found by extracting this principal subspace. This leads to the celebrated MUSIC/ESPRIT methods [2]. For instance, suppose that $\hat{\boldsymbol{U}}$ is the estimated principal subspace of $\boldsymbol{x}(t)$, the MUSIC method finds the p largest 'peaks' in the following *pseudo spectrum*

$$P(\theta) = \frac{\boldsymbol{a}^{\mathsf{H}}(d, \theta)\boldsymbol{a}(d, \theta)}{\boldsymbol{a}^{\mathsf{H}}(d, \theta)\boldsymbol{U}_\perp \boldsymbol{U}_\perp^{\mathsf{H}} \boldsymbol{a}(d, \theta)}\ , \tag{62}$$

where $\hat{\boldsymbol{U}}_\perp$ is an $N \times (N-p)$ matrix containing $N-p$ orthonormal vectors that are orthogonal to $\hat{\boldsymbol{U}}$ such that $\boldsymbol{U}_\perp^{\mathsf{H}} \hat{\boldsymbol{U}} = \boldsymbol{0}$.

In a distributed setting, each element of $\boldsymbol{x}(t)$ corresponds to a scalar signal received by an antenna operated by a sensor in the network. We have $S = N$ such sensors/antennas. The problem of determining $\hat{\boldsymbol{U}}$, the p-D principal subspace of $\boldsymbol{x}(t)$, is thus a *dynamic* PCA problem (4) with DRO data, as the object is moving.

A simulation example of tracking the DoA of a single moving object using an ULA with $S = N = 30$ antennas is shown below. In the example, the antennas/sensors are connected through a mean degree-4 small world graph with rewiring probability 0.2. Each of the antenna receives a total of $T = 1500$ samples and the SNR in (61) is 20dB. The heatmap for evolution of psuedo spectrum and the evolution of normalized objective value (NOV) [cf. (43)], *i.e.,* :

$$\mathsf{NOV}(t) := \frac{\mathbb{E}\|\boldsymbol{x}(t) - \boldsymbol{u}(t)\boldsymbol{u}^{\mathsf{H}}(t)\boldsymbol{x}(t)\|_2^2}{\mathbb{E}\|\boldsymbol{x}(t)\|_2^2}\ , \tag{63}$$

are shown in Fig. 10 and 11, respectively. Once again, we observe that the decentralized approach produces an accurate estimate of the DoA.

3) Distributed Beamforming: A related application is that of distributed PCA methods for *beamforming*, where the aim is to find the weights of the maximal ration combiner, to combine the signal at each antenna element in such a way to maximize

REVIEW COPY

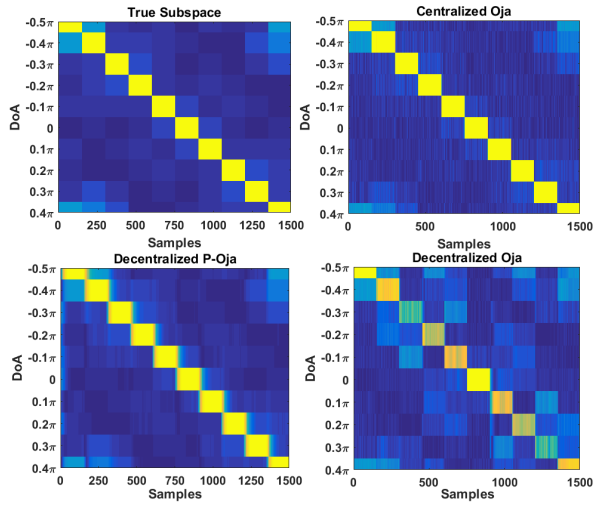

Fig. 10. **DoA tracking with Distributed PCA Methods**. The top-left plot shows the evolution of DoA against time/batch number, as indicated by the yellow dot. The remaining three plots show the evolution of pseudo-spectrum (62) using the Oja, D-Oja and distributed P-Oja methods. Each column in the plot represents a pseudo-spectrum. The P-Oja method is set with the batch size of $B = 5$ with a step size of $\gamma_t = 0.25$ while the Oja's method is set with a step size of $\gamma_t = 0.05$.

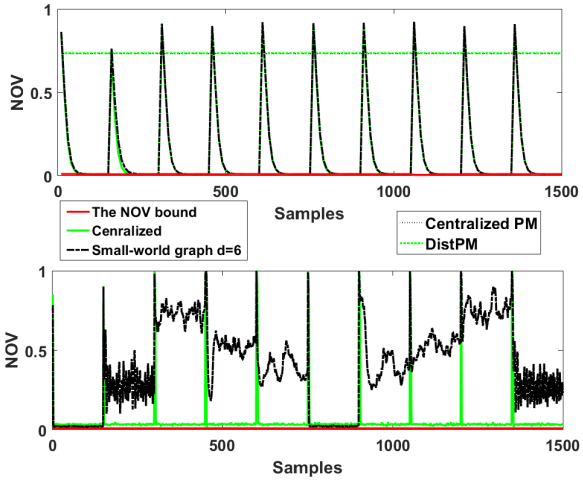

Fig. 11. **Evolution of the NOV against time in DoA tracking**. We apply the centralized and decentralized P-Oja and Oja methods for subspace tracking. The top figure is for the P-Oja method where the batch size is $B = 5$, the power iteration parameter is set as $P = 20$ and a step size $\gamma_t = 0.25$; the bottom figure is the Oja method with a step size $\gamma_t = 0.05$. For both methods, the number of gossip round is $L = 10$.

the SNR. This problem fits into the setting of a dynamic PCA problem with DRO data as we deploy subarrays of antennas.

Our physical set up is illustrated by Fig. 12, where we consider grouping $N = 256$ antennas into $S = 64$ sub-arrays, each with $N_i = 4$ antennas. The processor units form a mean degree-6 small world graph with rewiring probability of 0.2. Each processor unit receives $T = 1500$ samples with an SNR of 20dB. The signal $\boldsymbol{x}(t)$ is generated with $p = 2$ dimensional principal subspace. In Fig. 13, we compare the NOV [cf. (63)] of running the different PCA methods in this example. The numerical results show that the P-Oja method converges faster

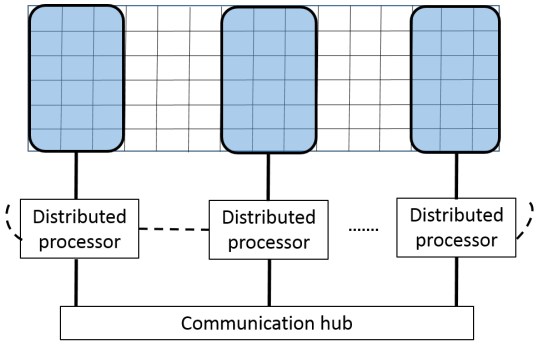

Fig. 12. **Grouping Antennas into Subarrays with Distributed Processors**. Each distributed processor can be treated as a supernode and communicate with the others.

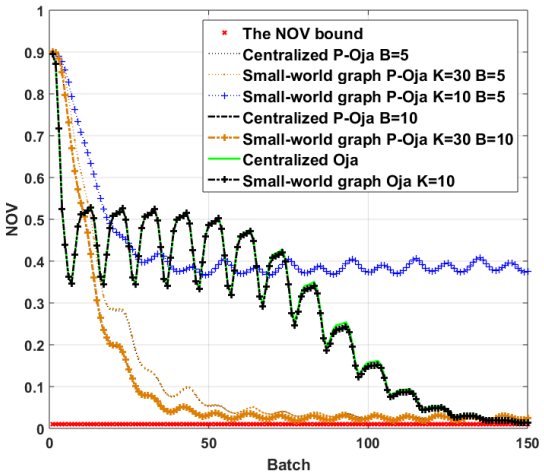

Fig. 13. **Evolution of NOV against time in Distributed Beamforming**. For Oja's and P-Oja method, we set the step size as $\gamma_t = 5 \times 10^{-4}$ and $\gamma_t = 0.01B$, respectively, where B is the batch size used.

than the Oja's method in the (quasi)-static setting, and the decentralized algorithms track the subspaces accurately.

E. Other Methods and Summary

In addition to the surveyed methods above, we note that an ADMM based algorithm has been developed in [24]. There, the authors proposed to relax (3) into a matrix factorization problem as:

$$\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p},\boldsymbol{Y}\in\mathbb{C}^{p\times T}} \|\boldsymbol{X}-\boldsymbol{U}\boldsymbol{Y}\|_{\mathrm{F}}^2 , \tag{64}$$

where they have relaxed the constraint that $\boldsymbol{Y} = \boldsymbol{U}^{\mathsf{H}}\boldsymbol{X}$. The problem (64) is then written as:

$$\begin{aligned} \min_{\boldsymbol{U}_i\in\mathbb{C}^{N_i\times p},\boldsymbol{Y}_i\in\mathbb{C}^{p\times T},\forall i} \quad & \textstyle\sum_{i=1}^{S} \|\boldsymbol{X}_i^r - \boldsymbol{U}_i\boldsymbol{Y}_i\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{Y}_i = \boldsymbol{Y}_j,\ \forall\ (i,j)\in\mathcal{E}\ . \end{aligned} \tag{65}$$

The optimization problem shown above is then tackled using an alternating minimization approach, in which the update of the coupled variables $\boldsymbol{Y}_i$ is handled by the celebrated ADMM method [42]. See also [27] for a related study.

To conclude this section, in Table I, we summarize the PCA strategies described in the above for DRO data by

REVIEW COPY

Methods	Comp.	Commun.	Convergence
DistPM [18]	$\mathcal{O}(TN_i)$	$\mathcal{O}(TL)$	fast
D-Oja [23]	$\mathcal{O}(N_i)$	$\mathcal{O}(L)$	slow
P-Oja [25]	$\mathcal{O}(PBN_i)$	$\mathcal{O}(PBL)$	medium

TABLE I
COMPARISON OF THE DISTRIBUTED PCA METHODS (DRO). WE FOCUS ON THE COMPUTATION AND COMMUNICATION COMPLEXITIES PER ITERATION UNDER THE SPECIAL CASE OF $p = 1$. NOTE THAT THE COMPUTATION COMPLEXITY IS COUNTED AT A *per-agent* LEVEL. FOR THE P-OJA METHOD, WE ASSUME THAT BATCH SIZE IS CONSTANT SUCH THAT $B = |\mathcal{B}_{k,s}|$ FOR ALL k, s.

comparing the computation and communication complexities required by each method. Notice that we chose not to compare the ADMM approach in [24] as the latter involves multiple nested loops which hinders a fair comparison. As seen from the comparison, even though the DistPM enjoys the fastest convergence rate, its computation and communication costs are also T times larger than the D-Oja method. To this end, the P-Oja method strikes a balance between the two methods, requiring moderate computation/communication cost and offering faster convergence rate than the D-Oja method.

VI. RELATED & OPEN PROBLEMS

In this section we first survey a few related problems and the state-of-the-art on how some of these problems have been solved using distributed computations. We then discuss open problems that are relevant to distributed PCA.

1) Distributed Robust PCA: The robust PCA problem [43], [44] deals with scenarios when the observed data is contaminated with outlier noise. In particular, we consider a modified observation model from (2):

$$\boldsymbol{x}(t) = \boldsymbol{U}_p \boldsymbol{z}(t) + \boldsymbol{c}(t) + \boldsymbol{e}(t) \ , \tag{66}$$

where $\boldsymbol{c}(t)$ represents the *outlier noise*, e.g., a sparse vector. Such model may be used to handle situations of faulty sensors or anomalies in the observed data. In particular, when $\boldsymbol{c}(t)$ is sufficiently sparse and the subspace dimension is sufficiently low, it is possible to recover the subspace spanned by $\boldsymbol{U}_p$ through solving a convex program [44], [45].

Recently, distributed methods have been developed for the robust PCA problem. For instance, in the DCO setting [46] proposed to tackle a modified problem of (8) with a distributed sub-gradient descent method. The authors therein essentially solve:

$$\min_{\boldsymbol{U} \in \mathbb{C}^{N \times p}} \ \sum_{t=1}^{T} \|(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{x}(t)\|_1 \text{ s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} \ , \tag{67}$$

where we have replaced the Euclidean norm in (3) by the sparsity-inducing ℓ_1 norm; [47] proposed a detection method for samples that are contaminated by outlier noise and prune away those samples. Notice that these methods are only compatible with the DCO setting. For the DRO case, [48] proposed an ADMM algorithm for the corresponding robust PCA problem.

2) Distributed Canonical Correlation Analysis: As a closely related problem to PCA, the canonical correlation analysis (CCA) problem [49] is widely applied in blind source separation, array processing, medical imaging and word embedding; see [50]–[52] for the applications.

In a nutshell, the CCA problem involves simultaneously analyzing two datasets of *paired* data, $\{\boldsymbol{x}(t)\}_{t=1}^{T}, \{\boldsymbol{y}(t)\}_{t=1}^{T}$, where $\boldsymbol{x}(t) \in \mathbb{C}^{N_1}$, $\boldsymbol{y}(t) \in \mathbb{C}^{N_2}$, in order to find a low-rank structure in the corresponding cross-correlation matrix $\boldsymbol{R}_{xy} := \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{y}^{\mathsf{H}}(t)]$. Each of the two datasets offers a different *view* of the same latent structure, e.g., $\boldsymbol{x}(t)$ correspond to the spelling features for document t while $\boldsymbol{y}(t)$ correspond to the contextual features. Mathematically, similar to the PCA problem with $p = 1$, the CCA problem can be given by:

$$\max_{\boldsymbol{u}_1 \in \mathbb{C}^{N_1}, \boldsymbol{v}_1 \in \mathbb{C}^{N_2}} \frac{\boldsymbol{u}_1^{\mathsf{H}} \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{y}^{\mathsf{H}}(t)]\boldsymbol{v}_1}{\sqrt{\mathbb{E}[|\boldsymbol{u}_1^{\mathsf{H}}\boldsymbol{x}(t)|^2] \cdot \mathbb{E}[|\boldsymbol{v}_1^{\mathsf{H}}\boldsymbol{y}(t)|^2]}} \ . \tag{68}$$

Efficient algorithms for CCA have been developed, e.g., [53]–[55]. However, to the best of our knowledge, the literature on distributed CCA problem is lacking except for a recent work in [56]. Therein, the authors considered a DRO setting and applied the idea of alternating optimization to tackle (68). On the down side, the proposed algorithm only works in a tree-network setting.

3) Distributed Dictionary Learning: The dictionary learning problem [57]–[59] aims at learning an *over-complete dictionary* which can describe the dataset with a sparse linear combination of the atoms. Let $\boldsymbol{X} \in \mathbb{R}^{N \times T}$ be the given dataset, the dictionary learning problem can be formulated as a matrix decomposition problem:

$$\min_{\boldsymbol{D}, \boldsymbol{Y}} \ \|\boldsymbol{X} - \boldsymbol{D}\boldsymbol{Y}\|_{\mathrm{F}}^2 \text{ s.t. } \|\boldsymbol{y}_t\|_0 \leq R_0, \ t = 1, ..., T \ , \tag{69}$$

where the ℓ_0 norm constraint on the columns of $\boldsymbol{Y}$, $\boldsymbol{y}_t$, ensures that the problem finds a *sparse* representation, and we have $\boldsymbol{D} \in \mathbb{R}^{N \times K}, \boldsymbol{Y} \in \mathbb{R}^{K \times T}$ such that the dictionary consists of K atoms. Problem (69) is challenging since both of its objective function and constraint are non-convex. As such, (69) is usually tackled by an alternating optimization procedure which iterates between two steps — a sparse coding step for updating $\boldsymbol{Y}$ and a dictionary learning step for updating $\boldsymbol{D}$ as well as the non-zero elements of $\boldsymbol{Y}$.

To tackle (69) distributively when the data is organized with the DCO structure [cf. (6)] one can follow the cloud K-SVD method in [38] to show how the techniques in distributed PCA can be applied. For the sparse coding step, it is obvious that the problem is decomposable due to the data structure. Let us denote the solution obtained at this step by $\boldsymbol{Y}^{(s)}$. For the dictionary learning step, [38] applies a cyclical update to optimize each atom (column) of $\boldsymbol{D}$ sequentially. For the kth atom, we consider:

$$\begin{aligned} &\min_{\boldsymbol{d}_k \in \mathbb{R}^N} \|\tilde{\boldsymbol{X}}^{(s)} - \boldsymbol{d}_k (\boldsymbol{y}_k^{(s),\mathrm{row}})^{\top}\|_{\mathrm{F}}^2 \\ \iff &\min_{\boldsymbol{d}_k \in \mathbb{R}^N} \|\tilde{\boldsymbol{X}}_R^{(s)} - \boldsymbol{d}_k (\boldsymbol{y}_{k,R}^{(s),\mathrm{row}})^{\top}\|_{\mathrm{F}}^2 \ , \end{aligned} \tag{70}$$

where $\tilde{\boldsymbol{X}}^{(s)}$ is obtained by taking the difference between $\boldsymbol{X}$ and the contributions from the atoms other than $\boldsymbol{d}_k$, $\boldsymbol{y}_k^{(s),\mathrm{row}}$ is the kth row of $\boldsymbol{Y}^{(s)}$, $\boldsymbol{y}_{k,R}^{(s),\mathrm{row}}$ is the sub-vector of $\boldsymbol{y}_k^{(s),\mathrm{row}}$

IEEE COPYRIGHT

retaining only the non-zero components of the latter, and $\tilde{\boldsymbol{X}}_R^{(s)}$ selects the columns of $\tilde{\boldsymbol{X}}^{(s)}$ with respect to $\boldsymbol{y}_{k,R}^{(s),\text{row}}$.

Importantly, the minimization on the right hand side of (70) can be solved by obtaining the *top* left and right singular vectors of $\tilde{\boldsymbol{X}}_R^{(s)}$. Note that this is precisely the distributed PCA problem with DCO data. As proposed in [38], a DistPM (DCO) method is then applied to compute the updated atom $\boldsymbol{d}_k^\star$ as well as updating the *non-zero* elements in $\boldsymbol{y}_k^{(s),\text{row}}$.

We remark that distributed dictionary learning has also been considered a number of prior works, e.g., [60]–[63] with a different approach towards the solution.

4) Distributed Low-rank Optimization: Distributed PCA methods can also be applied as a subroutine to solve low rank multi-agent optimization problem of the form:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{m_1\times m_2}} \sum_{i=1}^S f_i(\boldsymbol{\theta}) \;\; \text{s.t.} \;\; \|\boldsymbol{\theta}\|_* \leq R\,, \qquad (71)$$

which includes problems such as matrix completion as special cases [64]. Note $\|\boldsymbol{\theta}\|_*$ denotes the nuclear norm of the matrix $\boldsymbol{\theta}$. Each of $f_i(\boldsymbol{\theta})$ is the loss function attributed to the partition of data available at the ith agent/machine, which is assumed to be continuously differentiable but possibly non-convex. Importantly, in the above, the observed data can be split in an arbitrary fashion.

For large-scale problems with $m_1, m_2 \gg 1$, a popular method is to apply the so-called *projection-free* (a.k.a. Frank-Wolfe) algorithms [65], which amounts to finding the top singular vector of the gradient matrix at each iteration. Obviously, the top singular vector can be found using the PCA methods. Furthermore, as the gradient matrix is simply a sum of the *local* gradients, the associated PCA problem admits a DCO data structure. To this end, [39] applied the DistPM (DCO) with a set of carefully designed, time varying parameters L, K as a subroutine to tackle (71) in a distributed fashion, and the authors showed that the resulting decentralized Frank-Wolfe algorithm [66] converges at desirable rates for both convex and non-convex instances of (71). Similar effort can also be found in [67] for the master-slave architecture.

As an alternative, it is worthwhile mentioning that an heuristic solution to the matrix completion problem [64] (when the loss functions in (71) are squared Euclidean distances between entries of $\boldsymbol{\theta}$ and the observed data) can be found by iteratively computing the top singular vectors from a set of *partially observed* data, following the procedure proposed in [68]. Once again, this procedure can be applied to a distributed setting adopting the distributed PCA methods reviewed in this paper.

A. Open Problems

A problem in which the literature is lacking is in tackling the PCA problem distributively with *irregularly partitioned* data. The data may not conform to neither the DCO nor the DRO type exactly, but be divided in *an arbitrary fashion*. The the resulting PCA problem would essentially require each agent to infer the missing entries simultaneously as it computes the principal components.

Another interesting extension of distributed PCA algorithms, would be methods for solving the *tensor PCA* problem (or tensor decomposition in general) [69], which has rarely been studied in the distributed setting. Note that tensor models are effective in modeling high order relationships between observations, e.g., [70]. Some prior work on parallel tensor decomposition are found in [71]–[73], however, in addition to restricting the types of networks to operate the distributed algorithms on, these work either requires the data to be stored centrally [72], [73], or the same set be available at multiple agents at least in part [71]. These settings are more restrictive than the distributed data setting considered in distributed PCA. A fully distributed tensor PCA method has yet to be developed.

Lastly, thanks to the recent advancements in first order and non-convex optimization methods, a number of new computation techniques have been applied to the PCA problem, e.g., [74]–[76]. These new methods offer theoretically proven improvements and can overcome certain limitations in fast convergence for the traditional power and Oja's methods, which are the backbones for distributed PCA methods discussed in this paper. For instance, [75] proposed an PCA algorithm with a convergence rate that is independent of the spectral gap $\sigma_1(\hat{\boldsymbol{R}}_x) - \sigma_2(\hat{\boldsymbol{R}}_x)$; [76] combines the technique of variance reduction with acceleration to yield better dependence of the convergence rate on spectral gap. The methods above also demonstrate significant speedup in a centralized setting empirically, and it would be beneficial to explore the possibility of extending these benefits to the distributed PCA algorithms.

VII. Conclusions

The distributed PCA methods are motivated the increasing popularity of networked systems including sensor and computer networks. In this paper, we have surveyed recent advancements in distributed PCA methods, where signal processing strategies have been applied independently depending on how the data are acquired in the network. Compared to centralized PCA, these methods can efficiently harness the computation and storage resources at the distributed agents, as confirmed by the theoretical and empirical analysis presented.

References

[1] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE signal processing magazine*, vol. 13, no. 4, pp. 67–94, 1996.
[2] P. Stoica, *Introduction to spectral analysis*. Prentice hall, 1997.
[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
[4] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559–572, 1901.
[5] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.
[6] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 29.
[7] Y. Qu, G. Ostrouchov, N. Samatova, and A. Geist, "Principal component analysis for dimension reduction in massive distributed data sets," in *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 2002.
[8] Y. Liang, M.-F. F. Balcan, V. Kanchanapally, and D. Woodruff, "Improved distributed principal component analysis," in *Advances in Neural Information Processing Systems*, 2014, pp. 3113–3121.
[9] J. Fan, D. Wang, K. Wang, and Z. Zhu, "Distributed estimation of principal eigenspaces," *arXiv preprint arXiv:1702.06488*, 2017.

Review Copy

[10] R. Kannan, S. Vempala, and D. Woodruff, "Principal component analysis and higher correlations for distributed data," in *Conference on Learning Theory*, 2014, pp. 1040–1057.

[11] C. Boutsidis, D. P. Woodruff, and P. Zhong, "Optimal principal component analysis in distributed and streaming models," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 236–249.

[12] D. Garber, O. Shamir, and N. Srebro, "Communication-efficient algorithms for distributed stochastic principal component analysis," in *International Conference on Machine Learning*, 2017, pp. 1203–1212.

[13] Z.-J. Bai, R. H. Chan, and F. T. Luk, "Principal component analysis for distributed data sets with updating," in *International Workshop on Advanced Parallel Processing Technologies*. Springer, 2005, pp. 471–483.

[14] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson, "Distributed clustering using collective principal component analysis," *Knowledge and Information Systems*, vol. 3, no. 4, pp. 422–448, 2001.

[15] H. Qi, T.-W. Wang, and J. D. Birdwell, "Global principal component analysis for dimensionality reduction in distributed data mining," *Statistical data mining and knowledge discovery*, pp. 327–342, 2004.

[16] F. N. Abu-Khzam, N. F. Samatova, G. Ostrouchov, M. A. Langston, and A. Geist, "Distributed dimension reduction algorithms for widely dispersed data," in *IASTED PDCS*, 2002, pp. 167–174.

[17] J. Fellus, D. Picard, and P.-H. Gosselin, "Dimensionality reduction in decentralized networks by gossip aggregation of principal components analyzers," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2014, pp. 171–176.

[18] A. Scaglione, R. Pagliari, and H. Krim, "The decentralized estimation of the sample covariance," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*. IEEE, 2008, pp. 1722–1726.

[19] Y.-A. Le Borgne, S. Raybaud, and G. Bontempi, "Distributed principal component analysis for wireless sensor networks," *Sensors*, vol. 8, no. 8, pp. 4821–4850, 2008.

[20] M. E. Yildiz, F. Ciaramello, and A. Scaglione, "Distributed distance estimation for manifold learning and dimensionality reduction," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2009, pp. 3353–3356.

[21] W. Suleiman, M. Pesavento, and A. M. Zoubir, "Performance analysis of the decentralized eigendecomposition and ESPRIT algorithm," *IEEE Transactions on Signal Processing*, vol. 64, no. 9, pp. 2375–2386, 2016.

[22] S. B. Korada, A. Montanari, and S. Oh, "Gossip PCA," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 2011, pp. 209–220.

[23] L. Li, A. Scaglione, and J. H. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 725–738, 2011.

[24] I. D. Schizas and A. Aduroja, "A distributed framework for dimensionality reduction and denoising," *IEEE Transactions on Signal Processing*, vol. 63, no. 23, pp. 6379–6394, 2015.

[25] S. X. Wu, H.-T. Wai, A. Scaglione, and N. A. Jacklin, "The Power-Oja method for decentralized subspace estimation/tracking," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 3524–3528.

[26] A. Wiesel and A. O. Hero, "Decomposable principal component analysis," *IEEE Transactions on Signal Processing*, vol. 57, no. 11, pp. 4369–4377, 2009.

[27] A. Bertrand and M. Moonen, "Distributed adaptive estimation of covariance matrix eigenvectors in wireless sensor networks with application to distributed pca," *Signal Processing*, vol. 104, pp. 120–135, 2014.

[28] G. H. Golub and C. F. van Loan, *Matrix computations*, 4th ed. Johns Hopkins University Press, Baltimore, MD, 2013.

[29] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Journal of mathematical analysis and applications*, vol. 106, no. 1, pp. 69–84, 1985.

[30] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, M.I.T., Boston, MA, 1984.

[31] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[32] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. John Wiley & Sons, 2004, vol. 46.

[33] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[34] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006.

[35] A. Balsubramani, S. Dasgupta, and Y. Freund, "The fast convergence of incremental PCA," in *NIPS*, 2013.

[36] S. Attallah and K. Abed-Meraim, "Fast algorithms for subspace tracking," *IEEE Signal Processing Letters*, vol. 8, no. 7, pp. 203–206, 2001.

[37] J. H. Manton, I. Y. Mareels, and S. Attallah, "An analysis of the fast subspace tracking algorithm NOja," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 2002, pp. 1101–1104.

[38] H. Raja and W. U. Bajwa, "Cloud k-svd: A collaborative dictionary learning algorithm for big, distributed data," *IEEE Transactions on Signal Processing*, vol. 64, no. 1, pp. 173–188, 2016.

[39] H.-T. Wai, A. Scaglione, J. Lafond, and E. Moulines, "Fast and privacy preserving distributed low-rank regression," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 4451–4455.

[40] M. Hardt and E. Price, "The noisy power method: A meta algorithm with applications," in *Advances in Neural Information Processing Systems*, 2014, pp. 2861–2869.

[41] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845

[42] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[43] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[44] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky, "Rank-sparsity incoherence for matrix decomposition," *SIAM Journal on Optimization*, vol. 21, no. 2, pp. 572–596, 2011.

[45] A. Agarwal, S. Negahban, and M. J. Wainwright, "Noisy matrix decomposition via convex relaxation: Optimal rates in high dimensions," *The Annals of Statistics*, pp. 1171–1197, 2012.

[46] Y. Kopsinis, S. Chouvardas, and S. Theodoridis, "Distributed robust subspace tracking," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*. IEEE, 2015, pp. 2531–2535.

[47] M. Rahmani and G. K. Atia, "A decentralized approach to robust subspace recovery," in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*. IEEE, 2015, pp. 802–807.

[48] M. Mardani, G. Mateos, and G. B. Giannakis, "Decentralized sparsity-regularized rank minimization: Algorithms and applications," *IEEE Transactions on Signal Processing*, vol. 61, no. 21, pp. 5374–5388, 2013.

[49] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3/4, pp. 321–377, 1936.

[50] N. M. Correa, T. Adali, Y.-O. Li, and V. D. Calhoun, "Canonical correlation analysis for data fusion and group inferences," *IEEE signal processing magazine*, vol. 27, no. 4, pp. 39–50, 2010.

[51] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural computation*, vol. 16, no. 12, pp. 2639–2664, 2004.

[52] P. S. Dhillon, D. P. Foster, and L. H. Ungar, "Eigenwords: spectral word embeddings." *Journal of Machine Learning Research*, vol. 16, pp. 3035–3078, 2015.

[53] H. Avron, C. Boutsidis, S. Toledo, and A. Zouzias, "Efficient dimensionality reduction for canonical correlation analysis," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. S111–S131, 2014.

[54] R. Ge, C. Jin, P. Netrapalli, A. Sidford *et al.*, "Efficient algorithms for large-scale generalized eigenvector computation and canonical correlation analysis," in *International Conference on Machine Learning*, 2016, pp. 2741–2750.

[55] X. Fu, K. Huang, M. Hong, N. D. Sidiropoulos, and A. M.-C. So, "Scalable and flexible multiview max-var canonical correlation analysis," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4150–4165, 2017.

[56] A. Bertrand and M. Moonen, "Distributed canonical correlation analysis in wireless sensor networks with application to distributed blind source separation," *IEEE Transactions on Signal Processing*, vol. 63, no. 18, pp. 4800–4813, 2015.

[57] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

FOR REVIEW ONLY

[58] Q. Zhang and B. Li, "Discriminative K-SVD for dictionary learning in face recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 2691–2698.

[59] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 689–696.

[60] P. Chainais and C. Richard, "Learning a common dictionary over a sensor network," in *IEEE 5th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 2013, pp. 133–136.

[61] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary learning over distributed models," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1001–1016, 2015.

[62] H.-T. Wai, T.-H. Chang, and A. Scaglione, "A consensus-based decentralized algorithm for non-convex optimization with application to dictionary learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 3546–3550.

[63] M. Hong, D. Hajinezhad, and M.-M. Zhao, "Prox-PDA: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks," in *International Conference on Machine Learning*, 2017, pp. 1529–1538.

[64] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, p. 717, 2009.

[65] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *ICML (1)*, 2013, pp. 427–435.

[66] H.-T. Wai, J. Lafond, A. Scaglione, and E. Moulines, "Decentralized Frank–Wolfe algorithm for convex and nonconvex problems," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5522–5537, 2017.

[67] W. Zheng, A. Bellet, and P. Gallinari, "A distributed Frank-Wolfe framework for learning low-rank matrices with the trace norm," *arXiv preprint arXiv:1712.07495*, 2017.

[68] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *Journal of machine learning research*, vol. 11, no. Aug, pp. 2287–2322, 2010.

[69] E. Richard and A. Montanari, "A statistical model for tensor PCA," in *Advances in Neural Information Processing Systems*, 2014, pp. 2897–2905.

[70] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[71] A. L. De Almeida and A. Y. Kibangou, "Distributed large-scale tensor decomposition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 26–30.

[72] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 316–324.

[73] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[74] O. Shamir, "A stochastic PCA and SVD algorithm with an exponential convergence rate," in *ICML*, 2015.

[75] Z. Allen-Zhu and Y. Li, "LazySVD: Even faster SVD decomposition yet without agonizing pain," in *Advances in Neural Information Processing Systems*, 2016, pp. 974–982.

[76] C. De Sa, B. He, I. Mitliagkas, C. Ré, and P. Xu, "Accelerated stochastic power iteration," *arXiv preprint arXiv:1707.02670*, 2017.

REVIEW COPY

Distributed Principal Component Analysis

Sissi Xiaoxiao Wu, Hoi-To Wai Lin Li and Anna Scaglione

Abstract

Principal Component Analysis (PCA) is a fundamental primitive of many data analysis, array processing and machine learning methods. In applications where extremely large arrays of data are involved, particularly in distributed data acquisition systems, distributed PCA algorithms can harness local communications and network connectivity to overcome the need of communicating and accessing the entire array locally. A key feature of distributed PCA algorithm is that they defy the conventional notion that the first step towards computing the principal vectors is to form a sample covariance. This paper is a survey of the methodologies to perform distributed PCA on different datasets, their performance and of their applications in the context of distributed data acquisition systems.

REVIEW COPY

I. Overview

Distributed algorithms have a long history. In recent years, they have gained prominence in light of the end of Moore's law scaling and the seemingly exponential growth in data to analyze, due to the latest incarnation of networked technologies from social mobile media to the Internet of Things. This backdrop has sparked significant advances over the last decades on multi-agent signal processing algorithms. In contrast to their centralized counterparts, these algorithms requires the participating agents, *i.e.,* nodes in the network, to make use of their local processing power and the ability to communicate with each other by message passing, with the goal of tackling a common optimization problem.

In this context, a pervasive primitive in machine learning and sensor array processing is the computation of the *principal components* from the covariance matrix of several data streams. The goal of this paper is to survey *distributed algorithms* in Principal Component Analysis (PCA), which has wide applications in areas of communications, network, data mining and machine

S. X. Wu is with the Department of Communication and Information Engineering, Shenzhen University, China. H.-T. Wai and A. Scaglione are with the Ira A. Fulton School of Electrical Computer and Energy Engineering in Arizona State University, United States. Lin Li is with Lincoln Laboratory, Massachusetts Institute of Technology Human Language Technology Group, United States. E-mail: xxwu.eesissi@szu.edu.cn, {htwai, Anna.Scaglione}@asu.edu, lin.li@ll.mit.edu

learning [1]–[3]. Generally speaking, PCA is a statistical procedure that convert a set of high dimensional samples into a set of features that represent the data in a lower dimensional space spanned by the principal components. Since its first appearance in the seminal 1901 paper by Karl Pearson [4], PCA has evolved into various forms that fit different applications. The distributed PCA algorithms we review in this paper leave a residual relative to the original data set that is minimum in the least-square sense, *i.e.,* with the minimum Euclidean norm. This is the most common version of PCA employed in signal processing and data science, and it is associated with a number of tools in linear algebra such as the Karhunen-Loéve transform (KLT), the singular value decomposition (SVD), the eigenvalue decomposition (EVD), the Orthogonal-triangular (QR) factorization, etc.

The solution found by the PCA problem(s) is widely applied in dimensionality reduction and to cluster large amount of data into groups via spectral clustering [5], to classify word documents [6], for beam-forming in array processing [1], [2]. We refer the reader to [3] for a comprehensive survey of PCA applications.

The main goal of this review is to explain how distributed storage and computation systems can implement PCA. Next, we classify these algorithms based on the way the data are partitioned and the communications are structured in the network.

REVIEW COPY

A. *The family of distributed PCA algorithms*

The distributed PCA algorithms in the literature can be broadly divided into different classes, and they are often designed according to: 1) how the data are divided in the network; 2) how the communication and computations among the different agents (or workers) are structured, namely a hierarchical architecture versus a totally flat architecture based on message passing.

1) Data partitioning: The design of distributed PCA algorithms depend heavily on the way that data are partitioned and stored in the agents on the network. Below, we survey two of the main classes of data partitions.

In the first class of data partition, each agent has access to a different subset of samples of the data set. This type of partition occurs in applications where a large amount of high dimensional data is stored across different sites in a network. In this case, distributed PCA allows to learn the important features from these high dimensional data that help compress, summarize, classify or rank them, without sharing the data directly. A relevant instance for this setting occurs in document classification [cf. Fig. 1], where the data dimensions are the frequency of a specific

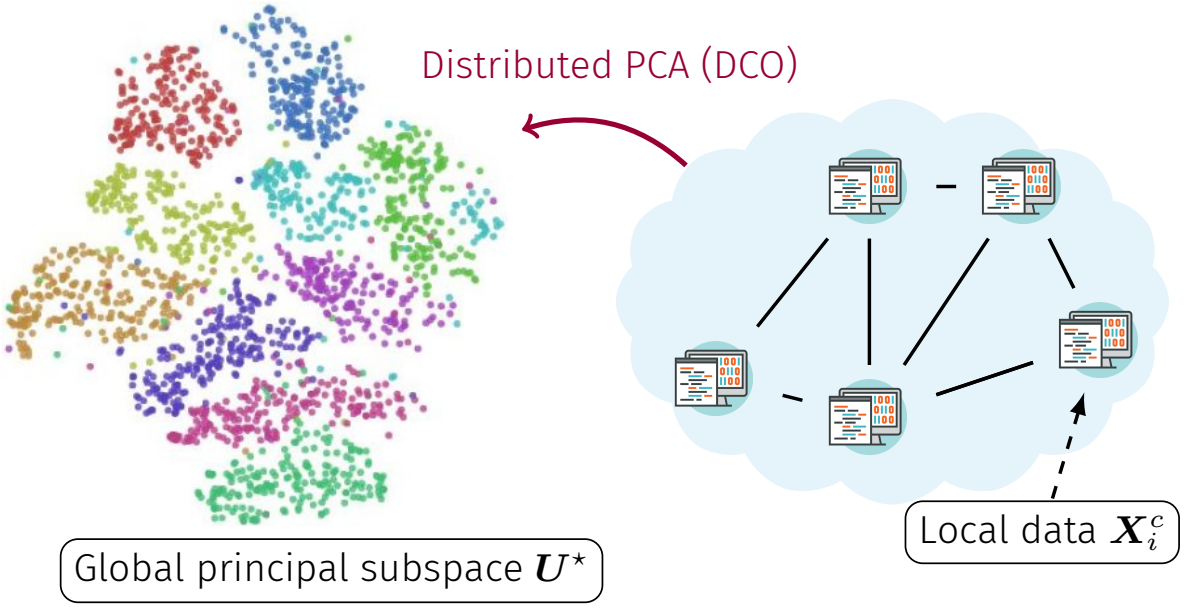

Fig. 1. **Example of Distributed PCA with DCO data**. In this illustration, each agent is a computer server gathering data locally from the users that it serves, organized through partitions by the columns.

REVIEW COPY

word in the document, and each sample corresponds to a different document. As the documents are scattered across various servers, each agent in the network only possesses a subset of the outcomes. Applying distributed PCA in this case provides a way to de-noise the unlabeled data (that can be interpreted as topics) by retaining only the useful subspaces and that can then be used to classify the documents.

Prior work on this class of the distributed PCA algorithms is prevalent in the machine learning community, e.g., [7]–[17]. Specifically, [7], [14] proposed to compute a set of local principal component vectors at the agents and then have a central coordinator fuse these results into the desired order principal sub-space; the performance of this approach (and its variants) have been studied in e.g. [8]–[11]. In [12], a multi-round distributed PCA algorithm was proposed to better balance communication and computation costs. In [13], the principal components are found by performing a sequence of local QR decomposition at the agents. In [15], the principal components are computed using the aggregated eigenvectors and eigenvalues of the local covariance matrix. It is worthwhile to point out that except for [17], the work cited above considers a star network architecture, with a hub that fuses the results of several servers iteratively. This is different from the setting where the servers communicate through a meshed network, which is the primary

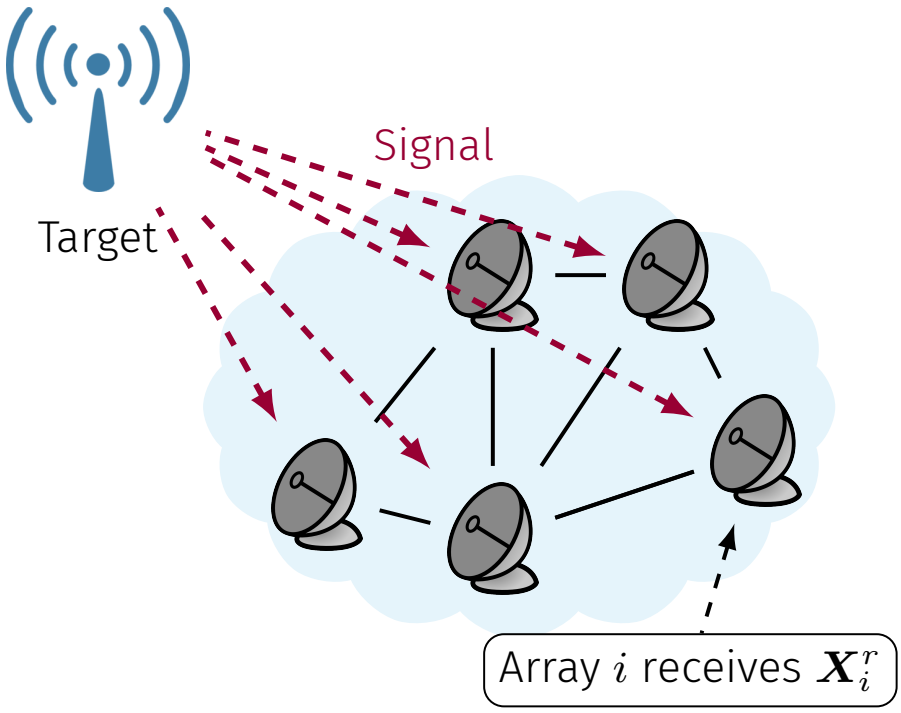

Fig. 2. **Example of Distributed PCA with DRO data.** In this example, each array receives its own copy of signal which is partitioned by rows and all the arrays estimate the subspace of received signals for target tracking.

REVIEW COPY

focus of this paper review in Section IV.

The second class of data partition corresponds to having a multi-dimensional time series and having the entries of each sample distributed across the agents, on an entry by entry (or a block by block) basis. This situation naturally arises in distributed sensors deployments where the sensors collect samples simultaneously of a continuous field that evolves in time and space. For example, the field could the signal emitted or backscattered by a moving target [cf. Fig. 2]. What motivates PCA is the underlying assumption that the field has few active signal components, therefore spanning an unknown but low dimensional signal subspace. Each sensor collects one (or a subset) of the projections of such vector field at a given time, *i.e.,* a *spatial* sample of the field.

This class of PCA problems has inspired a number of distributed PCA methods developed by the signal processing community [18]–[27] which we review in Section V. For instance, [18]–[20] considered extending the classical power method [28] to the distributed setting. The proposed distributed power method was later analyzed in [21] and extended to an asynchronous

setting in [22]. Note that the power method is a batch processing method with fast convergence, yet the method is non-adaptive and has high latency. Since the observations are evolving with time, it is natural to seek adaptive solutions; in the centralized setting this is achieved by the Oja's method [29], and the distributed setting can be found in [23], [25] which developed the decentralized subspace tracking algorithms. Interestingly, these class of algorithms are all for meshed networks.

Before we delve into the mathematical details in Section II, we shall fix some terminology by describing how the data samples are arranged. In particular, we define a data matrix which takes each sample of data as a column. In this way, the two main situations described above correspond, respectively, to what we shall refer as the Distributed Columns Observations (DCO) and the Distributed Rows Observations (DRO), respectively. In both cases, the PCA problem is aimed at retrieving the principal subspace for the *column span* of the data matrix, yet in the DRO case each node computes only a subset of the coordinates (one entry or one block) of the principal subspace vectors, while in the DCO case all nodes achieve consensus on the entire principal subspace basis. The way the knowledge about the principal subspace is shared in the DCO and DRO cases affects how the distributed PCA primitive can be used for different applications.

2) Communication and computation architecture: The design of distributed PCA algorithms also differ in terms of the type of communications they require: one variant uses a hierarchical master-slave type architecture and one uses a flat message passing architecture [cf. Fig. 3].

The master-slave approach for distributed PCA conforms to a hierarchical division of tasks, and employs a central coordinator that also acts as a fusion node [7]–[16]. In this case, the agents form a *star network* topology, where the computation tasks of the master node at the center are different from those of the servers. These models conform to the typical architecture for parallel computation in multi-core processors, where the aim of applying distributed PCA algorithms is to accelerate PCA computation by utilizing local computations and also local memory resources. It is also worthwhile to point out that the algorithms above fall into the category of the DCO type distributed PCA.

The algorithms developed for this architecture usually consists of two stages — a local stage and a global stage. In the local stage, each agent/slave performs its own local optimization, e.g., by solving a local PCA problem, and send the results to the central coordinator. In the global stage, the central coordinator then computes the global PCA from the aggregated data.

REVIEW COPY

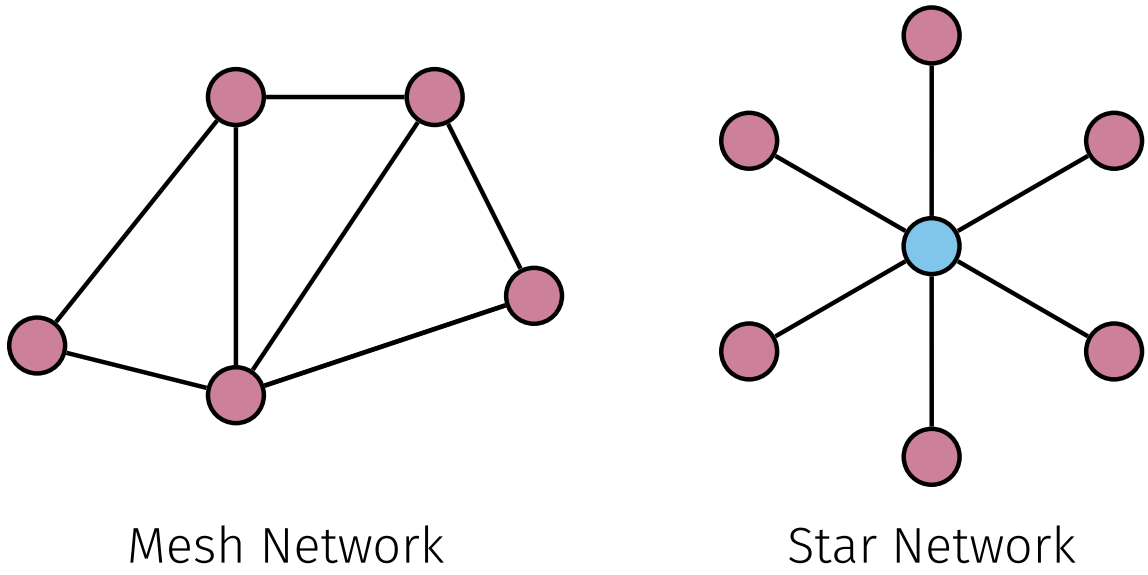

Fig. 3. **Mesh and Star Networks**. For star networks, the nodes are divided into slave nodes and a master node, where the latter receives data from all the slave nodes.

REVIEW COPY

The second class of distributed PCA algorithms works on arbitrarily meshed networks and conforms to the type of parallel processing that is performed in Graphic Processing Units (GPUs), or distributed storage systems. They have the following features: 1) all nodes and links must perform the same function and run the same procedures; 2) the nodes exchange partial computations but not the data, thus privacy is respected. This architecture can be seen as a generalization of the master-slave one, where the agents form a topology described by a general graph, and it arises from applications involving computer or wireless sensor networks. In particular, while the agents are still connected to each other, transmitting information from one agent to another may require multi-hop communications. In addition to accelerating PCA computations, this class of algorithms aims at offering better resiliencies in hostile environments such as random failures of agents. Examples of these algorithms can be found in [18]–[25]. To adapt to the mesh network setting, these algorithms are often developed by re-interpreting classical numerical methods for PCA such as power and Oja's method as a sequence of computation steps for averaging a set of local values across the agents. The distributed averaging can then be realized by resorting to instantiations of the *average consensus (AC)* subroutine; see [30], [31]. Their accuracy rests on the performance of the AC subroutine, which is known to converge exponentially at rate that

depends on the algebraic connectivity of the communication network; see Section III-A.

We observe that the majority of distributed PCA algorithms belong to either one of the two types — (i) DCO data on master-slave architectures and (ii) DRO data on arbitrary mesh networks. Such dichotomy is due to the different types of applications that are prevalent in the machine learning and signal processing communities respectively. Throughout this paper, we focus on surveying algorithms for the settings with *arbitrarily meshed networks*, though some of the representative work for the master-slave architecture will also be summarized in Section IV-A. Furthermore, we shall emphasize that the tailor-made master-slave distributed PCA algorithms can be more effective in handling specific *big-data* problems, and they are preferable in contexts where one can choose to build such an architecture.

Notation. We use boldfaced lower-case letters (e.g. $\boldsymbol{x}$) to denote vectors and boldfaced upper-case letters (e.g. $\boldsymbol{X}$) to denote matrices. While this convention is prevalent throughout, occasionally roman capitals $\mathbf{X}$ will be used to denote the random vectors whose outcomes are denoted with the lower-case boldface notation $\boldsymbol{x}$ used for deterministic vectors. For a square matrix $\boldsymbol{R}$, $\lambda_i(\boldsymbol{R})$ denotes its ith largest eigenvalue; while for a rectangular matrix $\boldsymbol{X}$, $\sigma_i(\boldsymbol{X})$ is its ith largest singular value. The operator $(\cdot)^{\mathsf{H}}$ (*resp.* $(\cdot)^{\top}$) denotes the Hermitian transpose (*resp.* the standard transpose). Unless otherwise specified, $\|\boldsymbol{x}\|$ is the standard Euclidean norm for the vector $\boldsymbol{x}$.

B. Paper organization

In the next section we formulate mathematically the distributed PCA problem and its two instantiations, for the DRO and DCO scenarios, respectively. Then, in Section III, we provide background on the building blocks that are at the basis of the distributed PCA algorithms: specifically, the AC subroutine, the power method and the Oja's method. Section IV and V review the distributed PCA methods for the DRO setting and for the DCO setting, respectively; within each of them, some salient applications are also reviewed. Possible extensions of the work cited and the related problems are presented in Section VI, which is followed by the conclusions in Section VII.

II. The PCA Problem & Its Distributed Forms

We consider a set of observations given as an $N \times T$ (potentially complex) matrix $\boldsymbol{X}$, *i.e.,*

$$\boldsymbol{X} := \big(\boldsymbol{x}(1)\ \boldsymbol{x}(2)\ \cdots\ \boldsymbol{x}(T)\big) \in \mathbb{C}^{N\times T}\,, \tag{1}$$

REVIEW COPY

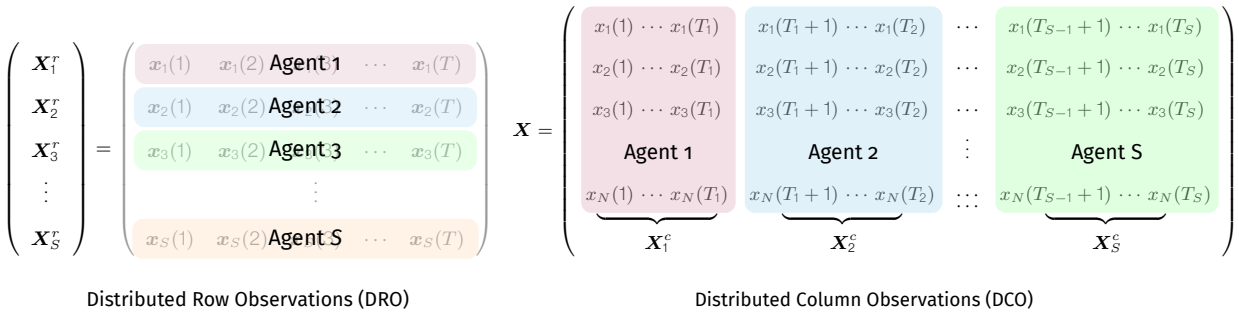

Fig. 4. **Types of Data Partition employed in Distributed PCA.** Each data partition type shall involve a different solution technique for distributed computation of the PCA.

The rows and columns of $\boldsymbol{X}$ represent the feature/spatial dimension and the sample/time dimension, respectively. The singular value decomposition (SVD) of $\boldsymbol{X}$ is denoted by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{H}}$ where $\boldsymbol{U}, \boldsymbol{V}$ are unitary matrices and $\boldsymbol{D} = \mathrm{Diag}([\sigma_1; \ldots; \sigma_R])$ is a diagonal matrix of the singular values with $R = \mathrm{rank}(\boldsymbol{X})$. We make the canonical assumption that all the singular values have multiplicity of one, *i.e.,* $\sigma_1 > \ldots > \sigma_R > 0$[1].

In many practical scenarios, the data samples $\boldsymbol{x}(t)$ are correlated with each other. When the dataset is large with $N, T \gg 1$ an option to reduce the size of the data is to project $\boldsymbol{X}$ onto its p-dimensional (p-D) principal subspace, which can be represented by an orthogonal transformation $\boldsymbol{U}_p$ — a sub-matrix of the unitary matrix $\boldsymbol{U}$ consisting of only its left p column vectors. In particular, given the orthogonal projection vector $\boldsymbol{z}(t) = \boldsymbol{U}_p^{\mathsf{H}}\boldsymbol{x}(t) \in \mathbb{C}^p$ the corresponding low-dimensional approximation of $\boldsymbol{x}(t)$ is given by $\hat{\boldsymbol{x}}(t) = \boldsymbol{U}_p\boldsymbol{z}(t)$. It is also convenient to write the tth sample as:

$$\boldsymbol{x}(t) = \hat{\boldsymbol{x}}(t) + \boldsymbol{e}(t) = \boldsymbol{U}_p\boldsymbol{z}(t) + \boldsymbol{e}(t) \ , \tag{2}$$

where $\boldsymbol{e}(t)$ represents the modeling error. PCA is effective when the Frobenious norm of $\boldsymbol{e}(t)$ is small. Based on (2), the PCA problem amounts to learning the orthogonal transformation $\boldsymbol{U}_p$ from the data $\boldsymbol{X}$ that would allow a *lossy* mapping of $\boldsymbol{x}(t) \mapsto \boldsymbol{z}(t)$. Under the standard assumption that $T \geq p$, the PCA problem can be represented as the following optimization:

$$\begin{aligned} \boldsymbol{U}^\star \in \arg \min_{\boldsymbol{U} \in \mathbb{C}^{N \times p}} \quad & \|(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{X}\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} \ . \end{aligned} \tag{3}$$

[1]This assumption can be relaxed, e.g., when one is interested only finding the top p principal subspaces, then only $\sigma_1 \geq \cdots \geq \sigma_p > \sigma_{p+1} \geq \cdots \geq \sigma_R \geq 0$ is needed for the algorithms in this paper.

REVIEW COPY

As seen, the solution of (3) minimizes the residual for the signal reconstructed from $\{\boldsymbol{z}(t)\}_{t=1}^{T}$ in the mean square sense. The problem may also be extended to a stochastic and dynamic setting as follows:

$$\begin{aligned} \boldsymbol{U}^{\star}(t) \in \arg \min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \quad & \mathbb{E}\big[\|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\mathbf{X}(t)\|^2\big] \\ \text{s.t.} \quad & \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I}\ , \end{aligned} \tag{4}$$

where we have denoted $\mathbf{X}(t) \in \mathbb{C}^N$ as a vector-valued random process and the expectation is taken with respect to the distribution of $\mathbf{X}(t)$. Note that the realization of $\mathbf{X}(t)$ is denoted as $\boldsymbol{x}(t)$. To account for the possible non-stationarity in $\mathbf{X}(t)$, the solution of (4) depends on the sample/time index t. We shall refer to (3) as the *batch/static* PCA problem, and (4) as the *dynamic/stochastic* PCA problem. Note that the solution to the PCA problems is intrinsically ambiguous subject to rotations, as we observe by first defining the equivalence class $[\boldsymbol{U}]$ with $\boldsymbol{U} \in \mathbb{C}^{N\times p}$:

$$[\boldsymbol{U}] := \{\hat{\boldsymbol{U}} \in \mathbb{C}^{N\times p} |\ \hat{\boldsymbol{U}} = \boldsymbol{U}\boldsymbol{Q},\ \boldsymbol{Q} \in \mathbb{C}^{p\times p} \text{ is unitary}\}\ . \tag{5}$$

Importantly, any matrix in $[\boldsymbol{U}^{\star}]$ (*resp.* $[\boldsymbol{U}^{\star}(t)]$) will also be an optimal solution to (3) (*resp.* (4)).

In the *batch* PCA setting, a *centralized* PCA algorithm [32] solves (3) by performing an SVD on $\boldsymbol{X}$, where we can simply set the optimal solution as $\boldsymbol{U}^{\star} = \boldsymbol{S}_p$. Note that this gives the optimal objective value of $\sum_{r=p+1}^{R} \sigma_r^2$. In a *distributed* or *dynamic* setting, the entries of $\boldsymbol{X}$, instead of being processed at a central machine, are scattered on S different machines/sensors, which we shall refer to as *agents* later on. We further provide the following definitions to distinguish the types of distributed algorithms:

- **Distributed Columns Observations (DCO)**: The DCO setting assumes that each agent observes a subset of *columns* of $\boldsymbol{X}$. We partition $\boldsymbol{X}$ by its columns such that

$$\boldsymbol{X} = (\boldsymbol{X}_1^c\ \boldsymbol{X}_2^c\ \cdots\ \boldsymbol{X}_S^c)\ , \tag{6}$$

  where $\boldsymbol{X}_i^c \in \mathbb{R}^{N\times T_i}$ is the column-partitioned sub-matrix kept by agent i and $T = \sum_{i=1}^{S} T_i$. Alternatively, the data available at agent i can also be represented as $\boldsymbol{x}(t) \in \mathbb{C}^N$ for $t \in \mathcal{T}_i$ with $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_S = \{1, ..., T\}$, $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$, and $|\mathcal{T}_i| = T_i$.
- **Distributed Rows Observations (DRO)**: The DRO setting assumes that each agent observes only a subset of *rows* of $\boldsymbol{X}$. We partition $\boldsymbol{X}$ by its rows such that

$$\boldsymbol{X} = ((\boldsymbol{X}_1^r)^{\top}\ (\boldsymbol{X}_2^r)^{\top}\ \cdots\ (\boldsymbol{X}_S^r)^{\top})^{\top}\ , \tag{7}$$

REVIEW COPY

where agent i keeps the row-partitioned sub-matrix $\boldsymbol{X}_i^r \in \mathbb{R}^{N_i \times T}$ with $N = \sum_{i=1}^{S} N_i$. Alternatively, the data available at agent i can also be represented as $\boldsymbol{x}_i(t) \in \mathbb{C}^{N_i}$ for $t = 1, ..., T$.

See Fig. 4 for an illustration on the types of data structure considered. A key feature that distinguishes distributed PCA methods from its centralized counterpart is that the agents have to solve (3) by *cooperating* with their neighbors in the network, given that each agent has only access to the partial observation matrix. Based on the settings we just discussed, next we describe the goals of the distributed PCA methods.

The *DCO* setting is encountered primarily in *big-data* mining applications. Each agent in this case can be a computer server that gathers data samples from a set of users that it is serving. For instance, the ith agent obtains samples $\{\boldsymbol{x}(t)\}_{t\in\mathcal{T}_i}$ from a group of users. In this case, our goal is:

> *Goal (DCO):* Agent i learns a common p-D principal subspace of $\boldsymbol{X}$ shared by the other agents, *i.e.,* to learn $\boldsymbol{U}^\star$, in a distributed fashion.

Notice that the goal here is similar to solving a *consensus* problem requiring the agents to agree with each other. Using the data structure in the DCO setting, we observe that (3) can be written as:

$$\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \sum_{i=1}^{S} \underbrace{\|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{X}_i^c\|_{\mathrm{F}}^2}_{:=f_i(\boldsymbol{U})} \;\text{s.t.}\; \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I}\,, \tag{8}$$

which has a separable objective function similar to the consensus optimization problem tackled in [33]. In other words, our aim is to find a *common* dictionary based on all the data accrued across the network. Furthermore, we remark that the DCO data structure is only relevant in the static/batch PCA setting.

As mentioned before, the *DRO* setting is typically encountered in sensor networks where each agent, depending on its siting, captures components of the vector field that corresponds to the block $\boldsymbol{x}_i(t)$. For instance, the ith agent may have access to the ith antenna that received the signal $x_i(t)$. In this case, we define our goal as:

> *Goal (DRO):* Agent i learns the ith partition of the p-D principal subspace of $\boldsymbol{X}$, *i.e.,* to learn $\boldsymbol{U}_i^{r,\star}$ in the partition $\boldsymbol{U}^\star = (\boldsymbol{U}_1^{r,\star}; \ldots; \boldsymbol{U}_S^{r,\star})$, in a distributed fashion.

This is a reasonable setting as $\boldsymbol{U}_i^{r,\star}$ keeps the components of $\boldsymbol{U}^\star$ that are related to the observations made at the ith agent. A typical example for DRO is the subspace estimation and tracking

REVIEW COPY

in a sensor or radar network. The main setting is an information fusion, which consists of many nodes, that could be sensor nodes, arrays, radars. Each node can be seen as a one-dimension unit for receiving the signal, aiming at cooperatively estimating and tracking the principal subspace of the signals. Once the nodes have retrieved the PCA, the distributed projection on the principal subspace can also be distributed and all nodes will achieve a consensus on the coordinates of the PCA approximation $\boldsymbol{z}(t)$.

The important observation here is that in the DRO setting the PCA problem in not separable like it is for the DCO setting, as seen in (8), and there is no consensus condition that ties the results the agents obtain.

Prior to reviewing the DCO and DRO algorithms in Sections IV and V respectively, in the next section we briefly introduce some of the building blocks these algorithms rely on.

REVIEW COPY

III. Building Blocks

In this section, we describe the important algorithms for network consensus and PCA, which will then serve as the building blocks for the distributed PCA methods surveyed.

A. Average Consensus Algorithm (a.k.a. Gossip Algorithm)

In this section we provide a brief introduction to the so called *average consensus algorithm* (also known as the gossip algorithm) [30], [31], which is key computation method used in distributed PCA methods. As its name has suggested, the average consensus algorithm describes a procedure for computing the *average* of a set of values stored at the agents through *local information exchanges*.

To proceed, it is necessary to define some terminology to refer to the communication network that connects the agents. As explained in the Introduction, we focus on the setting where the agents are connected on an arbitrarily *meshed network*. Specifically, the network used can be mapped onto a connected, undirected and simple graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., S\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of communication links between the agents. See Fig. 5 for examples of a few common models to generate graph topologies.

Suppose that the ith agent holds a certain vector/matrix $\boldsymbol{Y}_i$ and he/she wishes to compute the global *average* $\boldsymbol{Y}_{\text{avg}} := S^{-1} \sum_{i=1}^{S} \boldsymbol{Y}_i$. A distributed algorithm suitable for the task is the classical average consensus algorithm [30], [31], [34], also known as the gossip algorithm. In particular, for all $\ell \geq 1$, let $\boldsymbol{W}[\ell]$ be a symmetric and doubly stochastic matrix (i.e. such that $\boldsymbol{W}[\ell]\mathbf{1} = \mathbf{1}$

and $\mathbf{1}^T\boldsymbol{W}[\ell] = \mathbf{1}^T$) whose sparsity matches that of adjacency matrix for the graph G, that is $W_{ij}[\ell] = W_{ji}[\ell] > 0$ only if $(i,j) \in \mathcal{E}$, and $\boldsymbol{W}[\ell]\mathbf{1} = \boldsymbol{W}^\top[\ell]\mathbf{1} = \mathbf{1}$. We can compute $\boldsymbol{Y}_{\text{avg}}$ distributively as follows:

Average Consensus (AC) algorithm:

1) Initialize as $\boldsymbol{Z}_i[0] = \boldsymbol{Y}_i$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:

$$\boldsymbol{Z}_i[\ell] = \sum_{j=1}^{S} W_{ij}[\ell]\boldsymbol{Z}[\ell-1] \,, \tag{9}$$

for all $\ell \geq 1$ and we terminate after $\ell \geq L$. The collection $\{\boldsymbol{Z}_i[L]\}_{i=1}^S$ is retrieved as the output of the AC subroutine.

For simplicity, let us denote $\boldsymbol{Z}_i[L]$, *i.e.,* the variable after the Lth update, as output of the above subroutine:

$$\{\boldsymbol{Z}_i[L]\}_{i=1}^S := \mathsf{AC}\big(\{\boldsymbol{Y}_i\}_{i=1}^S; L\big) \,, \tag{10}$$

where the first argument to the subroutine are the initialization given to the network and the second argument specifies the number of average consensus updates required. We also use $\boldsymbol{Z}_i[L] := \mathsf{AC}_i\big(\{\boldsymbol{Y}_i\}_{i=1}^S; L\big)$ to denote the output of the AC subroutine stored at the ith agent as indicated by the subscript. We remark that the AC subroutine defined in the above is applicable for computing the averages of scalar, vector and matrices.

Above, we described the most general case where $\boldsymbol{W}[\ell]$ changes over time, e.g., it models the scenario when some links in the network may be inactive at times. Formally, for every $\ell \geq 1$, this matrix is drawn from a distribution with $\mathbb{E}[\boldsymbol{W}[\ell]] = \overline{\boldsymbol{W}}$ with

$$\lambda_{\text{conn.}} := \max\{\lambda_2(\overline{\boldsymbol{W}}), -\lambda_S(\overline{\boldsymbol{W}})\} < 1 \,. \tag{11}$$

An example is represented by the *pairwise gossiping protocol* introduced by [34], where an edge (i_ℓ, j_ℓ) is selected from $\mathcal{E}$ uniformly at random and the agents compute a convex combination of their current state values which then becomes their new state. In this case:

$$\boldsymbol{W}[\ell] = \boldsymbol{I} - \frac{1}{2}(\boldsymbol{e}_{i_\ell} - \boldsymbol{e}_{j_\ell})(\boldsymbol{e}_{i_\ell} - \boldsymbol{e}_{j_\ell})^\top \tag{12}$$

and the condition (11) will be satisfied as long as G is a connected graph.

An important feature of the average consensus algorithm is that its convergence is exponentially fast, *i.e.,* we have $\mathbb{E}[\|\boldsymbol{Z}_i[L] - \boldsymbol{Y}_{\text{avg}}\|] = \mathcal{O}(\lambda_{\text{conn.}}^L)$ for all i where the expectation is taken

REVIEW COPY

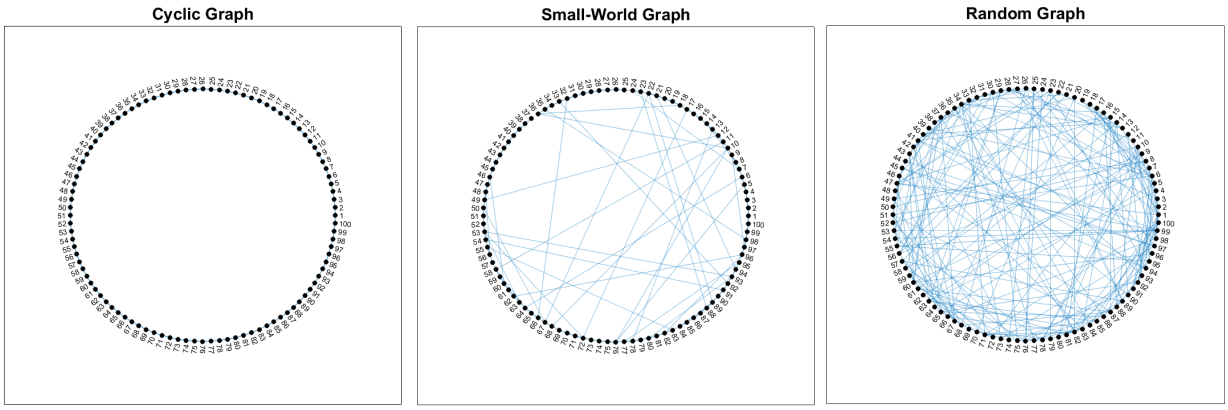

Fig. 5. **Examples of graph topology.** Left: a cyclic graph with $S = 100$ nodes and node degree 4; Middle: a small-world graph with $S = 100$, average degree 4 and re-wiring probability 0.2. Right: a random graph with $S = 100$ nodes and node degree 4.

REVIEW COPY

with respect to the realizations of $\boldsymbol{W}[\ell]$. To compute an ϵ-accurate average, one only requires $\Theta(\log \epsilon^{-1})$ average consensus updates.

B. Power Method for PCA

Motivated by the fact that an optimal solution to the *static* PCA problem (3) can be obtained from evaluating the top (left) singular vectors of the data matrix $\boldsymbol{X}$ (or equivalently the top eigenvectors of the auto-correlation $\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}}$), a natural idea is to apply the power method for solving the PCA problem, which is a well known numerical method [28] for computing the top eigenvectors of a symmetric matrix. In light of this, here we briefly review the power method and provide insights that will be instrumental to the development of distributed PCA methods. Consider the PCA problem (3) and observe that an optimal solution can be found by retrieving the top-p eigenvectors corresponding to the largest p eigenvalues of the sampled correlation matrix:

$$\hat{\boldsymbol{R}}_x := \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\ . \tag{13}$$

Note that $\boldsymbol{x}(t)$ is zero mean, then $\hat{\boldsymbol{R}}_x$ is also the sampled covariance. Alternatively, one could remove the mean in the data by a simple pre-processing step. To compute the *top* eigenvector of $\hat{\boldsymbol{R}}_x$, the power method is initialized by $\boldsymbol{u}_1[1] \sim \mathcal{CN}(0, \boldsymbol{I})$, and it adopts the following recursion:

$$\overline{\boldsymbol{u}}_1[k] = \frac{\boldsymbol{u}_1[k]}{\|\boldsymbol{u}_1[k]\|},\ \boldsymbol{u}_1[k+1] = \hat{\boldsymbol{R}}_x\overline{\boldsymbol{u}}_1[k],\ \forall\ k \geq 1\ . \tag{14}$$

It can be shown that $\overline{\boldsymbol{u}}_1[k]$ converges to $\boldsymbol{u}_1^\star$ as $k \to \infty$, where $\boldsymbol{u}_1^\star$ is the *top* eigenvector of $\hat{\boldsymbol{R}}_x$.

From an optimization perspective, the power method can be seen as a *fixed-point iteration* method which solves the nonlinear system arising from the optimality condition of (3). Its rate of convergence is exponential: it computes an ϵ-accurate eigenvector with $k = \Omega((\log(\sigma_1(\hat{\boldsymbol{R}}_x)/\sigma_2(\hat{\boldsymbol{R}}_x)))^{-1}\log(1/\epsilon)$ power iterations [28][2]. Note that the ratio $\sigma_1(\hat{\boldsymbol{R}}_x)/\sigma_2(\hat{\boldsymbol{R}}_x)$ is known as the *spectral gap* of $\hat{\boldsymbol{R}}_x$ and it is a key factor determining the convergence speed of the power method. Moreover, as we shall reveal later, the computations above can be performed distributively with the help of the average consensus subroutine, by exploiting the relationship of $\hat{\boldsymbol{R}}_x$ with the data.

To find the *second* eigenvector of $\hat{\boldsymbol{R}}_x$, denoted by $\boldsymbol{u}_2^\star$, we observe that $(\boldsymbol{I} - \boldsymbol{u}_1^\star(\boldsymbol{u}_1^\star)^{\mathsf{H}})\hat{\boldsymbol{R}}_x$ is also Hermitian and positive semidefinite, and this matrix's top eigenvector is the sought $\boldsymbol{u}_2^\star$. Naturally, we can apply the same power method procedure to compute $\boldsymbol{u}_2^\star$. Repeating the same procedures we can find $\boldsymbol{u}_3^\star, ..., \boldsymbol{u}_p^\star$, to complete the principal subspace $\boldsymbol{U}^\star = (\boldsymbol{u}_1^\star, ..., \boldsymbol{u}_p^\star)$.

C. Oja-based Methods for PCA

Originally proposed by Oja *et al.* [29] in 1985, the Oja's method for PCA was developed from a different philosophy than the power method. In particular, the method focuses on tackling the optimization problem (4) using a stochastic gradient descent (SGD) method. Notice that (4) is a non-convex and stochastic optimization problem which can be difficult to handle. As a remedy, we consider the special case of $p = 1$ and the following form of (3):

$$\boldsymbol{u}^\star(t) \in \arg\max_{\boldsymbol{u}\in\mathbb{C}^N} \; f_t(\boldsymbol{u}) := \frac{\boldsymbol{u}^{\mathsf{H}}\mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)]\boldsymbol{u}}{\|\boldsymbol{u}\|^2} \;, \tag{15}$$

where the objective function is also known as the Rayleigh coefficient of the correlation matrix $\boldsymbol{R}_x(t) := \mathbb{E}[\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)]$. The following function is a stochastic approximation of $f_t(\boldsymbol{u})$:

$$\hat{f}(\boldsymbol{u}; \{t\}) := \frac{\boldsymbol{u}^{\mathsf{H}}\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}}{\|\boldsymbol{u}\|^2} \approx f_t(\boldsymbol{u}) \;, \tag{16}$$

since only one sample is used in the above, we say that the batch size used is 1. We observe the approximation is unbiased as $\mathbb{E}[\hat{f}(\boldsymbol{u}; \{t\})] = f_t(\boldsymbol{u})$. Consequently, its gradient is also unbiased since $\mathbb{E}[\nabla\hat{f}(\boldsymbol{u}; \{t\})] = \nabla f_t(\boldsymbol{u})$.

[2]In the sense that $\sqrt{1 - |\overline{\boldsymbol{u}}_1[k]^{\mathsf{H}}\boldsymbol{u}_1^\star|^2} \le \epsilon$.

REVIEW COPY

The Oja's method in [29] is essentially an SGD method for (15) with a batch size of 1, *i.e.,* let $\boldsymbol{u}^{\mathsf{Oja}}(t) \in \mathbb{C}^N$ be the estimated principal component at iteration t, we have

$$\begin{aligned} \boldsymbol{u}^{\mathsf{Oja}}(t+1) &= \boldsymbol{u}^{\mathsf{Oja}}(t) + \tilde{\gamma}_t \nabla \hat{f}(\boldsymbol{u}^{\mathsf{Oja}}(t); \{t\}) \\ &= \boldsymbol{u}^{\mathsf{Oja}}(t) + \gamma_t \Big(\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t) - \frac{|\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)|^2}{\|\boldsymbol{u}^{\mathsf{Oja}}(t)\|^2} \boldsymbol{I} \Big) \boldsymbol{u}^{\mathsf{Oja}}(t) \ , \end{aligned} \tag{17}$$

where $\gamma_t > 0$ is a step size and $\gamma_t := 2\tilde{\gamma}_t / \|\boldsymbol{u}^{\mathsf{Oja}}(t)\|^2$. It is worth noting that, due to the non-convex nature of (15), the global convergence of Oja's learning rule has remained elusive. In fact, most of the available results are focused on the special cases with stationary $\boldsymbol{x}(t)$. For example, the authors in [35] proved that when $p = 1$ and $\gamma_t = c/t$, then (17) returns the top eigenvector of $\boldsymbol{R}_x$ as $t \to \infty$ at a sub-linear rate of $\mathcal{O}(1/t)$; [29, Theorem 2] proved if $\sum_t \gamma_t = \infty$, $\sum_t \gamma_t^2 < \infty$, then (17) converges almost surely to the principal subspace, yet the convergence rate is not given. Nevertheless, the Oja's learning rule is often used even when the process $\boldsymbol{x}(t)$ is non-stationary. To avoid getting stuck at a solution equal to the previous subspaces, an effective heuristic is to set γ_t to be a small *constant* such that the newly observed samples are sufficiently represented.

Various forms of Oja-based method have also been proposed. In general, their philosophy is to apply different relaxations to the PCA problem (4) and then to apply SGD on the relaxed problems. Examples are NOja and NOOja studied in [36]. Their convergence has been studied in [37].

IV. Distributed PCA Methods for DCO

In this section we survey some of the representative algorithms for distributed PCA in the DCO setting. As mentioned before, a distinguishing feature of the DCO scenario is that each agent keeps a (sub)set of the observed samples $\{\boldsymbol{x}(t)\}_{t \in \mathcal{T}_i}$, grouped into an $N \times T_i$ matrix $\boldsymbol{X}_i^c$. When $\boldsymbol{x}(t)$ are generated i.i.d. and T_i is large, each agent possesses sufficient data to compute the PCA locally. As such, we focus on the more restrictive case when T_i is not large and discuss the strategies in which the agents can leverage on additional information from other agents in the same network.

A. Strategies for Master-slave Architectures

A number of papers [7]–[17] tackled the distributed PCA problem (with DCO data) when the agents are organized according to a *master-slave* architecture, or equivalently a star network. As mentioned in the Introduction, these works keep the sets of data samples at the different servers

REVIEW COPY

sites and focus on the issue of reducing the *overall* computation complexity and communication cost required by the distributed methods.

As an example, Qu *et al.* [7] proposed to compute a *local PCA* from the partial data by having agent i perform an SVD of its own local subset of data matrix, deprived of their mean, *i.e.,* :

$$\boldsymbol{X}_i^c(\boldsymbol{I} - |\mathcal{T}_i|^{-1}\mathbf{1}\mathbf{1}^\top) = \boldsymbol{U}_i\boldsymbol{\Lambda}_i\boldsymbol{V}_i^{\mathsf{H}} \ . \tag{18}$$

Let $\tilde{\boldsymbol{U}}_i \in \mathbb{C}^{N\times p_i}$ be the matrix for the top p_i singular vector and $\tilde{\boldsymbol{\Lambda}}_i$ be the principal $p_i \times p_i$ submatrix of $\boldsymbol{\Lambda}$, where $p_i \geq p$. Agent i then transmits $\tilde{\boldsymbol{U}}_i\tilde{\boldsymbol{\Lambda}}_i^2\tilde{\boldsymbol{U}}_i^{\mathsf{H}}$ and the local mean $\bar{\boldsymbol{x}}_i := |\mathcal{T}_i|^{-1}\boldsymbol{X}_i^c\mathbf{1}^\top$ to the central server, which estimates the global PCA, $\hat{\boldsymbol{U}} \in \mathbb{C}^{N\times p}$, from:

$$\tilde{\boldsymbol{S}} = \sum_{i=1}^{S}\tilde{\boldsymbol{U}}_i\tilde{\boldsymbol{\Lambda}}_i^2\tilde{\boldsymbol{U}}_i^{\mathsf{H}} + \sum_{i=1}^{S}|\mathcal{T}_i|(\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}})(\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}})^{\mathsf{H}} \ . \tag{19}$$

We observe that this algorithm can be completed in one communication and computation round and its accuracy depends on the choice of p_i. On the other hand, it is clear that $\hat{\boldsymbol{U}} \neq \boldsymbol{U}^\star$ as $\tilde{\boldsymbol{S}}$ does not share the principal subspaces as $\boldsymbol{S}$ in general. In fact, the accuracy of the algorithm above can only guaranteed when the data observed are *homogeneous*, e.g., for all i, each column of $\boldsymbol{X}_i^c$ is generated from (2) with independently distributed modeling error $\boldsymbol{e}(t)$. The merits of these approaches rest on the savings in computation cost when implemented in a parallel computer system. Similar approach can be found in [8] which combines distributed PCA method with distributed K-means algorithms for further processing of the dimension reduced data.

B. Strategies for Mesh Networks

From now on we focus on the mesh networks case. As pointed out in [17], a naïve distributed PCA method can be obtained by simply approximating the global correlation matrix via the AC subroutine, *i.e.,* for some $L \geq 1$, we have

$$\hat{\boldsymbol{R}}_{x,i} = \frac{S}{T}\cdot \mathsf{AC}_i\big(\{\boldsymbol{X}_j^c(\boldsymbol{X}_j^c)^{\mathsf{H}}\}_{j=1}^{S}; L\big) \approx \hat{\boldsymbol{R}}_x \ . \tag{20}$$

In other words, each agent obtains an approximate of the global correlation matrix and the desired PCA can be then computed from $\hat{\boldsymbol{R}}_{x,i}$ using, for example, the power method. The drawback, however, lies on the communication and computation cost entailed in this approach, which is particularly onerous when considering high-dimensional data with $N \gg 1$. This is because (20) requires computing the average of an $N \times N$ matrix.

As an improvement over the correlation averaging method, next we describe a natural extension of the power method in (14) to the distributed setting with DCO data. In this setting, the

REVIEW COPY

distributed power method (DistPM) was introduced as a subroutine in [38], [39]. Observe that the first expression in (14) can be written as:

$$\begin{aligned} \boldsymbol{u}_1[k] &= \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^{\mathsf{H}}\overline{\boldsymbol{u}}_1[k-1] \\ &= \frac{1}{T}\sum_{i=1}^{S}\left(\boldsymbol{X}_i^c(\boldsymbol{X}_i^c)^{\mathsf{H}}\overline{\boldsymbol{u}}_1[k-1]\right), \end{aligned} \tag{21}$$

If a copy of $\overline{\boldsymbol{u}}_1[k-1]$ is also known to the agents, computing each of the terms requires data known to the ith agent and, therefore, $\boldsymbol{u}_1[k]$ can simply be computed adding up the S terms — a computation that can be accomplished using the average consensus subroutine. This would be the case if the nodes start with the same initial vector $\boldsymbol{u}_1[0]$ and achieve asymptotic convergence to the average. In reality both conditions can be relaxed.

Specifically, for the DistPM with DCO data, we denote $\boldsymbol{u}_1^i[k] \in \mathbb{C}^N$ (*resp.* $\overline{\boldsymbol{u}}_1^i[k]$) as the estimate of the first (*resp.* normalized) eigenvector of $\boldsymbol{R}_x$ at the kth power iteration, kept by the ith agent. The DistPM proceeds by:

Distributed Power Method (DistPM) for DCO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_1^i[0] \sim \mathcal{CN}(\boldsymbol{0}, \boldsymbol{I})$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:
$$\boldsymbol{u}_1^i[k] = S \cdot \mathsf{AC}_i\big(\{\boldsymbol{X}_j^c(\boldsymbol{X}_j^c)^{\mathsf{H}}\overline{\boldsymbol{u}}_1^j[k-1]\}_{j=1}^S; L\big),$$
$$\overline{\boldsymbol{u}}_1^i[k] = \boldsymbol{u}_1^i[k]/\|\boldsymbol{u}_1^i[k]\|,$$
for all $k \geq 1$ and we terminate after $k \geq K$.
3) Denote $\hat{\boldsymbol{u}}_1^i := \boldsymbol{u}_1^i[K]$ as the solution kept by the ith agent for all $i \in \{1, ..., S\}$.

The subsequent eigenvectors $\boldsymbol{u}_2^\star, ..., \boldsymbol{u}_p^\star$ can be found by using a similar strategy as in the centralized power method.

Theoretical Guarantee: The convergence of DistPM (DCO) has been analyzed in [39] for real data matrices and static network between the agents, *i.e.,* $\boldsymbol{W}[\ell] = \overline{\boldsymbol{W}}$ for all ℓ. Again, for simplicity, we only present the result for $p = 1$ where the top eigenvector of $\hat{\boldsymbol{R}}_x$ is sought. We have

Theorem IV.1 *Consider the DistPM (DCO) method for real data matrices and* $p = 1$. *Fix* $1/2 > \epsilon > 0$ *be the desirable accuracy, and* $c > 0$ *be the failure probability. If the DistPM*

REVIEW COPY

parameters satisfy:

$$L = \Omega\left(\frac{\log \epsilon^{-1} + \log(\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x))^{-1}}{\log(\lambda_{\text{conn.}}^{-1})}\right)$$
$$K = \Omega\left(\frac{\lambda_1(\hat{\boldsymbol{R}}_x)}{\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x)} \cdot \log\left(\frac{N}{c \cdot \epsilon}\right)\right), \tag{22}$$

and the initialization satisfies $(\boldsymbol{u}_1^\star)^\top \boldsymbol{u}_1^i[0] \neq 0$*, then with probability at least* $1 - Sc$*, we have:*

$$((\boldsymbol{u}_1^\star)^\top \hat{\boldsymbol{u}}_1^i)^2 \geq 1 - \epsilon^2, \quad \min_{j=2,\ldots,N}((\boldsymbol{u}_j^\star)^\top \hat{\boldsymbol{u}}_1^i)^2 \leq \epsilon^2 \ , \tag{23}$$

for all $i \in \{1, ..., S\}$*. Also, the eigenvectors found are in consensus:*

$$\|\hat{\boldsymbol{u}}_1^i(\hat{\boldsymbol{u}}_1^i)^\top - \hat{\boldsymbol{u}}_1^{i'}(\hat{\boldsymbol{u}}_1^{i'})^\top\| = \mathcal{O}(\epsilon), \ \forall \ i, i' \in \{1, ..., S\} \ . \tag{24}$$

Here, the randomness in the algorithm is due to the random initializations. The theorem above is taken from [39, Proposition 1] and was partially inspired by [40]. Theorem IV.1 gives a *non-asymptotic* bound on the accuracy achieved by the DistPM (DCO) as it does not require $P \to \infty$.

C. Application Examples

We consider applying PCA to dimensionality reduction and clustering in a distributed setting. Particularly, we assume that the ith machine/agent observes a dataset which can be described as:

$$\boldsymbol{X}_i^c = \boldsymbol{A}\boldsymbol{Z}_i^c + \boldsymbol{E}_i \in \mathbb{R}^{N \times T_i} \ , \tag{25}$$

where each row of $\boldsymbol{X}_i^c$ represents a *feature* and each column is a data point. In the above, $\boldsymbol{A} \in \mathbb{R}^{N \times p}$ is a common 'dictionary' with $p \ll N$, $\boldsymbol{Z}_i^c \in \mathbb{R}^{p \times T_i}$ is the latent parameters for the observations and $\boldsymbol{E}_i$ represents the modeling error. Furthermore, $\boldsymbol{X}_i^c$ constitutes a set of *training data*, and our aim is to learn the subspace spanned by the columns of $\boldsymbol{A}$ so that we can compress some unseen data, $\boldsymbol{X}_{\text{test}}$, which are also generated by the same model, using their p-D latent parameters. Specifically, if $\boldsymbol{U}^\star$ is the learnt p-D subspace from the collection of training data $\{\boldsymbol{X}_i^c\}_{i=1}^S$, then $\boldsymbol{X}_{\text{test}}$ can be compressed as $\boldsymbol{Z}_{\text{test}} := (\boldsymbol{U}^\star)^{\mathsf{H}} \boldsymbol{X}_{\text{test}}$. We assume that the modeling errors $\boldsymbol{E}_i$ are independent, and the dimensionality reduction problem can now be solved as a static PCA problem (3). Furthermore, our target problem corresponds to the DCO setting considered in (8) as the data points are collected locally by the machines.

REVIEW COPY

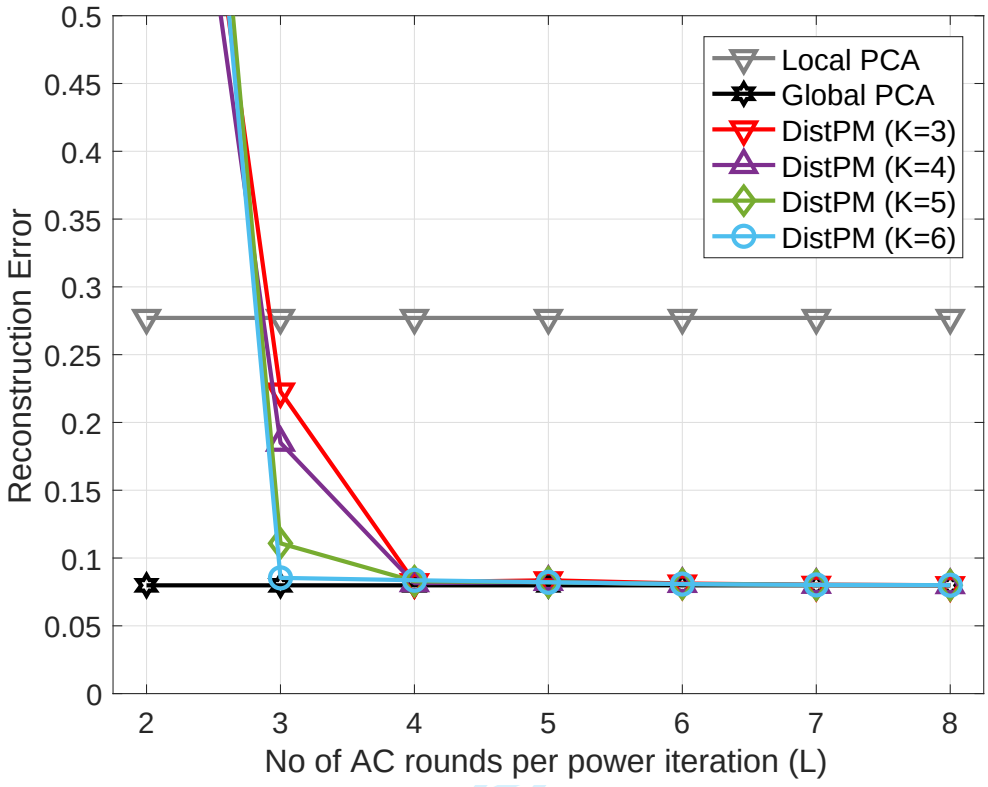

Fig. 6. **Applying DistPM (DCO) on the dimensionality reduction problem**. The reconstruction error is shown for agent $i = 1$ on the testing data, which is generated according to a similar model as (25). In the legend, 'local PCA' (*resp.* 'global PCA') refers to using only the local data (*resp.* all available data) to estimate the principal subspace.

To illustrate the performance of the distributed PCA method (DistPM), we consider the data model of (25) with $N = 1000$, $p = 2$ and $T_i = 2$; while the SNR is fixed at 25dB. We consider a network with $S = 100$ nodes, connected through a mean degree-10 small world graph with rewiring probability of 0.2. Fig. 6 shows the reconstruction error in different settings for the DistPM (DCO) with different number of AC rounds.

The development of distributed dimensionality reduction techniques provide both efficient storage and preprocessing (denoising) of the data for machine learning. Fig. 7 shows an application of dimensionality reduction in data clustering. We consider a set of high-dimensional data distributed over $S = 100$ nodes. Specifically, the data consist of four separated clusters. Each cluster is a high-dimensional embedding of a swiss roll using the model in (25) with $N = 200$. Fig. 7(a) shows the original clusters of swiss roll data that are used to generate the high-dimensional data. Fig. 7(b) shows the result of dimensionality reduction using the estimated

IEEE COPY

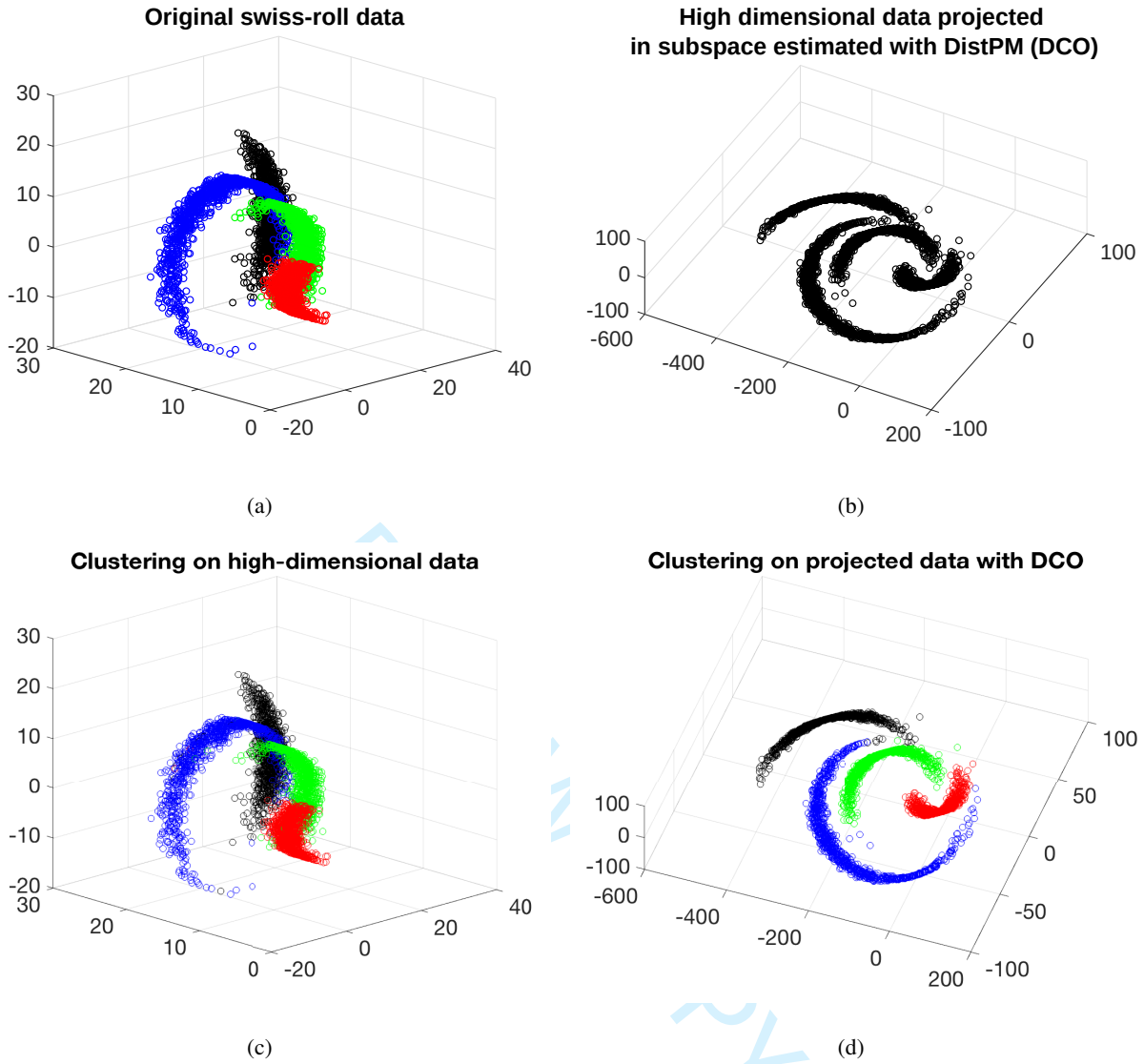

Fig. 7. **Clustering on the swiss roll data** (a) Original swiss roll data before high-dimensional embedding; (b) High-dimensional data projected in 3-dimensional subspace estimated with DistPM (DCO); (c) Results of hierarchical clustering on the original high-dimensional data; and, (d) Results of hierarchical clustering on the projected data with DistPM (DCO).

subspace with DistPM (DCO). We then perform Ward's hierarchical clustering algorithm [41] on both the original high-dimensional data and the projected data[3]. The normalized mutual information (NMI) is used to measure the clustering performance. Fig. 7(c) and 7(b) show the detected clusters with NMI = 0.9785 and 0.9797, respectively.

[3]Here we let the nodes to broadcast their local projected data to the network.

V. Distributed PCA Methods for DRO

In this section, we focus on the DRO setting and survey the available distributed PCA strategies in the literature. An interesting feature under this setting is that the goal of each agent is to compute a *portion* of the orthogonal transformation $\boldsymbol{U}^\star$ required. This requires one to design algorithms carefully in order to take advantage of these requirements while respecting the DRO data structure. We remark that most of the prior work for this data structure are designed for the mesh network scenario.

A. Distributed Power Method (DRO)

The distributed power method (DistPM) was developed in [18] for the DRO setting. In a nutshell, the method combines average consensus and power method to estimate the principal subspace distributively. Note that this is a *batch* method that applies to the *static* setting in (3).

We first derive the DistPM for just the top eigenvector of $\hat{\boldsymbol{R}}_x$, denoted as $\boldsymbol{u}_1^\star$. To set up the stage, we define $\boldsymbol{u}_{1,i}[k]$ as the local portion of the top eigenvector kept by the ith agent at iteration k and $\boldsymbol{u}_1[k] := (\boldsymbol{u}_{1,1}[k]; \ldots; \boldsymbol{u}_{1,S}[k])$ as the *global* eigenvector estimate. To initialize, each agent generates a random vector $\boldsymbol{u}_{1,i}[0] \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ independently.

Consider the kth power iteration in (14), we see that the normalization step may be relegated to the end of the iterations as it only involves a scalar multiplication. Instead, we focus on the second step in (14). Observe that the ith row partition in $\boldsymbol{u}_1[k+1]$ can be written as:

$$\begin{aligned} &\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}_i(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_1[k] \\ &= \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}_i(t)\left(\sum_{j=1}^{S} \boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_{1,j}[k]\right) , \end{aligned} \tag{26}$$

note that we are using the *unnormalized* vector $\boldsymbol{u}_{1,j}[k]$ on the right hand side of the equation. The vectors $\boldsymbol{x}_j(t)$ and $\boldsymbol{u}_{1,j}[k]$ are known to the jth agent. Therefore, the inner products inside the summation are *locally* computable by the jth agent. This inspires us to replace the summation with the previously introduced AC subroutine. Specifically, the required computation can be performed by the pseudo-code:

REVIEW COPY

Distributed Power Method (DistPM) for DRO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_{1,i}[0] \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:
$$\boldsymbol{u}_{1,i}[k+1] = \frac{S}{T} \sum_{t=1}^{T} \boldsymbol{x}_i(t) \mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t) \boldsymbol{u}_{1,j}[k]\}_{j=1}^{S}; L\big)$$
for all $k \geq 0$ and we terminate after $k \geq K-1$.
3) Denote $\boldsymbol{u}_{1,i}[K]$ as the un-normalized top eigenvector kept by the ith agent for all $i \in \{1, ..., S\}$.

Note that we have replaced the inner product $\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_1[k]$ by its distributed average approximation $\mathsf{AC}_i(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_{1,j}[k]\}_{j=1}^{S}; L)$.

The procedure above finds the ith block of the *un-normalized* top eigenvector of $\hat{\boldsymbol{R}}_x$. As a final step, we shall take care of the normalization factor which we have ignored. Observe that

$$\|\boldsymbol{u}_1[K]\| = \sqrt{\textstyle\sum_{i=1}^{S} \|\boldsymbol{u}_{1,i}[K]\|^2}\ , \tag{27}$$

once again we observe that the terms inside the summation are merely the squared norm of the local estimate $\boldsymbol{u}_{1,i}[K]$, which is obviously known to the ith agent. This can be replaced by another AC subroutine. Let us define:

$$\mathsf{norm}_i := \sqrt{S \cdot \mathsf{AC}_i(\{\|\boldsymbol{u}_{1,j}[K]\|^2\}_{j=1}^{S}; L')} \approx \|\boldsymbol{u}_1[K]\|\ , \tag{28}$$

where we set $L' \gg L$. Finally, we denote the obtained partial eigenvector as $\hat{\boldsymbol{u}}_{1,i}$ such that

$$\hat{\boldsymbol{u}}_{1,i} := \big(\mathsf{norm}_i\big)^{-1} \boldsymbol{u}_{1,i}[K]\ , \tag{29}$$

and the concatenated version of it as $\hat{\boldsymbol{u}}_1 := (\hat{\boldsymbol{u}}_{1,1}; \ldots; \hat{\boldsymbol{u}}_{1,S})$. We can also compute the eigenvalue λ_1 associated with $\boldsymbol{u}_1^\star$ by observing:

$$\lambda_1 = \hat{\boldsymbol{u}}_1^{\mathsf{H}} \Big(\underbrace{\tfrac{1}{T} \textstyle\sum_{t=1}^{T} \boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\hat{\boldsymbol{u}}_1}_{\approx(\boldsymbol{u}_{1,1}[K];\ldots;\boldsymbol{u}_{1,S}[K])} \Big)\ . \tag{30}$$

The above can be approximated by:

$$\hat{\lambda}_{1,i} = S \cdot \mathsf{AC}_i\big(\{\hat{\boldsymbol{u}}_{1,j}^{\mathsf{H}} \boldsymbol{u}_{1,j}[K]\}_{j=1}^{S}; L\big)\ . \tag{31}$$

However, we note that this step is optional as the eigenvalue is not required in the computation of the second, third, etc. eigenvectors.

REVIEW COPY

To compute the second eigenvector $\boldsymbol{u}_2^\star$, we observe that it is equivalent to computing the top eigenvector for $(\boldsymbol{I} - \boldsymbol{u}_1^\star(\boldsymbol{u}_1^\star)^{\mathsf{H}})\hat{\boldsymbol{R}}_x$. In light of this, the respective power iteration can be approximated as:

$$\boldsymbol{u}_2[k+1] = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{I} - \hat{\boldsymbol{u}}_1\hat{\boldsymbol{u}}_1^{\mathsf{H}})\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k] \tag{32}$$

Similar to the previous derivations in (26), the ith row block of the above can be written as:

$$\frac{1}{T}\sum_{t=1}^{T}\Big(\boldsymbol{x}_i(t) - \hat{\boldsymbol{u}}_{1,i}\underbrace{\hat{\boldsymbol{u}}_1^{\mathsf{H}}\boldsymbol{x}(t)}_{\approx \mathsf{prod}_{1,i}}\Big)\big(\underbrace{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k]}_{\approx \mathsf{prod}_{2,i}(k)}\big)\ , \tag{33}$$

Importantly, the highlighted inner products are replaced respectively by:

$$\begin{aligned}\mathsf{prod}_{1,i} &:= S\cdot\mathsf{AC}_i\big(\{\hat{\boldsymbol{u}}_{1,j}^{\mathsf{H}}\boldsymbol{x}_j(t)\}_{j=1}^{S}; L\big)\\ &\approx \hat{\boldsymbol{u}}_1^{\mathsf{H}}\boldsymbol{x}(t)\\ \mathsf{prod}_{2,i}(k) &:= S\cdot\mathsf{AC}_i\big(\{(\boldsymbol{x}_j(t))^{\mathsf{H}}\boldsymbol{u}_{2,j}[k]\}_{j=1}^{S}; L\big)\\ &\approx \boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}_2[k]\ .\end{aligned} \tag{34}$$

where the right hand side can be computed distributively using the AC subroutine. Finally, the power iteration is performed by:

$$\boldsymbol{u}_{2,i}[k+1] = \frac{1}{T}\sum_{t=1}^{T}\Big(\boldsymbol{x}_i(t) - \hat{\boldsymbol{u}}_{1,i}\cdot\mathsf{prod}_{1,i}\Big)\mathsf{prod}_{2,i}[k]\ . \tag{35}$$

We can compute $\hat{\boldsymbol{u}}_{3,i}, ..., \hat{\boldsymbol{u}}_{p,i}$ using a similar procedure, again in a distributed fashion.

Theoretical Guarantees: The convergence of DistPM (DRO) for the static network between the agents, *i.e.,* with $\boldsymbol{W}[\ell] = \overline{\boldsymbol{W}}$ for all ℓ, has been analyzed in [21] in the asymptotic regime as $P \to \infty$. Here, we only present a simplified version of their result for the special case of $p = 1$. Importantly, we observe that when $L \to \infty$ such that the AC subroutine computes the exact averages, the DistPM method will be equivalent to the original/centralized power method. As such, we expect that the performance of DistPM will improve with L, as shown in the following:

Theorem V.1 *Consider the DistPM (DRO) which terminates after P power iterations. If $P \to \infty$, we have*

$$\|\hat{\boldsymbol{u}}_1 - \boldsymbol{u}_1^\star\| = \mathcal{O}\left(\frac{\lambda_1(\hat{\boldsymbol{R}}_x)}{\lambda_1(\hat{\boldsymbol{R}}_x) - \lambda_2(\hat{\boldsymbol{R}}_x)}\cdot\lambda_{\text{conn.}}^{L}\right)\ . \tag{36}$$

The above theorem is derived from [21, Theorem 3]. Theorem V.1 bounds the difference between $\boldsymbol{u}_1^\star$ and $\hat{\boldsymbol{u}}_1$. It confirms our expectation that the DistPM's performance depends on the number of AC iterations L and in particular, the performance improves exponentially with L. Moreover,

REVIEW COPY

the error also depends on the spectral gap in $\hat{\boldsymbol{R}}_x$ such that the error is reduced when $\lambda_1(\hat{\boldsymbol{R}}_x) \gg \lambda_2(\hat{\boldsymbol{R}}_x)$. It can be seen that the required number of AC steps, L, has the same scaling as in Theorem IV.1.

B. Distributed Oja's Method

As opposed to the distributed power method, the Oja's method can be applied to the *dynamic* PCA problem (4) which is a non-convex, time varying, stochastic optimization problem. A distributed version of the Oja's method, named D-Oja, was developed in [23] for the DRO setting. Here, we focus on the setting with $p = 1$. The first step is to approximate the update (17) as:

$$\begin{aligned} &\boldsymbol{u}^{\mathsf{Oja}}(t+1) = \boldsymbol{u}^{\mathsf{Oja}}(t) + \\ &\gamma_t \Big(\boldsymbol{x}(t)\boldsymbol{x}^{\mathsf{H}}(t) - |\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)|^2 \boldsymbol{I} \Big) \boldsymbol{u}^{\mathsf{Oja}}(t) \ . \end{aligned} \tag{37}$$

Similar to the development of DistPM, the D-Oja method relies on interpreting the Oja's updates as a combination of several computations that are replaceable by distributed algorithms. In particular, consider the ith block of $\boldsymbol{u}^{\mathsf{Oja}}(t+1)$ in (37):

$$\begin{aligned} &\boldsymbol{u}_i^{\mathsf{Oja}}(t+1) = \boldsymbol{u}_i^{\mathsf{Oja}}(t) + \\ &\gamma_t \Big(\boldsymbol{x}_i(t) \cdot \underline{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)} - \boldsymbol{u}_i^{\mathsf{Oja}}(t) \cdot \big| \underline{\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)} \big|^2 \Big) \ . \end{aligned} \tag{38}$$

In particular, for the ith agent, it suffices to compute the complex-valued inner product $\boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t)$ in order to complete the update. To distribute the computation, again we observe that this inner product is computed from a number of terms that are computed locally at S agents:

$$\begin{aligned} \boldsymbol{x}^{\mathsf{H}}(t)\boldsymbol{u}^{\mathsf{Oja}}(t) &= \textstyle\sum_{i=1}^{S} \boldsymbol{x}_i^{\mathsf{H}}(t)\boldsymbol{u}_i^{\mathsf{Oja}}(t) \\ &\approx S \cdot \mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_j^{\mathsf{Oja}}(t)\}_{j=1}^{S}; L \big) \ , \end{aligned} \tag{39}$$

where L is a sufficiently large integer controlling the number of average consensus steps used. To summarize, we have the pseudo-code:

REVIEW COPY

Distributed Oja's (D-Oja) Method for DRO:

1) Initialize for each agent an independent random vector, *i.e.,* $\boldsymbol{u}_i^{\mathsf{Oja}}(0) \sim \mathcal{CN}(0, \boldsymbol{I}) \in \mathbb{C}^{N_i}$ for all i.
2) For all agent $i \in \{1, ..., S\}$, perform the recursion:

$$\begin{aligned} \mathsf{prod}_i(t) &:= S \cdot \mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(t)\boldsymbol{u}_j^{\mathsf{Oja}}(t)\}_{j=1}^{S}; L\big) \ , \\ \boldsymbol{u}_i^{\mathsf{Oja}}(t+1) &= \boldsymbol{u}_i^{\mathsf{Oja}}(t) + \gamma_t\Big(\mathsf{prod}_i(t) \cdot \boldsymbol{x}_i(t) \\ &\qquad - |\mathsf{prod}_i(t)|^2 \cdot \boldsymbol{u}_i^{\mathsf{Oja}}(t)\Big) \ . \end{aligned} \tag{40}$$

for all $t \geq 0$.

For general $p > 1$, the authors in [23] proposed an alternative method that involves tackling a different objective function than (15). The developed method will be described in the next subsection.

Theoretical Guarantees: The asymptotic [*i.e.,* when $t \to \infty$] convergence of D-Oja method has been analyzed in [23] under the assumption that $\boldsymbol{x}(t)$ is stationary. For simplicity, we also consider the case of static network. We have

Theorem V.2 *Suppose that the step size γ_t is sufficiently small and L is sufficiently large, then the D-Oja method* (40) *can be approximated as an ODE. Furthermore, the ODE has a stable equilibrium $\hat{\boldsymbol{u}}_1$ that satisfies:*

$$\|\boldsymbol{u}_1^\star - \hat{\boldsymbol{u}}_1\| \leq 3S\lambda_1(\hat{\boldsymbol{R}}_x) \cdot \lambda_{\text{conn.}}^L + \mathcal{O}(S^2\lambda_{\text{conn.}}^{2L}) \ . \tag{41}$$

The proof can be found in [23, Lemma 4.2]. Similar to the results for DistPM (DRO), Theorem V.2 also shows an exponential dependence on the number of AC steps L in the error produced by the D-Oja method.

C. Power-Oja's method

As discussed earlier on, the Oja's method (as well as the D-Oja method) is known to converge slowly, *i.e.,* at most at the rate of $\mathcal{O}(1/t)$. To accelerate the convergence rate, the authors in [25] proposed a Power-Oja's (P-Oja) method that combines the fast convergence of power method with the adaptivity of the Oja's method. Notice that the power method is a *batch* method that requires the random process $\boldsymbol{x}(t)$ to be stationary. As a compromise, we assume *quasi-stationarity* for $\boldsymbol{x}(t)$ such that the process is stationary over the period $\mathcal{I}_k$ in which $\boldsymbol{x}(\tau)$ has the same distribution

REVIEW COPY

for all $\tau \in \mathcal{I}_k$. Furthermore, in contrast to the Oja's method, the P-Oja's method tackles the following dynamic PCA problem:

$$\boldsymbol{U}^\star(k) \in \arg\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} f_\tau(\boldsymbol{U}) \text{ s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} \ , \tag{42}$$

where $\tau \in \mathcal{I}_k$ and

$$f_\tau(\boldsymbol{U}) := \mathbb{E}\Big[\mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\boldsymbol{x}(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau)\Big)\Big] \ . \tag{43}$$

It can be shown that the above problem is equivalent to (4).

We apply stochastic approximation for (42) to obtain a tractable objective function. Let $\mathcal{B}_{k,s} \subseteq \mathcal{I}_k$ be the sth batch in period $\mathcal{I}_k$, the objective function in the above can be approximated by:

$$\hat{f}(\boldsymbol{U};\mathcal{B}_{k,s}) := \mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\hat{\boldsymbol{R}}(\mathcal{B}_{k,s})\Big) \tag{44}$$

where

$$\hat{\boldsymbol{R}}(\mathcal{B}_{k,s}) := \frac{1}{|\mathcal{B}_{k,s}|}\sum_{\tau\in\mathcal{B}_{k,s}} \boldsymbol{x}(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau) \ , \tag{45}$$

which is a *mini-batch* sampled correlation matrix.

From here, one may apply the SGD method and update the principal subspaces by taking the gradient of $\hat{f}(\boldsymbol{U};\mathcal{B}_{k,s})$. Doing so results in a method similar to Oja's which may suffer from the same slow convergence rate. Instead, let P be a predefined integer constant, we consider a modified stochastic approximation function:

$$\hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s}) := \mathrm{Tr}\Big(\big(\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}} - 2\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\big)\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\Big) \ . \tag{46}$$

Importantly, the optimal solution set to the modified stochastic approximation function is the same as the original one:

$$\begin{aligned}\arg\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \ & \hat{f}(\boldsymbol{U};\mathcal{B}_{k,s}) = \arg\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \ \hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s}) \\ \text{s.t. } & \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} \qquad\qquad \text{s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I} \ .\end{aligned} \tag{47}$$

The benefit of considering $\hat{f}^{\mathsf{POja}}(\cdot)$ in lieu of $\hat{f}(\cdot)$ is that the former admits a better *spectral gap* for the sampled correlation since

$$\frac{\sigma_p(\hat{\boldsymbol{R}}^P) - \sigma_{p+1}(\hat{\boldsymbol{R}}^P)}{\sigma_p(\hat{\boldsymbol{R}}^P)} > \frac{\sigma_p(\hat{\boldsymbol{R}}) - \sigma_{p+1}(\hat{\boldsymbol{R}})}{\sigma_p(\hat{\boldsymbol{R}})} \ , \tag{48}$$

as we notice that $\sigma_p(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})) = \sigma_p(\hat{\boldsymbol{R}}(\mathcal{B}_{k,s}))^P$, where $\sigma_p(\boldsymbol{R})$ denotes the pth largest singular value of $\boldsymbol{R}$. As shown in the analysis of [35], the size of the spectral gap $\sigma_p(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})) -$

REVIEW COPY

$\sigma_{p+1}(\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s}))$ is an important factor in determining the convergence speed of subspace estimation/tracking algorithms. Therefore, we anticipate that using $\hat{f}^{\mathsf{POja}}(\cdot)$ should yield a faster PCA algorithm than using $\hat{f}^{\mathsf{Oja}}(\cdot)$. Finally, observe that the gradient of the new stochastic approximation function is:

$$\begin{aligned}\nabla \hat{f}^{\mathsf{POja}}(\boldsymbol{U};\mathcal{B}_{k,s}) = &-2\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U} \\ &+ \hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} + \boldsymbol{U}\boldsymbol{U}^{\mathsf{H}}\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}\ .\end{aligned} \tag{49}$$

Remarkably, $\hat{\boldsymbol{R}}^P(\mathcal{B}_{k,s})\boldsymbol{U}$ is similar to running a power method for P steps with the columns of $\boldsymbol{U}$ as initialization vectors (an insight that motivated this distributed PCA approach). Lastly, relaxing the Grassmanian manifold constraint $\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I}$ yields the P-Oja method:

$$\boldsymbol{U}(k,s+1) = \boldsymbol{U}(k,s) - \gamma_s \nabla \hat{f}^{\mathsf{POja}}(\boldsymbol{U}(k,s);\mathcal{B}_{k,s})\ , \tag{50}$$

this is similar to the NOja's method considered in [36].

Note that P-Oja method requires a non-trivial batch size, $|\mathcal{B}_{k,s}| \geq 2$ to show its benefits, which depend on a sufficient increase in the spectral gap.

Distributed Method: The authors in [25] considered a distributed extension for the P-Oja method. Their philosophy is similar to that employed by the DistPM and D-Oja method, as we observe that the update of the ith block [cf. (49)] is:

$$\boldsymbol{U}_i(s) - \gamma_s \big[\nabla \hat{f}^{\mathsf{POja}}(\boldsymbol{U}(s);\mathcal{B}_s)\big]_i\ , \tag{51}$$

where we have dropped the index of k for simplicity and

$$\begin{aligned}&\big[\nabla \hat{f}^{\mathsf{POja}}(\boldsymbol{U}(s);\mathcal{B}_s)\big]_i := \\ &-2\underbrace{[\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i}_{\approx \mathsf{prod}_{1,i}(s)} + \underbrace{[\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i}_{\approx \mathsf{prod}_{1,i}(s)}\underbrace{\boldsymbol{U}^{\mathsf{H}}(s)\boldsymbol{U}(s)}_{\approx \mathsf{prod}_{2,i}(s)} \\ &+ \boldsymbol{U}_i(s)\underbrace{\boldsymbol{U}^{\mathsf{H}}(s)\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)}_{\approx \mathsf{prod}_{3,i}(s)}\ .\end{aligned} \tag{52}$$

The required computation boils down to evaluating the highlighted terms in the above. First, we observe that $\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)$ is the result of applying P power iterations with the sampled correlation matrix $\hat{\boldsymbol{R}}(\mathcal{B}_s)$, initialized on $\boldsymbol{U}(s)$. Each of the power iteration can be handled in a similar fashion as in the DistPM. In particular, let $\hat{\boldsymbol{U}}(s,q)$ be the intermediate variable at the

REVIEW COPY

qth power iteration, $\hat{\boldsymbol{U}}_i(s,q)$ be its ith row block, and $\hat{\boldsymbol{U}}(s,1)=\boldsymbol{U}(s)$, the power iteration can be approximated as

$$\begin{aligned}&\hat{\boldsymbol{U}}_i(s,q+1)\\&=\frac{S}{|\mathcal{B}_s|}\sum_{\tau\in\mathcal{B}_s}\boldsymbol{x}_i(\tau)\cdot\mathsf{AC}_i\big(\{\boldsymbol{x}_j^{\mathsf{H}}(\tau)\hat{\boldsymbol{U}}_j(s,q)\}_{j=1}^S;L\big)\\&\approx\frac{1}{|\mathcal{B}_s|}\sum_{\tau\in\mathcal{B}_s}\boldsymbol{x}_i(\tau)\boldsymbol{x}^{\mathsf{H}}(\tau)\hat{\boldsymbol{U}}(s,q)\ .\end{aligned}\tag{53}$$

Repeating the above for P iterations yields

$$\mathsf{prod}_{1,i}(s):=\hat{\boldsymbol{U}}_i(s,P+1)\approx[\hat{\boldsymbol{R}}^P(\mathcal{B}_s)\boldsymbol{U}(s)]_i\ .\tag{54}$$

For the remaining terms, we can handle them easily by:

$$\begin{aligned}\mathsf{prod}_{2,i}(s)&:=S\cdot\mathsf{AC}_i\big(\{\boldsymbol{U}_j^{\mathsf{H}}(s)\boldsymbol{U}_j(s)\}_{j=1}^S;L\big)\\&\approx\textstyle\sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)\boldsymbol{U}_j(s)\ ,\end{aligned}\tag{55}$$

$$\begin{aligned}\mathsf{prod}_{3,i}(s)&:=S\cdot\mathsf{AC}_i\big(\{\boldsymbol{U}_j^{\mathsf{H}}(s)\mathsf{prod}_{1,j}(s)\}_{j=1}^S;L\big)\\&\approx\sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)\mathsf{prod}_{1,j}(s)\approx\sum_{j=1}^S\boldsymbol{U}_j^{\mathsf{H}}(s)[\hat{\boldsymbol{R}}^P\boldsymbol{U}(s)]_j\ .\end{aligned}\tag{56}$$

Finally, the P-Oja update for agent i can be approximated as:

$$\begin{aligned}\boldsymbol{U}_i(s+1)=\boldsymbol{U}_i(s)-\gamma_s\Big(&\boldsymbol{U}_i(s)\mathsf{prod}_{3,i}(s)\\&+\mathsf{prod}_{1,i}(s)\mathsf{prod}_{2,i}(s)-2\mathsf{prod}_{1,i}(s)\Big)\ .\end{aligned}\tag{57}$$

This demonstrates that the P-Oja update can be replaced by a number of average consensus steps to be computed distributively. We remark that the P-Oja update can be reduced to the multiple subspaces-tracking D-Oja method (*i.e.,* when $p>1$) by setting $P=1$.

D. Application Examples

1) Distributed Distance Matrix Estimation: We demonstrate that the distance matrix estimation problem can be cast as a PCA problem, which naturally admits a DRO data structure when we consider a sensor network setting. Here, agent/sensor i gathers an d-dimensional observation $\boldsymbol{x}_i\in\mathbb{R}^d$, $i=1,...,S$. Our goal is to compute the pairwise distances $\|\boldsymbol{x}_i-\boldsymbol{x}_j\|$ which are necessitated by clustering problems or dimensionality reduction with ISOMAP [20]. Let us observe that

$$\|\boldsymbol{x}_i-\boldsymbol{x}_j\|=\sqrt{\|\boldsymbol{x}_i\|^2+\|\boldsymbol{x}_j\|^2-2\boldsymbol{x}_i^\top\boldsymbol{x}_j}\ .\tag{58}$$

REVIEW COPY

Importantly, the squared norms and inner products in the above can be obtained from the following matrix:

$$\begin{aligned} \boldsymbol{A} &:= \boldsymbol{X}\boldsymbol{X}^\top \in \mathbb{R}^{S\times S}, \text{ where} \\ \boldsymbol{X} &:= (\boldsymbol{x}_1 \ \cdots \ \boldsymbol{x}_S)^\top \in \mathbb{R}^{S\times d} . \end{aligned} \tag{59}$$

We have $\boldsymbol{x}_i^\top \boldsymbol{x}_j = A_{ij}$ and therefore the pairwise distance can be computed through $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = \sqrt{A_{ii} + A_{jj} - 2A_{ij}}$. To estimate the pairwise distances, we can therefore compute the rank-p approximation of $\boldsymbol{A}$:

$$\hat{\boldsymbol{A}}_p = \textstyle\sum_{r=1}^p \lambda_r \boldsymbol{u}_r^\star (\boldsymbol{u}_r^\star)^\top \approx \boldsymbol{A} \ , \tag{60}$$

In particular, $\hat{\boldsymbol{A}}_p$ is computed from the eigenvalue-vector pairs $\{\lambda_r, \boldsymbol{u}_r^\star\}_{r=1}^p$, *i.e.,* it is a PCA problem. We notice that the data is structured in a DRO setting since each agent holds a row vector in $\boldsymbol{X}$. This is a static PCA problem as the data matrix does not change over time. Thus DistPM is a suitable algorithm. Fig. 8 compares the true and estimated distance matrix using DistPM (DRO) on a sensor network with $S = 1000$ sensors/agents, with the graph topology arranged according to a random geometric graph. The DistPM (DRO) finds the distance matrix accurately.

2) Distributed Direction-of-Arrival Tracking: We next illustrate how to track the direction-of-arrival (DoA) of multiple objects in a massive antenna array system using the distributed PCA methods. Consider a scenario where the (groups of) antennas are arranged in a *uniform linear array* (ULA) configuration, receiving signals from a far object. An example is depicted in Fig. 9. When there are p objects to be tracked, it is well known [1], [2] that the signal received at the antennas is:

$$\boldsymbol{x}(t) = \textstyle\sum_{q=1}^p \boldsymbol{a}(d, \theta_q) \boldsymbol{z}_q(t) + \boldsymbol{e}(t) \ , \tag{61}$$

where

$$\boldsymbol{a}(d, \theta_q) := (1, \ e^{-j\omega_c d \sin\theta_q/\lambda}, \ldots, \ e^{-j\omega_c (S-1) d \sin\theta_q/\lambda})^\top,$$

such that λ is the wavelength and ω_c is the carrier frequency, $\boldsymbol{z}_q(t)$ is the signal transmitted from the qth object, and $\boldsymbol{e}(t)$ is a white additive noise with variance σ^2. Importantly, the vectors $\{\boldsymbol{a}(d, \theta_q)\}_{q=1}^p$ are mutually orthogonal when the number of antennas is large, provided that $\theta_q \neq \theta_{q'}$, thus the p-D principal subspace of the signal $\boldsymbol{x}(t)$ is given by $\boldsymbol{A} := (\boldsymbol{a}(d, \theta_1) \ \cdots \ \boldsymbol{a}(d, \theta_p))$ and the model fits into the one described by (2). The direction-of-arrival (DoA) for multiple objects can thus be found by extracting this principal subspace. This leads to the celebrated

REVIEW COPY

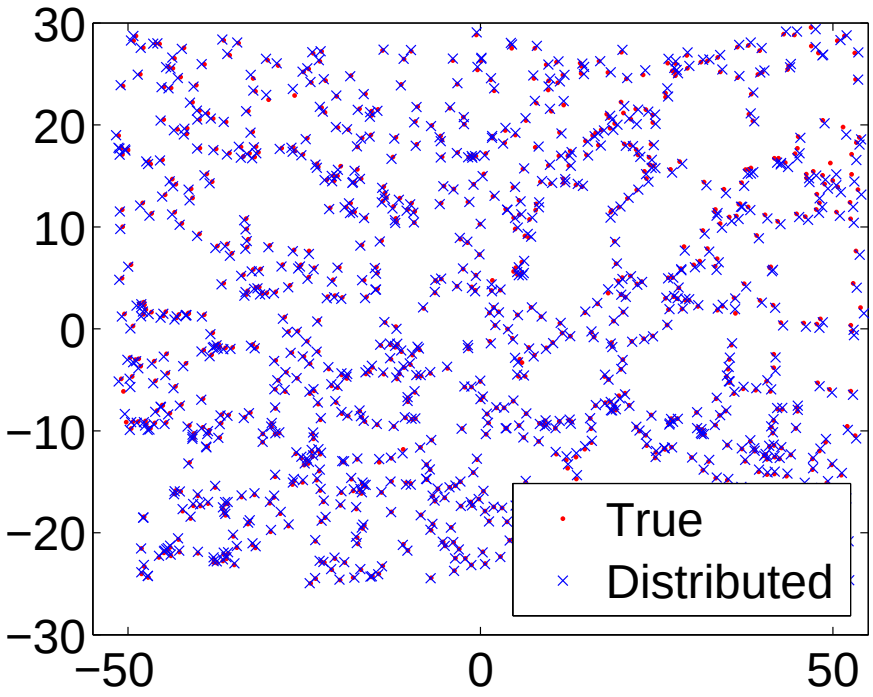

Fig. 8. **Distributed Distance Matrix Estimation**. Comparing the true and estimated distance matrix on a sensor network with $S = 1000$ sensors. We set $K = 50, L = 500$ for the DistPM and reconstructed the distance matrix through finding its rank-2 approximation.

IEEE COPY

MUSIC/ESPRIT methods [2]. For instance, suppose that $\hat{\boldsymbol{U}}$ is the estimated principal subspace of $\boldsymbol{x}(t)$, the MUSIC method finds the p largest 'peaks' in the following *pseudo spectrum*

$$P(\theta) = \frac{\boldsymbol{a}^{\mathsf{H}}(d,\theta)\boldsymbol{a}(d,\theta)}{\boldsymbol{a}^{\mathsf{H}}(d,\theta)\boldsymbol{U}_{\perp}\boldsymbol{U}_{\perp}^{\mathsf{H}}\boldsymbol{a}(d,\theta)}\,, \tag{62}$$

where $\hat{\boldsymbol{U}}_{\perp}$ is an $N \times (N-p)$ matrix containing $N-p$ orthonormal vectors that are orthogonal to $\hat{\boldsymbol{U}}$ such that $\boldsymbol{U}_{\perp}^{\mathsf{H}}\hat{\boldsymbol{U}} = \boldsymbol{0}$.

In a distributed setting, each element of $\boldsymbol{x}(t)$ corresponds to a scalar signal received by an antenna operated by a sensor in the network. We have $S = N$ such sensors/antennas. The problem of determining $\hat{\boldsymbol{U}}$, the p-D principal subspace of $\boldsymbol{x}(t)$, is thus a *dynamic* PCA problem (4) with DRO data, as the object is moving.

A simulation example of tracking the DoA of a single moving object using an ULA with $S = N = 30$ antennas is shown below. In the example, the antennas/sensors are connected through a mean degree-4 small world graph with rewiring probability 0.2. Each of the antenna

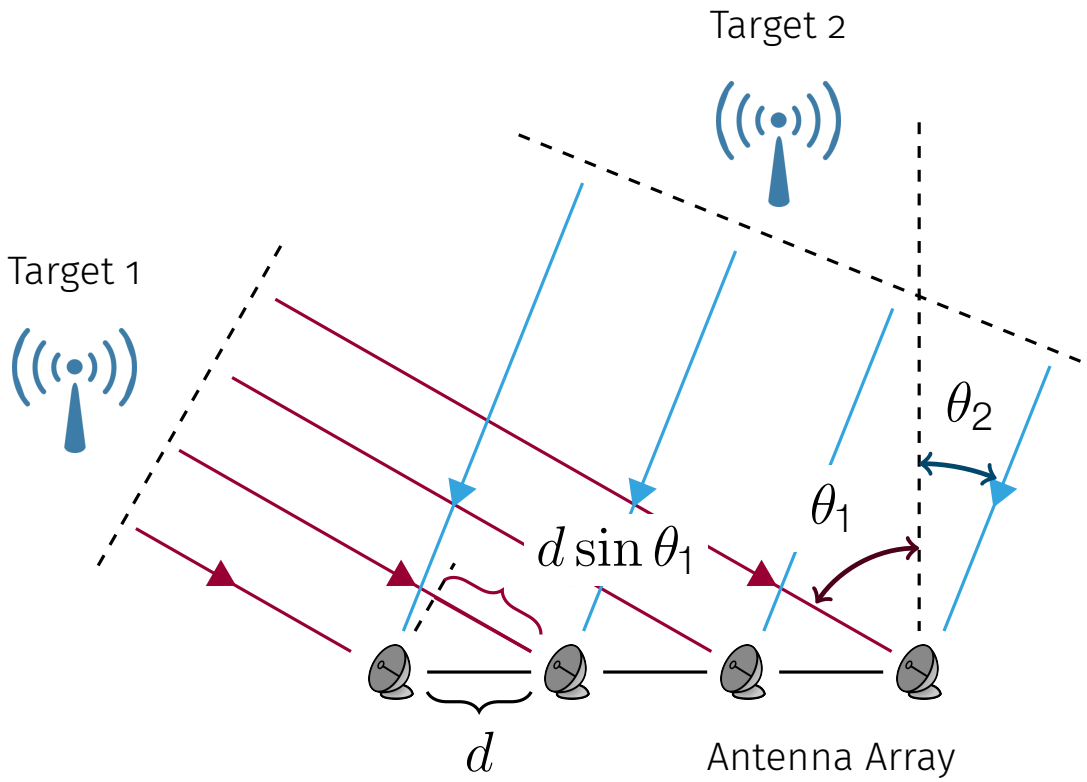

Fig. 9. **Direction of Arrival Estimation**. Our goal is to estimate the directions θ_1, θ_2, ... of the *targets* in the figure from the signals received at the *uniform linear array* (ULA).

receives a total of $T = 1500$ samples and the SNR in (61) is 20dB. The heatmap for evolution of psuedo spectrum and the evolution of normalized objective value (NOV) [cf. (43)], *i.e.,* :

$$\mathsf{NOV}(t) := \frac{\mathbb{E}\|\boldsymbol{x}(t) - \boldsymbol{u}(t)\boldsymbol{u}^{\mathsf{H}}(t)\boldsymbol{x}(t)\|_2^2}{\mathbb{E}\|\boldsymbol{x}(t)\|_2^2}\,, \tag{63}$$

are shown in Fig. 10 and 11, respectively. Once again, we observe that the decentralized approach produces an accurate estimate of the DoA.

3) Distributed Beamforming: A related application is that of distributed PCA methods for *beamforming*, where the aim is to find the weights of the maximal ration combiner, to combine the signal at each antenna element in such a way to maximize the SNR. This problem fits into the setting of a dynamic PCA problem with DRO data as we deploy subarrays of antennas.

Our physical set up is illustrated by Fig. 12, where we consider grouping $N = 256$ antennas into $S = 64$ sub-arrays, each with $N_i = 4$ antennas. The processor units form a mean degree-6 small world graph with rewiring probability of 0.2. Each processor unit receives $T = 1500$

w COPY

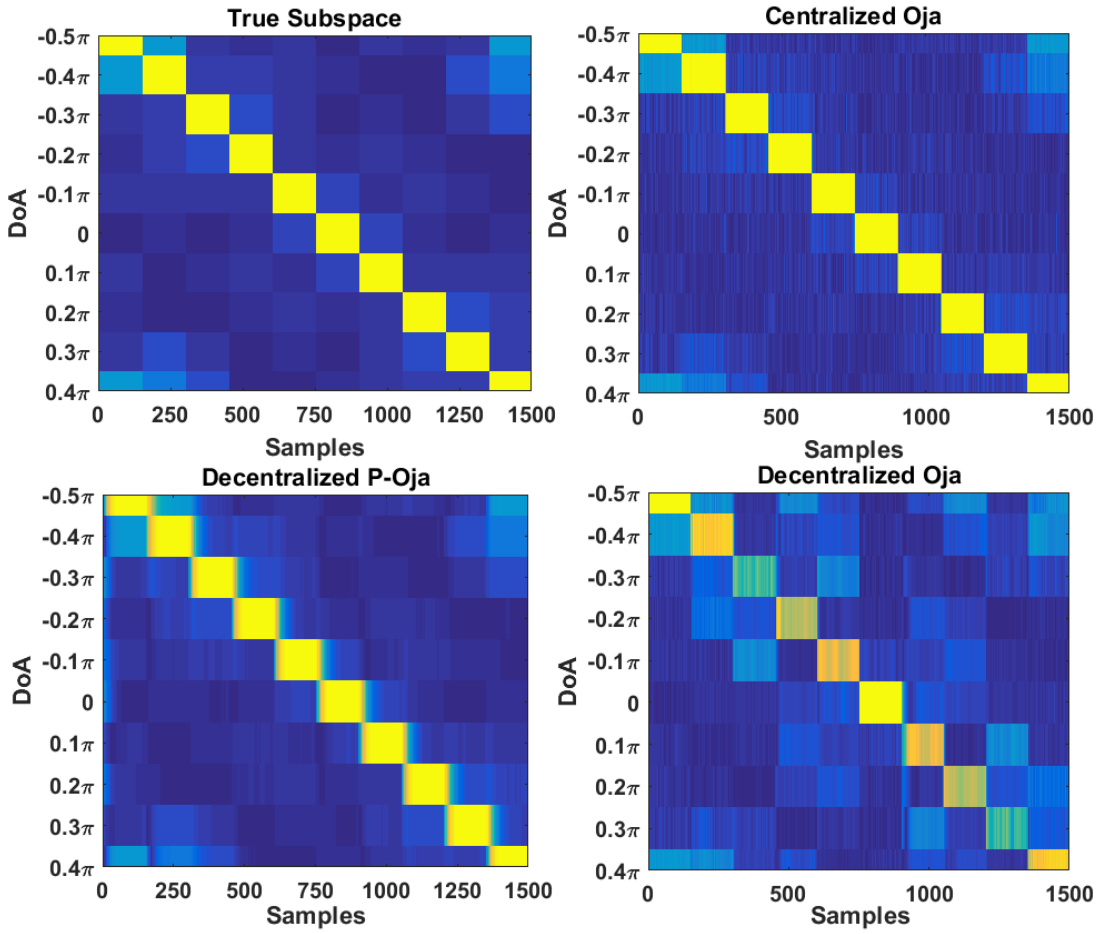

Fig. 10. **DoA tracking with Distributed PCA Methods**. The top-left plot shows the evolution of DoA against time/batch number, as indicated by the yellow dot. The remaining three plots show the evolution of pseudo-spectrum (62) using the Oja, D-Oja and distributed P-Oja methods. Each column in the plot represents a pseudo-spectrum. The P-Oja method is set with the batch size of $B = 5$ with a step size of $\gamma_t = 0.25$ while the Oja's method is set with a step size of $\gamma_t = 0.05$.

samples with an SNR of 20dB. The signal $\boldsymbol{x}(t)$ is generated with $p = 2$ dimensional principal subspace. In Fig. 13, we compare the NOV [cf. (63)] of running the different PCA methods in this example. The numerical results show that the P-Oja method converges faster than the Oja's method in the (quasi)-static setting, and the decentralized algorithms track the subspaces accurately.

E. Other Methods and Summary

In addition to the surveyed methods above, we note that an ADMM based algorithm has been developed in [24]. There, the authors proposed to relax (3) into a matrix factorization problem

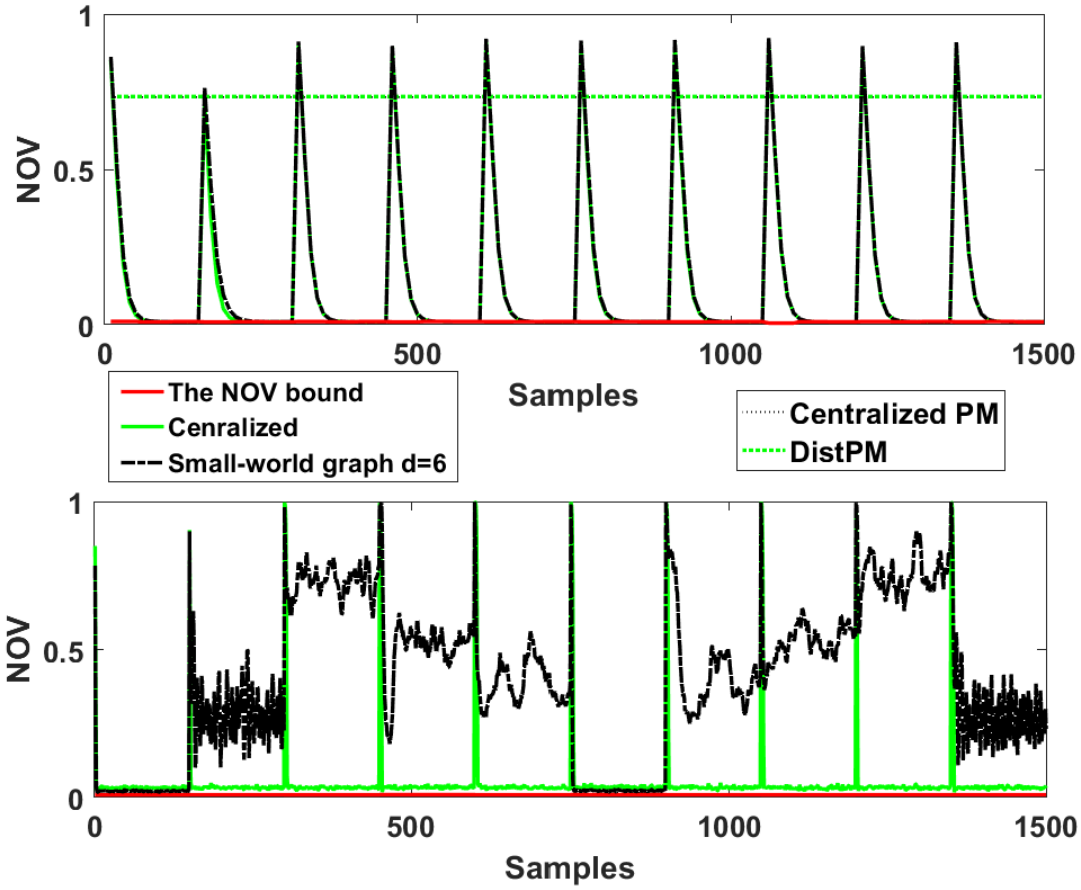

Fig. 11. **Evolution of the NOV against time in DoA tracking**. We apply the centralized and decentralized P-Oja and Oja methods for subspace tracking. The top figure is for the P-Oja method where the batch size is $B = 5$, the power iteration parameter is set as $P = 20$ and a step size $\gamma_t = 0.25$; the bottom figure is the Oja method with a step size $\gamma_t = 0.05$. For both methods, the number of gossip round is $L = 10$.

as:

$$\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p},\boldsymbol{Y}\in\mathbb{C}^{p\times T}} \|\boldsymbol{X}-\boldsymbol{U}\boldsymbol{Y}\|_{\mathrm{F}}^2 \,, \tag{64}$$

where they have relaxed the constraint that $\boldsymbol{Y} = \boldsymbol{U}^{\mathsf{H}}\boldsymbol{X}$. The problem (64) is then written as:

$$\begin{aligned} \min_{\boldsymbol{U}_i\in\mathbb{C}^{N_i\times p},\boldsymbol{Y}_i\in\mathbb{C}^{p\times T},\forall i} \quad & \textstyle\sum_{i=1}^{S} \|\boldsymbol{X}_i^r - \boldsymbol{U}_i\boldsymbol{Y}_i\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{Y}_i = \boldsymbol{Y}_j,\ \forall\ (i,j)\in\mathcal{E}\ . \end{aligned} \tag{65}$$

The optimization problem shown above is then tackled using an alternating minimization approach, in which the update of the coupled variables $\boldsymbol{Y}_i$ is handled by the celebrated ADMM method [42]. See also [27] for a related study.

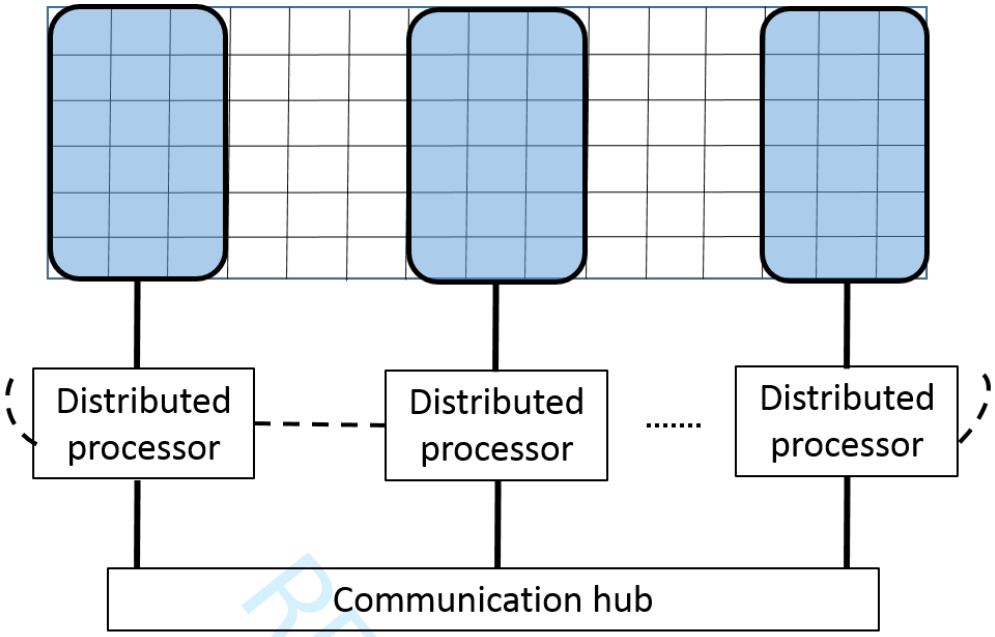

Fig. 12. **Grouping Antennas into Subarrays with Distributed Processors**. Each distributed processor can be treated as a supernode and communicate with the others.

REVIEW COPY

To conclude this section, in Table I, we summarize the PCA strategies described in the above for DRO data by comparing the computation and communication complexities required by each method. Notice that we chose not to compare the ADMM approach in [24] as the latter involves multiple nested loops which hinders a fair comparison. As seen from the comparison, even though the DistPM enjoys the fastest convergence rate, its computation and communication costs are also T times larger than the D-Oja method. To this end, the P-Oja method strikes a balance between the two methods, requiring moderate computation/communication cost and offering faster convergence rate than the D-Oja method.

VI. Related & Open Problems

In this section we first survey a few related problems and the state-of-the-art on how some of these problems have been solved using distributed computations. We then discuss open problems that are relevant to distributed PCA.

1) Distributed Robust PCA: The robust PCA problem [43], [44] deals with scenarios when the observed data is contaminated with outlier noise. In particular, we consider a modified

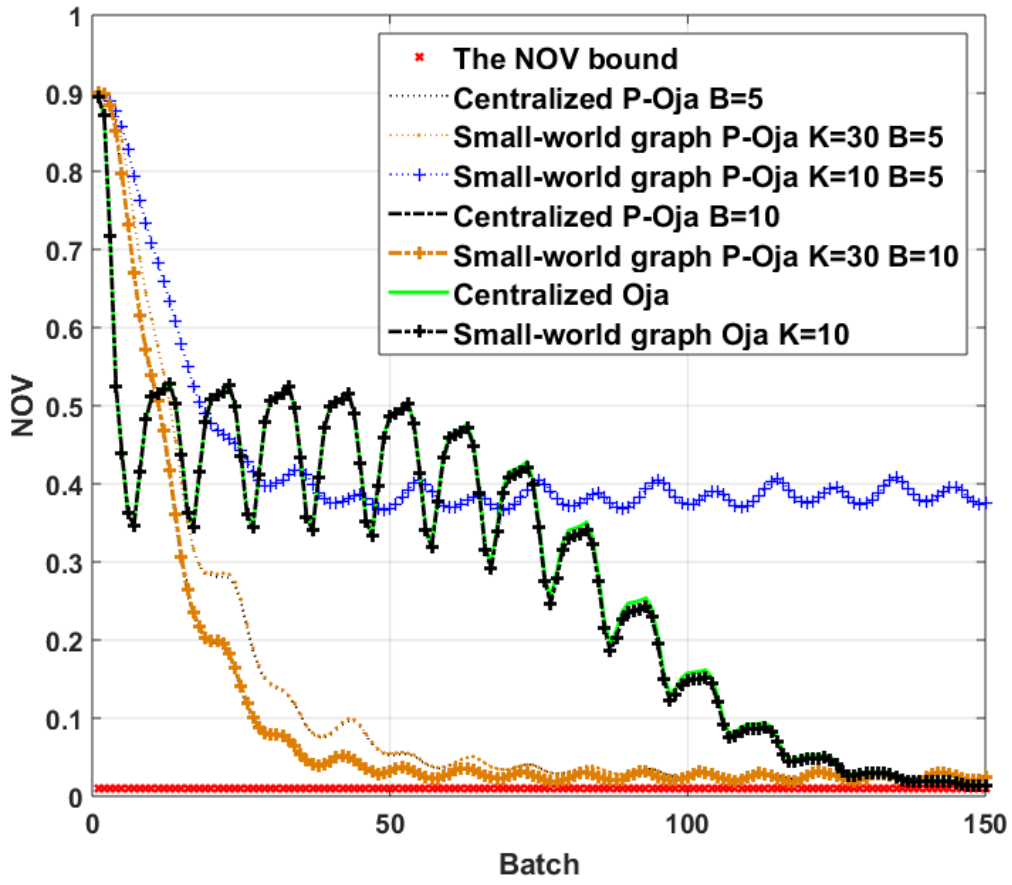

Fig. 13. **Evolution of NOV against time in Distributed Beamforming**. For Oja's and P-Oja method, we set the step size as $\gamma_t = 5 \times 10^{-4}$ and $\gamma_t = 0.01B$, respectively, where B is the batch size used.

observation model from (2):

$$\boldsymbol{x}(t) = \boldsymbol{U}_p \boldsymbol{z}(t) + \boldsymbol{c}(t) + \boldsymbol{e}(t) \ , \tag{66}$$

where $\boldsymbol{c}(t)$ represents the *outlier noise*, e.g., a sparse vector. Such model may be used to handle situations of faulty sensors or anomalies in the observed data. In particular, when $\boldsymbol{c}(t)$ is sufficiently sparse and the subspace dimension is sufficiently low, it is possible to recover the subspace spanned by $\boldsymbol{U}_p$ through solving a convex program [44], [45].

Recently, distributed methods have been developed for the robust PCA problem. For instance, in the DCO setting [46] proposed to tackle a modified problem of (8) with a distributed sub-

Methods	Comp.	Commun.	Convergence
DistPM [18]	$\mathcal{O}(TN_i)$	$\mathcal{O}(TL)$	fast
D-Oja [23]	$\mathcal{O}(N_i)$	$\mathcal{O}(L)$	slow
P-Oja [25]	$\mathcal{O}(PBN_i)$	$\mathcal{O}(PBL)$	medium

TABLE I

COMPARISON OF THE DISTRIBUTED PCA METHODS (DRO). WE FOCUS ON THE COMPUTATION AND COMMUNICATION COMPLEXITIES PER ITERATION UNDER THE SPECIAL CASE OF $p = 1$. NOTE THAT THE COMPUTATION COMPLEXITY IS COUNTED AT A *per-agent* LEVEL. FOR THE P-OJA METHOD, WE ASSUME THAT BATCH SIZE IS CONSTANT SUCH THAT $B = |\mathcal{B}_{k,s}|$ FOR ALL k, s.

gradient descent method. The authors therein essentially solve:

$$\min_{\boldsymbol{U}\in\mathbb{C}^{N\times p}} \sum_{t=1}^{T} \|(\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^{\mathsf{H}})\boldsymbol{x}(t)\|_1 \text{ s.t. } \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}=\boldsymbol{I}\ , \tag{67}$$

where we have replaced the Euclidean norm in (3) by the sparsity-inducing ℓ_1 norm; [47] proposed a detection method for samples that are contaminated by outlier noise and prune away those samples. Notice that these methods are only compatible with the DCO setting. For the DRO case, [48] proposed an ADMM algorithm for the corresponding robust PCA problem.

2) Distributed Canonical Correlation Analysis: As a closely related problem to PCA, the canonical correlation analysis (CCA) problem [49] is widely applied in blind source separation, array processing, medical imaging and word embedding; see [50]–[52] for the applications.

In a nutshell, the CCA problem involves simultaneously analyzing two datasets of *paired* data, $\{\boldsymbol{x}(t)\}_{t=1}^{T}, \{\boldsymbol{y}(t)\}_{t=1}^{T}$, where $\boldsymbol{x}(t)\in\mathbb{C}^{N_1}$, $\boldsymbol{y}(t)\in\mathbb{C}^{N_2}$, in order to find a low-rank structure in the corresponding cross-correlation matrix $\boldsymbol{R}_{xy}:=\mathbb{E}[\boldsymbol{x}(t)\boldsymbol{y}^{\mathsf{H}}(t)]$. Each of the two datasets offers a different *view* of the same latent structure, e.g., $\boldsymbol{x}(t)$ correspond to the spelling features for document t while $\boldsymbol{y}(t)$ correspond to the contextual features. Mathematically, similar to the PCA problem with $p=1$, the CCA problem can be given by:

$$\max_{\boldsymbol{u}_1\in\mathbb{C}^{N_1},\boldsymbol{v}_1\in\mathbb{C}^{N_2}} \frac{\boldsymbol{u}_1^{\mathsf{H}}\mathbb{E}[\boldsymbol{x}(t)\boldsymbol{y}^{\mathsf{H}}(t)]\boldsymbol{v}_1}{\sqrt{\mathbb{E}[|\boldsymbol{u}_1^{\mathsf{H}}\boldsymbol{x}(t)|^2]\cdot\mathbb{E}[|\boldsymbol{v}_1^{\mathsf{H}}\boldsymbol{y}(t)|^2]}}\ . \tag{68}$$

Efficient algorithms for CCA have been developed, e.g., [53]–[55]. However, to the best of our knowledge, the literature on distributed CCA problem is lacking except for a recent work in [56]. Therein, the authors considered a DRO setting and applied the idea of alternating optimization to tackle (68). On the down side, the proposed algorithm only works in a tree-network setting.

REVIEW COPY

3) Distributed Dictionary Learning: The dictionary learning problem [57]–[59] aims at learning an *over-complete dictionary* which can describe the dataset with a sparse linear combination of the atoms. Let $\boldsymbol{X} \in \mathbb{R}^{N\times T}$ be the given dataset, the dictionary learning problem can be formulated as a matrix decomposition problem:

$$\min_{\boldsymbol{D},\boldsymbol{Y}} \ \|\boldsymbol{X}-\boldsymbol{D}\boldsymbol{Y}\|_{\mathrm{F}}^2 \text{ s.t. } \|\boldsymbol{y}_t\|_0 \leq R_0,\ t=1,...,T\ , \tag{69}$$

where the ℓ_0 norm constraint on the columns of $\boldsymbol{Y}$, $\boldsymbol{y}_t$, ensures that the problem finds a *sparse* representation, and we have $\boldsymbol{D} \in \mathbb{R}^{N\times K}, \boldsymbol{Y} \in \mathbb{R}^{K\times T}$ such that the dictionary consists of K atoms. Problem (69) is challenging since both of its objective function and constraint are non-convex. As such, (69) is usually tackled by an alternating optimization procedure which iterates between two steps — a sparse coding step for updating $\boldsymbol{Y}$ and a dictionary learning step for updating $\boldsymbol{D}$ as well as the non-zero elements of $\boldsymbol{Y}$.

To tackle (69) distributively when the data is organized with the DCO structure [cf. (6)] one can follow the cloud K-SVD method in [38] to show how the techniques in distributed PCA can be applied. For the sparse coding step, it is obvious that the problem is decomposable due to the data structure. Let us denote the solution obtained at this step by $\boldsymbol{Y}^{(s)}$. For the dictionary learning step, [38] applies a cyclical update to optimize each atom (column) of $\boldsymbol{D}$ sequentially. For the kth atom, we consider:

$$\begin{aligned}&\min_{\boldsymbol{d}_k\in\mathbb{R}^N} \|\tilde{\boldsymbol{X}}^{(s)}-\boldsymbol{d}_k(\boldsymbol{y}_k^{(s),\mathrm{row}})^\top\|_{\mathrm{F}}^2\\ \Longleftrightarrow &\min_{\boldsymbol{d}_k\in\mathbb{R}^N} \|\tilde{\boldsymbol{X}}_R^{(s)}-\boldsymbol{d}_k(\boldsymbol{y}_{k,R}^{(s),\mathrm{row}})^\top\|_{\mathrm{F}}^2\ ,\end{aligned} \tag{70}$$

where $\tilde{\boldsymbol{X}}^{(s)}$ is obtained by taking the difference between $\boldsymbol{X}$ and the contributions from the atoms other than $\boldsymbol{d}_k$, $\boldsymbol{y}_k^{(s),\mathrm{row}}$ is the kth row of $\boldsymbol{Y}^{(s)}$, $\boldsymbol{y}_{k,R}^{(s),\mathrm{row}}$ is the sub-vector of $\boldsymbol{y}_k^{(s),\mathrm{row}}$ retaining only the non-zero components of the latter, and $\tilde{\boldsymbol{X}}_R^{(s)}$ selects the columns of $\tilde{\boldsymbol{X}}^{(s)}$ with respect to $\boldsymbol{y}_{k,R}^{(s),\mathrm{row}}$.

Importantly, the minimization on the right hand side of (70) can be solved by obtaining the *top* left and right singular vectors of $\tilde{\boldsymbol{X}}_R^{(s)}$. Note that this is precisely the distributed PCA problem with DCO data. As proposed in [38], a DistPM (DCO) method is then applied to compute the updated atom $\boldsymbol{d}_k^\star$ as well as updating the *non-zero* elements in $\boldsymbol{y}_k^{(s),\mathrm{row}}$.

We remark that distributed dictionary learning has also been considered a number of prior works, e.g., [60]–[63] with a different approach towards the solution.

REVIEW COPY

4) Distributed Low-rank Optimization: Distributed PCA methods can also be applied as a subroutine to solve low rank multi-agent optimization problem of the form:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{m_1\times m_2}} \sum_{i=1}^{S} f_i(\boldsymbol{\theta}) \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_* \leq R\,, \tag{71}$$

which includes problems such as matrix completion as special cases [64]. Note $\|\boldsymbol{\theta}\|_*$ denotes the nuclear norm of the matrix $\boldsymbol{\theta}$. Each of $f_i(\boldsymbol{\theta})$ is the loss function attributed to the partition of data available at the ith agent/machine, which is assumed to be continuously differentiable but possibly non-convex. Importantly, in the above, the observed data can be split in an arbitrary fashion.

For large-scale problems with $m_1, m_2 \gg 1$, a popular method is to apply the so-called *projection-free* (a.k.a. Frank-Wolfe) algorithms [65], which amounts to finding the top singular vector of the gradient matrix at each iteration. Obviously, the top singular vector can be found using the PCA methods. Furthermore, as the gradient matrix is simply a sum of the *local* gradients, the associated PCA problem admits a DCO data structure. To this end, [39] applied the DistPM (DCO) with a set of carefully designed, time varying parameters L, K as a subroutine to tackle (71) in a distributed fashion, and the authors showed that the resulting decentralized Frank-Wolfe algorithm [66] converges at desirable rates for both convex and non-convex instances of (71). Similar effort can also be found in [67] for the master-slave architecture.

As an alternative, it is worthwhile mentioning that an heuristic solution to the matrix completion problem [64] (when the loss functions in (71) are squared Euclidean distances between entries of $\boldsymbol{\theta}$ and the observed data) can be found by iteratively computing the top singular vectors from a set of *partially observed* data, following the procedure proposed in [68]. Once again, this procedure can be applied to a distributed setting adopting the distributed PCA methods reviewed in this paper.

A. Open Problems

A problem in which the literature is lacking is in tackling the PCA problem distributively with *irregularly partitioned* data. The data may not conform to neither the DCO nor the DRO type exactly, but be divided in *an arbitrary fashion*. The the resulting PCA problem would essentially require each agent to infer the missing entries simultaneously as it computes the principal components.

REVIEW COPY

Another interesting extension of distributed PCA algorithms, would be methods for solving the *tensor PCA* problem (or tensor decomposition in general) [69], which has rarely been studied in the distributed setting. Note that tensor models are effective in modeling high order relationships between observations, e.g., [70]. Some prior work on parallel tensor decomposition are found in [71]–[73], however, in addition to restricting the types of networks to operate the distributed algorithms on, these work either requires the data to be stored centrally [72], [73], or the same set be available at multiple agents at least in part [71]. These settings are more restrictive than the distributed data setting considered in distributed PCA. A fully distributed tensor PCA method has yet to be developed.

Lastly, thanks to the recent advancements in first order and non-convex optimization methods, a number of new computation techniques have been applied to the PCA problem, e.g., [74]–[76]. These new methods offer theoretically proven improvements and can overcome certain limitations in fast convergence for the traditional power and Oja's methods, which are the backbones for distributed PCA methods discussed in this paper. For instance, [75] proposed an PCA algorithm with a convergence rate that is independent of the spectral gap $\sigma_1(\hat{\boldsymbol{R}}_x) - \sigma_2(\hat{\boldsymbol{R}}_x)$; [76] combines the technique of variance reduction with acceleration to yield better dependence of the convergence rate on spectral gap. The methods above also demonstrate significant speedup in a centralized setting empirically, and it would be beneficial to explore the possibility of extending these benefits to the distributed PCA algorithms.

REVIEW COPY

VII. Conclusions

The distributed PCA methods are motivated the increasing popularity of networked systems including sensor and computer networks. In this paper, we have surveyed recent advancements in distributed PCA methods, where signal processing strategies have been applied independently depending on how the data are acquired in the network. Compared to centralized PCA, these methods can efficiently harness the computation and storage resources at the distributed agents, as confirmed by the theoretical and empirical analysis presented.

References

[1] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE signal processing magazine*, vol. 13, no. 4, pp. 67–94, 1996.

[2] P. Stoica, *Introduction to spectral analysis*. Prentice hall, 1997.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559–572, 1901.

[5] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[6] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 29.

[7] Y. Qu, G. Ostrouchov, N. Samatova, and A. Geist, "Principal component analysis for dimension reduction in massive distributed data sets," in *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 2002.

[8] Y. Liang, M.-F. F. Balcan, V. Kanchanapally, and D. Woodruff, "Improved distributed principal component analysis," in *Advances in Neural Information Processing Systems*, 2014, pp. 3113–3121.

[9] J. Fan, D. Wang, K. Wang, and Z. Zhu, "Distributed estimation of principal eigenspaces," *arXiv preprint arXiv:1702.06488*, 2017.

[10] R. Kannan, S. Vempala, and D. Woodruff, "Principal component analysis and higher correlations for distributed data," in *Conference on Learning Theory*, 2014, pp. 1040–1057.

[11] C. Boutsidis, D. P. Woodruff, and P. Zhong, "Optimal principal component analysis in distributed and streaming models," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 236–249.

[12] D. Garber, O. Shamir, and N. Srebro, "Communication-efficient algorithms for distributed stochastic principal component analysis," in *International Conference on Machine Learning*, 2017, pp. 1203–1212.

[13] Z.-J. Bai, R. H. Chan, and F. T. Luk, "Principal component analysis for distributed data sets with updating," in *International Workshop on Advanced Parallel Processing Technologies*. Springer, 2005, pp. 471–483.

[14] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson, "Distributed clustering using collective principal component analysis," *Knowledge and Information Systems*, vol. 3, no. 4, pp. 422–448, 2001.

[15] H. Qi, T.-W. Wang, and J. D. Birdwell, "Global principal component analysis for dimensionality reduction in distributed data mining," *Statistical data mining and knowledge discovery*, pp. 327–342, 2004.

[16] F. N. Abu-Khzam, N. F. Samatova, G. Ostrouchov, M. A. Langston, and A. Geist, "Distributed dimension reduction algorithms for widely dispersed data," in *IASTED PDCS*, 2002, pp. 167–174.

[17] J. Fellus, D. Picard, and P.-H. Gosselin, "Dimensionality reduction in decentralized networks by gossip aggregation of principal components analyzers," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2014, pp. 171–176.

[18] A. Scaglione, R. Pagliari, and H. Krim, "The decentralized estimation of the sample covariance," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*. IEEE, 2008, pp. 1722–1726.

[19] Y.-A. Le Borgne, S. Raybaud, and G. Bontempi, "Distributed principal component analysis for wireless sensor networks," *Sensors*, vol. 8, no. 8, pp. 4821–4850, 2008.

[20] M. E. Yildiz, F. Ciaramello, and A. Scaglione, "Distributed distance estimation for manifold learning and dimensionality reduction," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2009, pp. 3353–3356.

[21] W. Suleiman, M. Pesavento, and A. M. Zoubir, "Performance analysis of the decentralized eigendecomposition and ESPRIT algorithm," *IEEE Transactions on Signal Processing*, vol. 64, no. 9, pp. 2375–2386, 2016.

[22] S. B. Korada, A. Montanari, and S. Oh, "Gossip PCA," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 2011, pp. 209–220.

[23] L. Li, A. Scaglione, and J. H. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 725–738, 2011.

REVIEW COPY

[24] I. D. Schizas and A. Aduroja, "A distributed framework for dimensionality reduction and denoising," *IEEE Transactions on Signal Processing*, vol. 63, no. 23, pp. 6379–6394, 2015.

[25] S. X. Wu, H.-T. Wai, A. Scaglione, and N. A. Jacklin, "The Power-Oja method for decentralized subspace estimation/tracking," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 3524–3528.

[26] A. Wiesel and A. O. Hero, "Decomposable principal component analysis," *IEEE Transactions on Signal Processing*, vol. 57, no. 11, pp. 4369–4377, 2009.

[27] A. Bertrand and M. Moonen, "Distributed adaptive estimation of covariance matrix eigenvectors in wireless sensor networks with application to distributed pca," *Signal Processing*, vol. 104, pp. 120–135, 2014.

[28] G. H. Golub and C. F. van Loan, *Matrix computations*, 4th ed. Johns Hopkins University Press, Baltimore, MD, 2013.

[29] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Journal of mathematical analysis and applications*, vol. 106, no. 1, pp. 69–84, 1985.

[30] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, M.I.T., Boston, MA, 1984.

[31] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[32] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. John Wiley & Sons, 2004, vol. 46.

[33] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[34] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006.

[35] A. Balsubramani, S. Dasgupta, and Y. Freund, "The fast convergence of incremental PCA," in *NIPS*, 2013.

[36] S. Attallah and K. Abed-Meraim, "Fast algorithms for subspace tracking," *IEEE Signal Processing Letters*, vol. 8, no. 7, pp. 203–206, 2001.

[37] J. H. Manton, I. Y. Mareels, and S. Attallah, "An analysis of the fast subspace tracking algorithm NOja," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 2002, pp. 1101–1104.

[38] H. Raja and W. U. Bajwa, "Cloud k-svd: A collaborative dictionary learning algorithm for big, distributed data," *IEEE Transactions on Signal Processing*, vol. 64, no. 1, pp. 173–188, 2016.

[39] H.-T. Wai, A. Scaglione, J. Lafond, and E. Moulines, "Fast and privacy preserving distributed low-rank regression," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 4451–4455.

[40] M. Hardt and E. Price, "The noisy power method: A meta algorithm with applications," in *Advances in Neural Information Processing Systems*, 2014, pp. 2861–2869.

[41] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845

[42] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[43] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[44] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky, "Rank-sparsity incoherence for matrix decomposition," *SIAM Journal on Optimization*, vol. 21, no. 2, pp. 572–596, 2011.

REVIEW COPY

[45] A. Agarwal, S. Negahban, and M. J. Wainwright, "Noisy matrix decomposition via convex relaxation: Optimal rates in high dimensions," *The Annals of Statistics*, pp. 1171–1197, 2012.

[46] Y. Kopsinis, S. Chouvardas, and S. Theodoridis, "Distributed robust subspace tracking," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*. IEEE, 2015, pp. 2531–2535.

[47] M. Rahmani and G. K. Atia, "A decentralized approach to robust subspace recovery," in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*. IEEE, 2015, pp. 802–807.

[48] M. Mardani, G. Mateos, and G. B. Giannakis, "Decentralized sparsity-regularized rank minimization: Algorithms and applications," *IEEE Transactions on Signal Processing*, vol. 61, no. 21, pp. 5374–5388, 2013.

[49] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3/4, pp. 321–377, 1936.

[50] N. M. Correa, T. Adali, Y.-O. Li, and V. D. Calhoun, "Canonical correlation analysis for data fusion and group inferences," *IEEE signal processing magazine*, vol. 27, no. 4, pp. 39–50, 2010.

[51] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural computation*, vol. 16, no. 12, pp. 2639–2664, 2004.

[52] P. S. Dhillon, D. P. Foster, and L. H. Ungar, "Eigenwords: spectral word embeddings." *Journal of Machine Learning Research*, vol. 16, pp. 3035–3078, 2015.

[53] H. Avron, C. Boutsidis, S. Toledo, and A. Zouzias, "Efficient dimensionality reduction for canonical correlation analysis," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. S111–S131, 2014.

[54] R. Ge, C. Jin, P. Netrapalli, A. Sidford *et al.*, "Efficient algorithms for large-scale generalized eigenvector computation and canonical correlation analysis," in *International Conference on Machine Learning*, 2016, pp. 2741–2750.

[55] X. Fu, K. Huang, M. Hong, N. D. Sidiropoulos, and A. M.-C. So, "Scalable and flexible multiview max-var canonical correlation analysis," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4150–4165, 2017.

[56] A. Bertrand and M. Moonen, "Distributed canonical correlation analysis in wireless sensor networks with application to distributed blind source separation," *IEEE Transactions on Signal Processing*, vol. 63, no. 18, pp. 4800–4813, 2015.

[57] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[58] Q. Zhang and B. Li, "Discriminative K-SVD for dictionary learning in face recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 2691–2698.

[59] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 689–696.

[60] P. Chainais and C. Richard, "Learning a common dictionary over a sensor network," in *IEEE 5th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 2013, pp. 133–136.

[61] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary learning over distributed models," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1001–1016, 2015.

[62] H.-T. Wai, T.-H. Chang, and A. Scaglione, "A consensus-based decentralized algorithm for non-convex optimization with application to dictionary learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 3546–3550.

[63] M. Hong, D. Hajinezhad, and M.-M. Zhao, "Prox-PDA: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks," in *International Conference on Machine Learning*, 2017, pp. 1529–1538.

[64] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, p. 717, 2009.

[65] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *ICML (1)*, 2013, pp. 427–435.

REVIEW COPY

[66] H.-T. Wai, J. Lafond, A. Scaglione, and E. Moulines, "Decentralized Frank–Wolfe algorithm for convex and nonconvex problems," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5522–5537, 2017.

[67] W. Zheng, A. Bellet, and P. Gallinari, "A distributed Frank-Wolfe framework for learning low-rank matrices with the trace norm," *arXiv preprint arXiv:1712.07495*, 2017.

[68] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *Journal of machine learning research*, vol. 11, no. Aug, pp. 2287–2322, 2010.

[69] E. Richard and A. Montanari, "A statistical model for tensor PCA," in *Advances in Neural Information Processing Systems*, 2014, pp. 2897–2905.

[70] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[71] A. L. De Almeida and A. Y. Kibangou, "Distributed large-scale tensor decomposition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 26–30.

[72] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 316–324.

[73] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[74] O. Shamir, "A stochastic PCA and SVD algorithm with an exponential convergence rate," in *ICML*, 2015.

[75] Z. Allen-Zhu and Y. Li, "LazySVD: Even faster SVD decomposition yet without agonizing pain," in *Advances in Neural Information Processing Systems*, 2016, pp. 974–982.

[76] C. De Sa, B. He, I. Mitliagkas, C. Ré, and P. Xu, "Accelerated stochastic power iteration," *arXiv preprint arXiv:1707.02670*, 2017.

REVIEW COPY