Distributed static linear Gaussian models using consensus

Pavle Belanovic , Sergio Valcarcel Macua , Santiago Zazo

A B S T R A C T

Algorithms for distributed agreement are a powerful means for formulating distributed versions of existing centralized algorithms. We present a toolkit for this task and show how it can be used systematically to design fully distributed algorithms for static linear Gaussian models, including principal component analysis, factor analysis, and probabilistic principal component analysis. These algorithms do not rely on a fusion center, require only low-volume local (1-hop neighborhood) communications, and are thus efficient, scalable, and robust. We show how they are also guaranteed to asymptotically converge to the same solution as the corresponding existing centralized algorithms. Finally, we illustrate the functioning of our algorithms on two examples, and examine the inherent cost-performance trade-off.

1. Introduction

There exists a strong trend in modern signal processing research of moving away from classical centralized processing architectures towards fully distributed ones. These new platforms rely on a (possibly large) number of interconnected nodes to perform any given task without relying on any central entity such as a fusion center. This lends them unmatched adaptability, robustness, and fault-tolerance.

However, this shift of paradigm also creates a need to reinvent traditional algorithms, since they are no longer applicable, for the lack of a fusion center. This is done by designing new distributed implementations aimed at such distributed platforms.

Two distinct and equally important parts of any such implementation are the *intra-node* local processing at each node, and the *inter-node* communications which provides coordination. Only a coupled design of these two components assures the emergence of global behavior matching, or at least approximating, that of the original centralized algorithm.

Investigation of such distributed platforms may be motivated by the wish to understand and recreate emergent behavior in large-scale decentralized biological systems, such as groups of animals (e.g. schools of fish), cardiomyocyte cells, or even the nervous system. Also, the same type of algorithm appears in large-scale computer networks, such as wireless sensor networks.

In this work, we will examine the processing and communications (intra- and inter-node) aspects of performing static linear Gaussian models (SLGMs), all in a distributed way based on consensus and gossip algorithms. As we will see, SLGMs include principal component analysis (PCA), as well as two closely related algorithms, factor analysis (FA) and probabilistic PCA (PPCA).

1.1. Related work

PCA is one of the most fundamental and best known feature extraction algorithms (Hotelling, 1933; Jolliffe, 2002; Pearson, 1901), dating back to the 1930s. Since then it has enjoyed tremendous success in many diverse fields, inspiring numerous variations and extensions.

There exist various partially distributed implementations of PCA. They focus on saving part of the multi-hop communication cost by either local computations (Kargupta, Huang, Sivakumar, & Johnson, 2001) or aggregation services (Bai, Chan, & Luk, 2005; Le Borgne, Raybaud, & Bontempi, 2008; Qi & Wang, 2004), but they still rely on a fusion center for merging the local results. In the context of distributed compression and source coding, Gastpar, Dragotti, and Vetterli (2006) proposed a distributed Karhunen–Loéve transform which is posed as an optimization problem, where convergence to the global optimum is, in general, not assured. Our two consensus-based distributed PCA algorithms (Valcarcel Macua, Belanovic, & Zazo, 2010) outperform all the above because they both guarantee convergence, with no fusion center, just by local neighborhood communications.

Meanwhile, computing over networks of processing elements is a potent paradigm, offering robust and scalable implementations for a variety of different algorithms (Bertsekas & Tsitsiklis, 1997). The rich body of knowledge on this topic includes parallel, decentralized, and distributed implementations. Parallel computing focuses on splitting the input data, available at once on an incoming bus, into many processing units, each in charge of processing its own slice of information, and the final results are again gathered at the output. Performing algebraic operations on such platforms is well known (van de Geijn, 1997). On the other hand, when we talk of decentralized processing, the original data is spatially dispersed, i.e. partially available at each node, and some local processing is performed before the intermediate results are passed on to a single fusion center, which produces the final result. An excellent example is given in Tsitsiklis (1993).

Finally, in distributed processing, the spatially dispersed data is processed locally, usually in an iterative way, and the intermediate results are communicated only among neighboring nodes. Hence, no fusion center exists, and the final result is available at all the nodes when the iterative process stops. Consensus algorithms, including gossip, have in recent years provided a powerful tool for distributing existing centralized algorithms. For a comprehensive review of consensus and gossip, the reader is directed to Garin and Schenato (2011) and the references therein. Examples of algorithms distributed using consensus are the Kalman filter (Olfati-Saber, 2005), detection (Bajovic, Jakovetic, Xavier, Sinopoli, & Moura, 2011), clustering (Forero, Cano, & Giannakis, 2011), support vector machines (Forero, Cano, & Giannakis, 2010), linear discriminant analysis (Valcarcel Macua, Belanovic, & Zazo, 2011), and many others. In this work we explore the application of consensus algorithms to SLGMs.

1.2. Contributions

The first contribution we present is a "toolkit": a set of matrix operations useful in distributing existing algorithms over networks of nodes. These operations are based on the well-studied average consensus algorithms and include the distributed matrix product, distributed least-squares, and distributed estimation of the first two moments of a multi-dimensional data set.

The main contribution of this article is a direct application of the toolkit: a set of fully distributed algorithms to systematically distribute SLGMs. We begin with two distributed algorithms to perform PCA. The first is a *direct method*, deriving local approximations of the sample covariance matrix of the global data set, and hence the dominant eigenvectors and the principal subspace spanned by these. The other is an *iterative method*, based on an expectation maximization (EM) procedure, producing local approximations of the global principal subspace. Both algorithms are guaranteed to asymptotically converge to the centralized solution given by classical PCA.

In addition, both algorithms are based only on local computations, with strictly limited communications among nodes, only via consensus iterations. These low-volume communications do not grow with the number of data samples and involve only neighboring (1-hop) nodes. Hence, the presented algorithms scale excellently and are applicable to arbitrarily large networks.

We then present two extensions of our iterative algorithm, to distribute FA and PPCA algorithms. Although, as already stated, a multitude of variations of the PCA, PPCA, and FA algorithms exists, here we focus only on their basic forms in order to illustrate our key contribution. The application of this or similar methods to the numerous other variants of each algorithm falls outside the scope of this particular contribution.

Our final contribution are two experimental examples of the use of our algorithms, in distributed scenarios over large, sparse networks, representing the most difficult type of system configuration.

1.3. Outline

We start this article in Section 2 with a brief unifying review of SLGMs. Then in Section 3 we offer a description of the system model we will be considering, followed by an overview of distributed agreement algorithms, focusing on average consensus. These algorithms are summarized in their scalar, vector, and matrix forms.

Based on these well-known algorithms we present our first contribution in Section 4: a toolkit of techniques for distributing algorithms using consensus interactions.

Then, in Section 5, we present two distributed forms of performing PCA based on average consensus; one direct and the other iterative. In the same section we also show two further algorithms, to distributed FA and PPCA, which are extensions of the iterative PCA algorithm.

In order to illustrate the functioning of these algorithms, we show two experiments in Section 6. The first demonstrates the gain achieved by all the nodes through (limited) cooperation via consensus interactions, while the second shows the application of our algorithms in typical real-world scenarios where PCA is known to be useful.

Finally, we conclude the article in Section 7.

2. Static linear Gaussian models

Roweis and Ghahramani (1999) showed how a single mathematical model, and a rather simple and commonly seen one at that, can be used to represent fully many different popular algorithms, such as PCA, FA, PPCA, Independent Component Analysis (ICA), Hidden Markov Models (HMM) and the Kalman filter, among others.

The mathematical model is that of a hidden system being imperfectly observed. It is composed of two rather generic, linear, discrete-time, difference equations for the *state* and the *observation*. In the first, the hidden system progresses through a number of states, $\mathbf{x}$, as governed by the state transition matrix $\mathbf{A}$, and affected by state noise $\mathbf{w}_\bullet \sim \mathcal{N}(0, \mathbf{Q})$.

$$\mathbf{x}[k] = \mathbf{A}\mathbf{x}[k-1] + \mathbf{w}_\bullet. \quad (1)$$

In the second equation, noisy observations $\mathbf{y}$ are produced from the system state through an observation matrix $\mathbf{C}$, and are also affected by observation noise $\mathbf{v}_\bullet \sim \mathcal{N}(0, \mathbf{R})$.

$$\mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{v}_\bullet. \quad (2)$$

We note that both noise variables are represented without a time index k, to emphasize the fact that their realizations are iid, i.e. not to be seen as a sequence.

For obvious reasons the term *linear Gaussian models* is used to refer to all the models united under this umbrella. A particular subset of these are the static linear Gaussian models (SLGMs), in which $\mathbf{A} = \mathbf{0}$, so that (1) reduces to

$$\mathbf{x}_\bullet = \mathbf{w}_\bullet \quad (3)$$

and (2) becomes

$$\mathbf{y}_\bullet = \mathbf{C}\mathbf{x}_\bullet + \mathbf{v}_\bullet. \quad (4)$$

In other words, in these models the time index is lost, as all the states $\mathbf{x}_\bullet$ (and consequently the observations $\mathbf{y}_\bullet$ as well) are iid realizations without any particular (temporal) ordering.

The differentiation among the different SLGMs comes from the ways of constraining, or modeling $\mathbf{R}$, the covariance matrix that controls the observation noise. As we will see later, $\mathbf{R}$ may be assumed to vanish (PCA), be a scaled identity matrix (PPCA), or diagonal (FA).

In this paper we present a systematic way of distributing SLGMs using distributed agreement, and present how the particular assumptions on $\mathbf{R}$ of each SLGM affects the distributed algorithms we propose.

3. Algorithms for distributed agreement

As was discussed earlier, consensus and gossip algorithms are relatively new techniques for distributed agreement in networks, useful in deriving global functions using only iterative local interactions. They offer advantages such as excellent scalability, high energy efficiency, and under certain conditions even a guarantee of asymptotic convergence to the same result obtained by a centralized solution. In this section we first define the system model used throughout this paper, followed by a brief introduction to the workings of distributed agreement algorithms.

3.1. Notation

Bold uppercase letters like $\mathbf{A}$ denote matrices, while bold lowercase letters like $\mathbf{x}$ denote column vectors. Consequently, $\mathbf{1}$ is a column vector of all ones, while $\mathbf{I}$ is the identity matrix. Using Matlab-like notation, we say that the scalar element in the xth row and yth column of the matrix $\mathbf{A}$ is a_{xy}, its entire row is $\mathbf{A}_{x,:}$, and entire column is $\mathbf{A}_{:,y}$.

The subscript $\square_n$ is given to local variables at any node n. When a variable converges to the same value in all the nodes (i.e. the network reaches *consensus* on that value), the subscript $\square_*$ denotes that common value. In iterative algorithms, the discrete index $\square[k]$ indicates the kth iteration. We use $\mathcal{N}(\boldsymbol{\mu}, \mathbf{S})$ to denote Gaussian distributed data with mean $\boldsymbol{\mu}$ and covariance $\mathbf{S}$.

3.2. System model

Let us consider a networked system made up of N nodes. Assuming all the connections are bi-directional, we represent this network with an undirected graph $G = \{V, E\}$, where V is the set of N nodes (or vertices) v_n, and E is the set of edges $e_{nm} = e_{mn}$ that connect ordered pairs of nodes (v_n, v_m).

Furthermore, we will assume an unweighted graph, i.e. all the edges in E have equal weights. Thus, we define the binary symmetric adjacency matrix associated with the graph G as $\mathbf{A} = [a_{nm}]_{N\times N}$, where $a_{nm} = 1$ when $e_{nm} \in E$, and $a_{nm} = 0$ otherwise. From this we derive the degree of each node as $d_n = \|v_n\| = \sum_{n=1}^{N} a_{nm} = \sum_{m=1}^{N} a_{nm}$. We further define the symmetric Laplacian matrix associated with G as $\mathbf{L} = [l_{nm}]_{N\times N}$, such that the diagonal entries are $l_{nn} = d_n$ and $l_{nm} = -a_{nm}$ when $n \neq m$.

Let us also assume that each node has associated with it a scalar local parameter $x_n \in \mathbb{R}$. We let the distributed processing occur in discrete time steps k, which we will refer to as *rounds*, starting at $k = 0$. The initial value of the parameter, $x_n[0]$, may be obtained by any process, such as measurement for example. We denote the set of values of this parameter at all the nodes in the network (i.e. the network state) as $\mathbf{z}[k] = [x_1[k], x_2[k], \ldots, x_n[k], \ldots, x_N[k]]^\mathsf{T}$.

Given such a distributed system, we are interested in evaluating a global function $\chi(\mathbf{z}[0]) : \mathbb{R}^N \to \mathbb{R}$ in a distributed manner, i.e. without the need to gather the contents of $\mathbf{z}$ at any central location. Algorithms for distributed agreement such as consensus and gossip are excellent tools for iteratively deriving this function in a distributed manner.

3.3. Consensus and gossip

Hence, at the core of every consensus or gossip algorithm is such a global function $\chi(\mathbf{z}[0])$ which is to be evaluated by the entire network. In the discussion to follow, we will focus on averaging algorithms, where

$$\chi(\mathbf{z}[0]) = N^{-1}\mathbf{1}^\mathsf{T}\mathbf{z}[0] = \frac{1}{N}\sum_{n=1}^{N} x_n[0] = x_*. \tag{5}$$

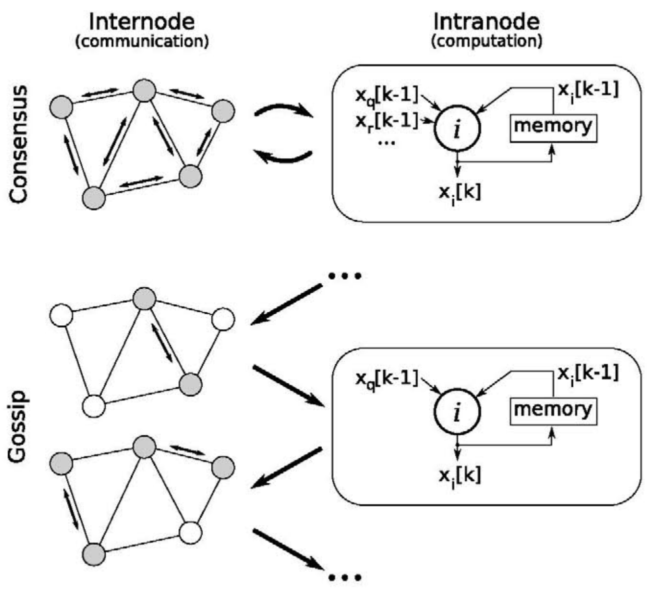


Fig. 1. High-level illustration of the consensus and gossip algorithms, as iterative alternations between internode and intranode processing steps.

In both consensus and gossip, this is achieved by iteratively applying

$$\mathbf{z}[k] = \mathbf{W}[k-1]\mathbf{z}[k-1] \tag{6}$$

where $\mathbf{W}[k] \in \mathbb{R}^{N\times N}$ is a time-varying doubly stochastic weights matrix, associated with the graph G.

Many approaches for forming $\mathbf{W}[k]$ exist. The reader is directed to Garin and Schenato (2011) and Olshevsky and Tsitsiklis (2009) for a thorough review of the various approaches of building the weights, and of consensus and gossip algorithms in general.

Some strategies use time-varying weights, while others have a static matrix $\mathbf{W}[k] = \mathbf{W}$. Some methods are globally optimal (e.g. fastest convergence Xiao and Boyd (2004)), while others are heuristics. Also, some methods are distributed, i.e. use only local information, while others are centralized, i.e. require the knowledge of a global quantity, such as N, or $\mathbf{L}$.

In general, consensus algorithms use a fully populated weights matrix, where every edge in G is represented in $\mathbf{W}[k]$ at every round k. Often the weights matrix will be static, i.e. each edge in G is present in $\mathbf{W}$ and assigned a fixed weight. Gossip algorithms on the other hand use relatively sparse weights matrices that are always dynamic. Typically each $\mathbf{W}[k]$ in a gossip algorithm represents non-zero weighs on a subset of the edges in G. These edges are "active" in this particular round k. At the next round, a different subset of active edges is selected, and so on. These mechanisms are illustrated in Fig. 1, highlighting some of the different ways the internode and intranode stages can be alternated. It is important to note, as previously stated, that many variations on these common themes have been proposed in the literature. Fig. 1 also serves to illustrate how the information originating in each node diffuses throughout the network with each iteration.

As we will see in Sections 4 and 5, the averaging ability of consensus and gossip algorithms will turn out to be a very powerful tool for distributed processing. In the literature there are many neural models performing PCA, either from the Hebbian learning rule or from back-propagation and other least-squares learning rules. The reader is directed to Diamantaras and Kung (1996) for a particularly lucid treatment. Although aesthetically the consensus-based approach shown here resembles a neural network, its functioning is distinct. The key difference is that information in a neural network flows in a very specific direction, from the input to the output, while in consensus information is

Table 1
Scalar, vector, and matrix forms of distributed agreement.

	Scalar	Vector	Matrix
Local parameter	$x_n \in \mathbb{R}$	$\mathbf{x}_n \in \mathbb{R}^P$	$\mathbf{X}_n \in \mathbb{R}^{P\times Q}$
Network state	$\mathbf{z}[k] \in \mathbb{R}^N$	$\mathbf{Z}[k] \in \mathbb{R}^{N\times P}$	$\mathbf{Z}[k] \in \mathbb{R}^{NP\times Q}$
Update, dynamic	$\mathbf{z}[k] = \mathbf{W}[k-1]\mathbf{z}[k-1]$	$\mathbf{Z}[k] = \mathbf{W}[k-1]\mathbf{Z}[k-1]$	$\mathbf{Z}[k] = \widetilde{\mathbf{W}}[k-1]\mathbf{Z}[k-1]$
Update, static	$\mathbf{z}[k] = \mathbf{W}^k\mathbf{z}[0]$	$\mathbf{Z}[k] = \mathbf{W}^k\mathbf{Z}[0]$	$\mathbf{Z}[k] = \widehat{\mathbf{W}}_k\mathbf{Z}[0]$
Global function	$\chi(\mathbf{z}[0]) : \mathbb{R}^N \to \mathbb{R}$	$\chi(\mathbf{Z}[0]) : \mathbb{R}^{N\times P} \to \mathbb{R}^P$	$\chi(\mathbf{Z}[0]) : \mathbb{R}^{NP\times Q} \to \mathbb{R}^{P\times Q}$

diffused in every direction, coming in and out bidirectionally at each link (see Fig. 1). Moreover, in consensus algorithms, every node of the network asymptotically converges to the whole result, while in neural networks each neuron only has part of it. Another effect of diffusion is the reduced dependence on the topology. With consensus algorithms, the only restriction on the topology is that the graph of the network is connected (on average for dynamic topologies), as opposed to classical neural networks that are more sensitive to, and dependent on their hard-wired structure.

3.4. Higher dimensions

It is of course possible to extend the averaging procedure from the scalar case given above, to a vector or a matrix of independent components at each node.

In the vector case, each node starts with a vector of independent components $\mathbf{x}_n[0] \in \mathbb{R}^P$, so that the network state becomes the matrix $\mathbf{Z}[k] \in \mathbb{R}^{N\times P}$. We see that (6) is now given by

$$\mathbf{Z}[k] = \mathbf{W}[k-1]\mathbf{Z}[k-1] \tag{7}$$

and in the static case (6) becomes

$$\mathbf{Z}[k] = \mathbf{W}\mathbf{Z}[k-1] = \mathbf{W}^k\mathbf{Z}[0] \tag{8}$$

and similarly (5) becomes

$$\chi(\mathbf{Z}[0]) = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n[0] = \mathbf{x}_*. \tag{9}$$

The matrix case is only slightly more complicated. Letting each node start with a matrix of independent components $\mathbf{X}_n[0] \in \mathbb{R}^{P\times Q}$, one way of proceeding is to vectorize each node's matrix. Then the nth row of the state matrix becomes

$$\mathbf{Z}[k]_{n,:} = \operatorname{vec}(\mathbf{X}_n[k]^\mathsf{T})^\mathsf{T} \tag{10}$$

and the complete state matrix becomes $\mathbf{Z}[k] \in \mathbb{R}^{N\times PQ}$. This allows us to proceed with (7) or (8) as before. The global function $\chi(\mathbf{Z}[0])$ is then given by

$$\chi(\mathbf{Z}[0]) = \frac{1}{N}\sum_{n=1}^{N}\mathbf{X}_n[0] = \mathbf{X}_*. \tag{11}$$

However, if we wish to observe matrix properties of $\mathbf{X}_n[k]$ as the system iterates, an alternative, but fully equivalent view of matrix consensus exists. It is achieved by stacking the matrices $\mathbf{X}_n[0]$ vertically to obtain the network state matrix $\mathbf{Z}[k] \in \mathbb{R}^{NP\times Q}$.

In the case of a dynamic weights matrix, the update (6) is then given by

$$\mathbf{Z}[k] = \widetilde{\mathbf{W}}[k-1]\mathbf{Z}[k-1] \tag{12}$$

where

$$\widetilde{\mathbf{W}}[k] = \mathbf{W}[k] \otimes \mathbf{I} \in \mathbb{R}^{NP\times NP}. \tag{13}$$

However, if the weights are static, (6) becomes

$$\mathbf{Z}[k] = \widehat{\mathbf{W}}_1\mathbf{Z}[k-1] = \widehat{\mathbf{W}}_k\mathbf{Z}[0] \tag{14}$$

where

$$\widehat{\mathbf{W}}_k = \mathbf{W}^k \otimes \mathbf{I} \in \mathbb{R}^{NP\times NP}. \tag{15}$$

Both (12) and (14) lead to the same global function $\chi(\mathbf{Z}[0])$ as given in (11).

The three forms of distributed agreement are summarized in Table 1.

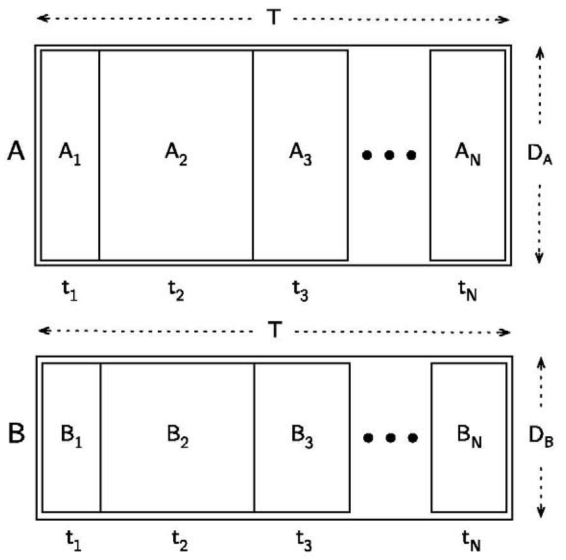


Fig. 2. Global matrices **A** and **B**, each with a total of T samples, distributed unevenly over N nodes.

4. Distributed toolkit

Given the basis described in Section 3, further more complex tools can be derived. Here we focus solely on the tools derived from the arithmetic average algorithms, be they consensus or gossip based. A later section will then show how these tools can in turn be used to distribute applied algorithms such as PCA in an elegant and systematic way.

4.1. Distributed matrix product operator

Let us assume the existence of two *global* and *distributed* matrices **A** and **B**, such that their contents are not available at any one location in the system, but rather are partially available at each node. This is depicted in Fig. 2, where each node n has the knowledge of a block of t_n vectors of the matrix **A**, and the *corresponding* block of the matrix **B**. We note that **A** and **B** share a dimension, $T = \sum_{n=1}^{N} t_n$, with the other dimensions D_A and D_B generally not being the same.

Should we desire to calculate the product $\mathbf{A}\mathbf{B}^\mathsf{T}$, we would normally have to resort to a traditional centralized solution. This involves gathering both matrices at a fusion center, calculating the result, and distributing it to all the nodes.

However, this product can also be approximated arbitrarily closely in a distributed way, by letting each node calculate the local product

$$\mathbf{X}_n[0] = \mathbf{A}_n\mathbf{B}_n^\mathsf{T}. \tag{16}$$

By iteratively applying (12) or (14), we see from (11) that

$$\lim_{k\to\infty}\mathbf{X}_n[k] = \mathbf{X}_* = \frac{1}{N}\sum_{n=1}^{N}\mathbf{X}_n[0] \quad \forall n. \tag{17}$$

Hence, we define the distributed matrix product operator $\bar{*}$

$$\mathbf{A} \bar{*} \mathbf{B}^{\mathrm{T}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{A}_n \mathbf{B}_n^{\mathrm{T}} = \frac{1}{N} \mathbf{A} \mathbf{B}^{\mathrm{T}}. \tag{18}$$

In other words, we are able to guarantee asymptotic convergence of all the nodes in the system to a scaled version of the desired product. Although the appearance of the scaling factor N^{-1} may be undesirable in some situations, we will see how it can be removed in some expressions of great practical importance that are derived from (18).

It is of course also possible to evaluate the product $\mathbf{A} \bar{*} \mathbf{A}^{\mathrm{T}}$. We note that while the residual factor of N^{-1} does scale the eigenvalues of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$, it neither changes their order, nor affects the associated eigenvectors in any way. Hence, the distributed matrix product operator may be useful in eigenanalysis applications.

Furthermore, the dimensionality of the variables that are exchanged during local interactions ($\mathbf{X}_n[k]$) is relatively low. Typically, the dimension shared by $\mathbf{A}$ and $\mathbf{B}$, being t_n and T, is much larger than both D_A and D_B, especially in large scale systems, where N is large.

4.2. Least-squares

Given a linear system of equations $\mathbf{A}\mathbf{Q} = \mathbf{B}$, the least squares solution for $\mathbf{Q}$ is given by $\widehat{\mathbf{Q}} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{B}$. We can immediately split the pseudo-inverse into two factors

$$\boldsymbol{\delta} = \mathbf{A}^{\mathrm{T}}\mathbf{B} \tag{19}$$

$$\boldsymbol{\gamma} = \mathbf{A}^{\mathrm{T}}\mathbf{A}. \tag{20}$$

By using the distributed matrix product operator in (18) we can easily approximate, in a distributed fashion, both averaged factors

$$\mathbf{A}^{\mathrm{T}} \bar{*} \mathbf{B} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\delta}_n = \frac{1}{N} \boldsymbol{\delta} = \boldsymbol{\delta}_* \tag{21}$$

$$\mathbf{A}^{\mathrm{T}} \bar{*} \mathbf{A} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\gamma}_n = \frac{1}{N} \boldsymbol{\gamma} = \boldsymbol{\gamma}_*. \tag{22}$$

Moreover, since (22) has to be inverted, the scaling factors cancel out, and hence the required least squares solution of $\mathbf{Q}$ is immediately achieved in a distributed way

$$\widehat{\mathbf{Q}}_* = [\mathbf{A}^{\mathrm{T}} \bar{*} \mathbf{A}]^{-1} (\mathbf{A}^{\mathrm{T}} \bar{*} \mathbf{B}) = \boldsymbol{\gamma}^{-1}\boldsymbol{\delta} = \widehat{\mathbf{Q}}. \tag{23}$$

4.3. Estimation of the first and second moments

Let us assume that the matrix $\mathbf{A}$ contains a distributed sample set of T vector samples, as depicted in Fig. 2. Letting $\mathbf{a}_n^t = (\mathbf{A}_n)_{:,t}$ be the tth sample at node n, we remember that each node n has t_n samples. It is then possible for all the nodes to calculate the average number of samples,

$$t_* = \frac{1}{N} \sum_{n=1}^{N} t_n \tag{24}$$

by simply performing average consensus or gossip on the parameter t_n itself. We note that $Nt_* = T$.

From here, each node can derive in isolation (i.e. without any communications) the local *globally weighted* mean

$$\boldsymbol{\mu}_n = \frac{1}{t_*} \sum_{t=1}^{t_n} \mathbf{a}_n^t. \tag{25}$$

The global mean of the sample set $\mathbf{A}$ can be estimated by performing average consensus or gossip on these local means as

$$\boldsymbol{\mu}_* = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\mu}_n = \frac{1}{T} \sum_{n=1}^{N} \sum_{t=1}^{t_n} \mathbf{a}_n^t = \boldsymbol{\mu}(\mathbf{A}). \tag{26}$$

It is important to note that $\boldsymbol{\mu}_*$ can be used to center the data set as

$$\mathbf{A}_{\circ} = \mathbf{A} - \boldsymbol{\mu}_* \mathbf{1}^{\mathrm{T}}. \tag{27}$$

In fact, in the rest of this paper, we will assume that the data sets are centered, using the procedure (24)–(27) above, i.e. $\mathbf{A} = \mathbf{A}_{\circ}$.

In the final step, we note that the sample covariance matrix $\mathbf{S}$ can be derived in a distributed manner by

$$\mathbf{S}_* = \frac{1}{t_*} \mathbf{A} \bar{*} \mathbf{A}^{\mathrm{T}} = \frac{1}{T} \mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{S}. \tag{28}$$

Hence, it is possible to derive the first two moments of a distributed data set by performing only three averaging consensus algorithms. In particular the distributed derivation of the sample covariance matrices will play a significant role in the presented algorithms, as we will see in Section 5.

One critical property of the estimates of the sample covariance matrices derived by (28) is their positive-definiteness. We must recall at this point that the distributed derivation of $\mathbf{S}_*$ is in fact an iterative procedure, whereby each node starts at

$$\mathbf{S}_n[0] = \frac{1}{t_*} \mathbf{A}_n \mathbf{A}_n^{\mathrm{T}} \tag{29}$$

and converges towards

$$\lim_{k \to \infty} \mathbf{S}_n[k] = \mathbf{S}_* = \mathbf{S} \quad \forall n. \tag{30}$$

Although we see that $\mathbf{S}_n[k] \succ 0$ for $k \in \{0, \infty\}$ in (29) and (30) respectively, it is not immediately obvious that this property also holds for all integers $k \in [1, \infty)$, which is of course the range of k that is of greatest practical importance.

Let us recall that in the case of a dynamic weights matrix, (12) and (13) tell us that

$$\mathbf{Z}[1] = \widetilde{\mathbf{W}}[0]\mathbf{Z}[0] = (\mathbf{W}[0] \otimes \mathbf{I})\mathbf{Z}[0] \tag{31}$$

where $\mathbf{Z}[0]$ is a stack of local covariance matrices $\mathbf{S}_n[0]$ from (29), and $\mathbf{Z}[1]$ is a stack of the distributed estimates of the global covariance after the 1st iteration. Since each sub-block of $\widetilde{\mathbf{W}}[0]$ is just a scaled identity matrix, $(\mathbf{W}[0])_{i,j}\mathbf{I}$, we see that

$$\mathbf{S}_n[1] = \sum_{i=0}^{N} (\mathbf{W}[0])_{n,i} \mathbf{S}_i[0]. \tag{32}$$

We now recall from (29) that all $\mathbf{S}_n[0] \succ 0$. Also, since all $\mathbf{W}[k]$ are doubly stochastic matrices, all elements $(\mathbf{W}[0])_{i,j} \geq 0$. Therefore, from (32) we stay in the positive semi-definite cone $\mathbf{S}_n[1] \succ 0$, $\forall n$. The same argument can be extended for all further iterations beyond the first, and hence $\mathbf{S}_n[k] \succ 0$, $\forall n, k$.

Things are even simpler in the case of a static weights matrix. We see from (14) and (15) that

$$\mathbf{Z}[k] = \widehat{\mathbf{W}}_k \mathbf{Z}[0] = (\mathbf{W}^k \otimes \mathbf{I})\mathbf{Z}[0] \tag{33}$$

where $\mathbf{Z}[k]$ is a stack of the distributed estimates of the global covariance at the kth iteration. Since each sub-block of $\widehat{\mathbf{W}}_k$ is again a scaled identity matrix, $(\mathbf{W}^k)_{i,j}\mathbf{I}$, we see that

$$\mathbf{S}_n[k] = \sum_{i=0}^{N} (\mathbf{W}^k)_{n,i} \mathbf{S}_i[0]. \tag{34}$$

In other words, all the distributed estimates of the global covariance matrix are always just a weighted sum of all the initial local covariance matrices. The same as before, since $\mathbf{W}$ is a doubly stochastic matrix, so is $\mathbf{W}^k$, and thus $(\mathbf{W}^k)_{i,j} \geq 0$. Therefore, from (34) we have that $\mathbf{S}_n[k] \succ 0$, $\forall n, k$, as desired.

5. Distributed algorithms for SLGM

Provided the toolkit presented in Section 4, we are now ready to introduce two distributed algorithms for computing PCA. Following this, we extend the same argument to construct distributed algorithms for FA and PPCA.

5.1. Distributed PCA from consensus in the covariance matrix

The standard centralized PCA (computed in a fusion center) requires all the data samples, from all the nodes, to be gathered at the fusion center, assuming there is one in the network. With this access to the full data set $\mathbf{Y} \in \mathbb{R}^{D\times T}$, the fusion center can obtain the principal subspace $\mathbf{C} \in \mathbb{R}^{D\times P}$ $(P \leq D)$ directly by taking the P dominant eigenvectors of the sample data covariance matrix $\mathbf{S} = T^{-1}\mathbf{Y}\mathbf{Y}^{\mathsf{T}}$.

To distribute the same method, every node should first obtain the global mean (26) and remove it from its local data. Then, using (28), each node can start with its local, globally weighted covariance matrix $\mathbf{S}_n[0] = t_*^{-1}\mathbf{Y}_n\mathbf{Y}_n$ and proceed iteratively applying (12) or (14) until asymptotically converging to the global data covariance matrix $\mathbf{S}_* = t_*^{-1}\mathbf{Y} \bar{*} \mathbf{Y}^{\mathsf{T}} = \mathbf{S}$.

Finally, once the global covariance matrix is available, each node can locally obtain its own approximation to the global PCA as the subspace spanned by the dominant eigenvectors of its estimate of the global covariance. This subspace will be arbitrarily close, depending on the number of consensus iterations, to that spanned by the dominant eigenvectors of $\mathbf{S}$. This is the base of Algorithm 1, denoted Consensus Based Distributed PCA (CB-DPCA).

Algorithm 1 Consensus Based Distributed PCA (CB-DPCA)

```
1: INPUT $t_n$, $\mathbf{Y}_n$
2: $t_* \leftarrow N^{-1}\sum_{n=1}^{N} t_n$                      $\Longleftarrow$ consensus loop
3: $\mathbf{a}_n \leftarrow t_*^{-1}\sum_{j=1}^{t_n} \mathbf{y}_n^j$
4: $\boldsymbol{\mu}_* \leftarrow N^{-1}\sum_{n=1}^{N} \mathbf{a}_n$                 $\Longleftarrow$ consensus loop
5: $\overline{\mathbf{Y}}_n \leftarrow \mathbf{Y}_n - \boldsymbol{\mu}_*$
6: $\mathbf{S}_* \leftarrow t_*^{-1}\overline{\mathbf{Y}} \bar{*} \overline{\mathbf{Y}}^{\mathsf{T}}$                        $\Longleftarrow$ consensus loop
7: $\mathbf{C}_n \leftarrow P$ dominant eigenvectors of $\mathbf{S}_*$
8: OUTPUT $\mathbf{C}_n$
```

5.2. Distributed PCA as an expectation maximization algorithm

In many real applications (like ambient intelligence, with cheap sensors ubiquitously deployed) nodes will likely offer low storing and processing resources; at least compared with the amount of continuously collected data that they may have to manage. Hence, an iterative algorithm to compute $\mathbf{C}$ is desirable.

Roweis (1998) and Tipping and Bishop (1999) proposed a view where the observed data is the projection of lower-dimensional underlying latent data. Specifically, we let

$$\mathbf{Y} = \mathbf{C}\mathbf{X} + \boldsymbol{\varepsilon} \tag{35}$$

where $\mathbf{Y} \sim \mathcal{N}(0, \mathbf{C}\mathbf{C}^{\mathsf{T}} + \mathbf{R}) \in \mathbb{R}^{D\times T}$ is the observed data, $\mathbf{X} \sim \mathcal{N}(0, \mathbf{I}) \in \mathbb{R}^{P\times T}$ is the latent data set, and $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{R})$ is the observation noise. Then, the conditional distribution of the observed data, conditioned on the value of the latent variable, is again Gaussian, of the form $p(\mathbf{Y}|\mathbf{X}) = \mathcal{N}(\mathbf{Y}|\mathbf{C}\mathbf{X}, \mathbf{R})$. Taking the limit of zero observation noise and assuming independent data points results in the standard PCA model $\mathbf{Y} = \mathbf{C}\mathbf{X}$, where the maximum likelihood estimate for $\mathbf{C}$ spans the P-dimensional subspace which maximizes the variance of the projection of $\mathbf{Y}$ onto that space.

Based on this probabilistic latent variable model, Roweis (1998) and Tipping and Bishop (1999) also showed an iterative EM algorithm to obtain the principal components, without explicitly computing the sample covariance matrix. In each iteration, the *E-step* projects the data onto a lower dimensional subspace, while the *M-step* seeks for an update of that subspace which could minimize the mean square distance between the original and the projected data (i.e. the reconstruction error). Thus, in the kth iteration, the *E-step* is given by

$$\mathbf{X}[k] = (\mathbf{C}[k-1]^{\mathsf{T}}\mathbf{C}[k-1])^{-1}\mathbf{C}[k-1]^{\mathsf{T}}\mathbf{Y} \tag{36}$$

while the *M-step* is given by

$$\mathbf{C}[k] = \mathbf{Y}\mathbf{X}[k]^{\mathsf{T}}(\mathbf{X}[k]\mathbf{X}[k]^{\mathsf{T}})^{-1}. \tag{37}$$

This algorithm will converge to a unique local and global maximum, or to a saddle point if the initial $\mathbf{C}[0]$ is rank deficient. De la Torre (2009) showed this *EM* algorithm to be equivalent to a block coordinate descent algorithm, used for least squares regression, whose convergence has already been analyzed in Baldi and Hornik (1989).

It is important to note that, after centering the dataset, just as all the local measurements $\mathbf{Y}_n$ are independent of each other, but together form the global data set $\mathbf{Y}$, so are the local projections $\mathbf{X}_n$, in turn forming the global projected data set $\mathbf{X}$. In other words, $\mathbf{X}$ has the same block-wise structure depicted in Fig. 2. Hence,

$$\mathbf{Y}_n = \mathbf{C}\mathbf{X}_n \tag{38}$$

so that given the global projection matrix $\mathbf{C}$, each node can in isolation derive its part of the global projected data set.

In order to distribute the *EM* algorithm for PCA (Eqs. (36) and (37)), we first note that, assuming the same $\mathbf{C}_*[k-1]$ is available at all the nodes at the start of the iteration k, each node can carry out the *E-step* in isolation and derive its own portion of the projections $\mathbf{X}_n[k]$.

For the *M-step*, we can define the local variables $\boldsymbol{\delta}_n = \mathbf{Y}_n\mathbf{X}_n[k]^{\mathsf{T}}$ and $\boldsymbol{\gamma}_n = \mathbf{X}_n[k]\mathbf{X}_n[k]^{\mathsf{T}}$ as inputs into an embedded consensus loop, whose results are $\boldsymbol{\delta}_* = \mathbf{Y} \bar{*} \mathbf{X}[k]^{\mathsf{T}}$ and $\boldsymbol{\gamma}_* = \mathbf{X}[k] \bar{*} \mathbf{X}[k]^{\mathsf{T}}$. These outputs asymptotically approach the same values in all the nodes in the network. Indeed, as in consensus least squares (23), the nodes then locally compute

$$\mathbf{C}_*[k] = \boldsymbol{\delta}_*(\boldsymbol{\gamma}_*)^{-1} \tag{39}$$

which in each node asymptotically approaches that calculated by (37).

Eqs. (38) and (39) are the base of the proposed **Consensus Based EM Distributed PCA** (CB-EM-DPCA), as it is shown in Algorithm 2, which asymptotically matches the results of both the *E* and *M* steps of the centralized *EM* algorithm for PCA at every iteration.

Algorithm 2 Consensus Based EM Distributed PCA (CB-EM-DPCA)

```
1: INPUT $t_n$, $\mathbf{Y}_n$
2: $t_* \leftarrow N^{-1}\sum_{n=1}^{N} t_n$                      $\Longleftarrow$ consensus loop
3: $\mathbf{a}_n \leftarrow t_*^{-1}\sum_{j=1}^{t_n} \mathbf{y}_n^j$
4: $\boldsymbol{\mu}_* \leftarrow N^{-1}\sum_{n=1}^{N} \mathbf{a}_n$                 $\Longleftarrow$ consensus loop
5: $\overline{\mathbf{Y}}_n \leftarrow \mathbf{Y}_n - \boldsymbol{\mu}_*$
6: $\mathbf{C}_n[0] \leftarrow P$ random columns
7: repeat
8:     $\boldsymbol{\beta}_* \leftarrow (\mathbf{C}_*[k-1]^{\mathsf{T}}\mathbf{C}_*[k-1])^{-1}\mathbf{C}_*[k-1]^{\mathsf{T}}$
9:     $\mathbf{X}_n[k] \leftarrow \boldsymbol{\beta}_*\overline{\mathbf{Y}}_n$
10:    $\boldsymbol{\delta}_* \leftarrow \overline{\mathbf{Y}} \bar{*} \mathbf{X}[k]^{\mathsf{T}}$                   $\Longleftarrow$ consensus loop
11:    $\boldsymbol{\gamma}_* \leftarrow \mathbf{X}[k] \bar{*} \mathbf{X}[k]^{\mathsf{T}}$                 $\Longleftarrow$ consensus loop
12:    $\mathbf{C}_*[k] \leftarrow \boldsymbol{\delta}_*(\boldsymbol{\gamma}_*)^{-1}$
13: until the desired convergence criterion for $\mathbf{C}$ is met
14: OUTPUT $\mathbf{C}_n$
```

It is worth noting that both the columns of $\mathbf{C}$ and the dominant P eigenvectors of the sample covariance matrix span the same principal subspace. However, they are not equal. In fact the columns of $\mathbf{C}$ are not orthogonal. In the case orthogonality is required, Ahn, Oh, and Choi (2007) showed that adding very simple lower and upper triangularization operators to the E and M step, respectively, the *EM* loop will output directly an orthogonal basis of the principal subspace, similar (up to rotation) to the one found in Algorithm 1. These operators could also be locally computed in each node.

5.3. Factor analysis

By restricting $\mathbf{R}$, known as the uniqueness matrix in this context, to be diagonal, a standard statistical model known as maximum likelihood factor analysis is recovered (for a detailed review on FA see e.g. Reyment and Jöreskog (1996)).

The main difference between FA and PCA is that, instead of assuming that most of the total variance of a variable is important and in common with other observed variables, FA allows for a considerable amount of "uniqueness" to be present in the data. Hence, it takes into account that part of each variable that takes part in correlation with other variables.

The learning algorithm for $\mathbf{C}$, known as the factor loading matrix, and $\mathbf{R}$ is in fact an EM algorithm. In each iteration, during the *E-step*, the *factors* (i.e. latent data) are obtained as the expectation $\mathbf{X}$ and covariance $\mathbf{V}$ of their posterior distribution given the observed data, using parameters of the last iteration (Roweis & Ghahramani, 1999). Thus, defining $\boldsymbol{\beta} = \mathbf{C}[k-1]^\top(\mathbf{C}[k-1]\mathbf{C}[k-1]^\top + \mathbf{R}[k-1])^{-1}$, we get

$$\mathbf{X} = \boldsymbol{\beta}\mathbf{Y} \tag{40}$$

and

$$\mathbf{V}[k] = \mathbf{I} - \boldsymbol{\beta}\mathbf{C}[k-1]. \tag{41}$$

During the *M-step*, the maximization of the expected log likelihood over the new latent data $\mathbf{X}[k]$ yields the new parameter values $\mathbf{C}[k]$ and $\mathbf{R}[k]$. Thus, defining $\boldsymbol{\delta} = \mathbf{Y}\mathbf{X}[k]^\top$ and $\boldsymbol{\gamma} = \mathbf{X}[k]\mathbf{X}[k]^\top + T\mathbf{V}[k]$, we get

$$\mathbf{C}[k] = \boldsymbol{\delta}\boldsymbol{\gamma}^{-1} \tag{42}$$

and

$$\mathbf{R}[k] = \operatorname{diag}\left[\frac{1}{T}(\mathbf{Y}\mathbf{Y}^\top - \mathbf{C}[k]\boldsymbol{\delta}^\top)\right]. \tag{43}$$

To distribute this EM algorithm for FA, we follow the same approach as in CB-EM-DPCA (Algorithm 2). Assuming the same $\mathbf{C}_*[k-1]$ and $\mathbf{R}_*[k-1]$ are available at all the nodes at the start of every new iteration k, then the *E-step* can be performed locally, because $\mathbf{X}_n[k]$ and $\mathbf{V}_n[k]$ only depend on local or shared quantities.

However, the *M-step* involves inter-node communications. Again, we split $\mathbf{C}_n[k]$ into terms $\boldsymbol{\delta}_n = \mathbf{Y}_n[k]\mathbf{X}_n[k]^\top$ and $\boldsymbol{\gamma}_n = \mathbf{X}_n[k]\mathbf{X}_n[k]^\top + \mathbf{V}_n[k]$. After running a consensus loop on the former and on the first term of the latter, all the nodes will agree on $\boldsymbol{\delta}_*$ and an intermediate result, $\boldsymbol{\eta}_* = \mathbf{X}[k] \bar{*} \mathbf{X}[k]^\top$. Since $\mathbf{V}_n[k]$ depends only on shared quantities, all the nodes will agree on $\mathbf{V}_*[k]$ too, as well as on $\boldsymbol{\gamma}_*$.

Although both $\boldsymbol{\delta}_*$ and $\boldsymbol{\eta}_*$ are weighted by the total number of samples T, the locally computed $\mathbf{V}_*[k]$ needs to be multiplied by the same factor. Therefore, T disappears from the equation, and every node will approximate the same value $\mathbf{C}_*[k]$, arbitrarily close to the centralized implementation (42).

$$\begin{aligned}\mathbf{C}_*[k] &= \boldsymbol{\delta}_*(\boldsymbol{\gamma}_*)^{-1}\\ &= \frac{1}{T}\mathbf{Y}\mathbf{X}[k]^\top\left(\frac{1}{T}\mathbf{X}[k]\mathbf{X}[k]^\top + \mathbf{V}[k]\right)^{-1}\\ &= \mathbf{Y}\mathbf{X}[k]^\top(\mathbf{X}[k]\mathbf{X}[k]^\top + T\mathbf{V}[k])^{-1}\\ &= \mathbf{C}[k].\end{aligned} \tag{44}$$

Finally, to distribute (43), let us define $\mathbf{R}_n[k] = \operatorname{diag}[\mathbf{S}_n - \mathbf{C}_*[k]\boldsymbol{\delta}_*{}^\top]$. Since all the nodes already know $\mathbf{C}_*[k]$ and $\boldsymbol{\delta}_*$, they only miss the global covariance matrix, which can be approximated through consensus $\mathbf{S}_* = \mathbf{Y} \bar{*} \mathbf{Y}$. Since both $\mathbf{S}_*$ and $\boldsymbol{\delta}_*$ are scaled equally, the factor T^{-1} can be moved out of the brackets resulting in $\mathbf{R}_*[k]$ that is arbitrarily close to $\mathbf{R}[k]$ and similar for every node

$$\begin{aligned}\mathbf{R}_*[k] &= \operatorname{diag}[\mathbf{S}_* - \mathbf{C}_*[k]\boldsymbol{\delta}_*^\top]\\ &= \operatorname{diag}\left[\frac{1}{T}\mathbf{Y}\mathbf{Y}^\top - \mathbf{C}[k]\left(\frac{1}{T}\mathbf{Y}\mathbf{X}^\top\right)^\top\right]\\ &= \mathbf{R}[k].\end{aligned} \tag{45}$$

Eqs. (44) and (45) are the baseline for the proposed **Consensus Based Distributed FA** (CB-DFA), shown as Algorithm 3.

Algorithm 3 Consensus Based Distributed FA (CB-DFA)

```
1: INPUT $t_n$, $\mathbf{Y}_n$
2: $t_* \leftarrow N^{-1}\sum_{n=1}^{N} t_n$                                  ⟸ consensus loop
3: $\mathbf{a}_n \leftarrow t_*^{-1}\sum_{j=1}^{t_n} \mathbf{y}_n^j$
4: $\boldsymbol{\mu}_* \leftarrow N^{-1}\sum_{n=1}^{N} \mathbf{a}_n$                                  ⟸ consensus loop
5: $\overline{\mathbf{Y}}_n \leftarrow \mathbf{Y}_n - \boldsymbol{\mu}_*$
6: $\mathbf{S}_* \leftarrow t_*^{-1}\overline{\mathbf{Y}} \bar{*} \overline{\mathbf{Y}}^\top$                                  ⟸ consensus loop
7: $\mathbf{C}_n[0] \leftarrow P$ random columns
8: $\mathbf{R}_n[0] \leftarrow$ random diagonal matrix
9: repeat
10:    $\boldsymbol{\beta}_* \leftarrow \mathbf{C}_*[k-1]^\top(\mathbf{C}_*[k-1]\mathbf{C}_*[k-1]^\top + \mathbf{R}_*[k-1])^{-1}$
11:    $\mathbf{X}_n[k] \leftarrow \boldsymbol{\beta}_*\overline{\mathbf{Y}}_n$
12:    $\mathbf{V}_*[k] \leftarrow \mathbf{I} - \boldsymbol{\beta}_*\mathbf{C}_*[k-1]$
13:    $\boldsymbol{\delta}_* \leftarrow \overline{\mathbf{Y}} \bar{*} \mathbf{X}[k]^\top$                                  ⟸ consensus loop
14:    $\boldsymbol{\eta}_* \leftarrow \mathbf{X}[k] \bar{*} \mathbf{X}[k]^\top$                                  ⟸ consensus loop
15:    $\boldsymbol{\gamma}_* \leftarrow \boldsymbol{\eta}[k]_* + \mathbf{V}_*[k]$
16:    $\mathbf{C}_*[k] \leftarrow \boldsymbol{\delta}_*(\boldsymbol{\gamma}_*)^{-1}$
17:    $\mathbf{R}_*[k] \leftarrow \operatorname{diag}(\mathbf{S}_* - \mathbf{C}_*[k]\boldsymbol{\delta}_*{}^\top)$
18: until the desired convergence criterion for $\mathbf{C}$ is met
19: OUTPUT $\mathbf{C}_n$, $\mathbf{R}_n$
```

5.4. Probabilistic PCA

Similar to FA, if we allow the covariance of the observed noise in (35) to be $\mathbf{R} \sim \mathcal{N}(0, \alpha\mathbf{I})$, then we get a fully probabilistic model for PCA, known as PPCA. Thus, an iterative EM algorithm to solve PPCA exists (Roweis, 1998; Tipping & Bishop, 1999), and we show here its distributed implementation. Indeed, the only difference to FA is how to update the observed noise

$$\alpha[k] = \frac{1}{DT}\operatorname{tr}(\mathbf{Y}\mathbf{Y}^\top(\mathbf{I} - \mathbf{C}[k]\boldsymbol{\beta})). \tag{46}$$

Hence, we follow the very same approach as CB-DFA to introduce a fully distributed EM algorithm to compute PPCA. Once every node has $\mathbf{S}_*$, $\mathbf{C}_*[k]$ and $\boldsymbol{\beta}_*$ available, it updates the observed noise

$$\alpha_* = \frac{1}{D}\operatorname{tr}(\mathbf{S}_*(\mathbf{I} - \mathbf{C}_*\boldsymbol{\beta}_*)) = \alpha. \tag{47}$$

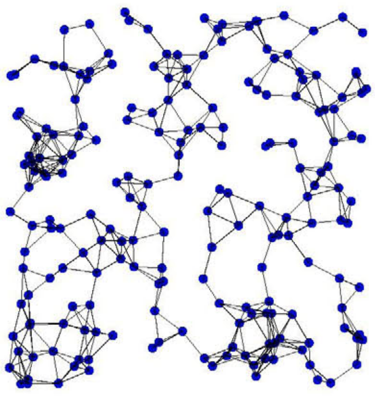
We denote our proposed algorithm for distributed computation of PPCA as **Consensus Based Distributed PPCA** (CB-DPPCA). It is identical to Algorithm 3 for CB-DFA except for line 17, which must be replaced by these two lines:

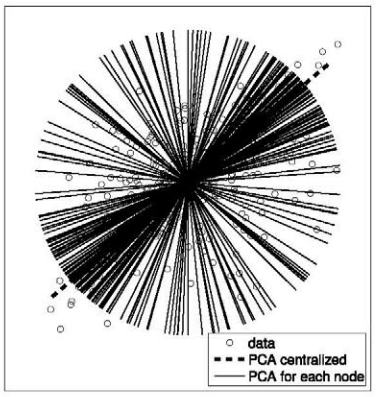
$$\alpha_*[k] \leftarrow \frac{1}{D}\operatorname{tr}(\mathbf{S}_*(\mathbf{I} - \mathbf{C}_*[k]\boldsymbol{\beta}_*))$$

$$\mathbf{R}_*[k] \leftarrow \alpha_*[k]\mathbf{I}.$$

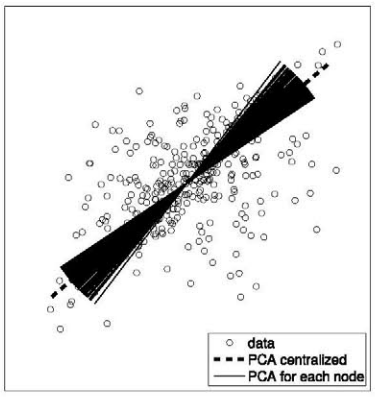
6. Experiments

In order to test the proposed distributed algorithms we present two simulated scenarios: a simple toy problem in which we examine the inherent properties of the proposed algorithms; and


(a) Network.



(b) Without cooperation.



(c) With 10 consensus rounds.

Fig. 3. Performance of PCA in isolation and with cooperation via consensus using CB-DPCA, showing the convergence of all nodes towards the centralized solution.

image compression, in order to demonstrate the applicability to traditional real-world applications.

In both experiments we use a relatively large and sparse randomly deployed Euclidean network of 200 nodes with average degree 5 (see Fig. 3(a)). The weight matrix is filled (in a fully distributed manner) using Metropolis–Hasting weights (Garin & Schenato, 2011). Although both examples are shown for the CB-DPCA algorithm, equivalent figures, behavior and conclusions are also valid for CB-EM-DPCA, CB-DPPCA and CB-DFA algorithms.

We focus on the asymptotic behavior of our algorithms. Hence, we do not examine here the influence of noisy links, switching topologies, and delay and quantization effects. They have been extensively analyzed in the literature (Kar & Moura, 2009; Olfati-Saber & Murray, 2004; Schizas, Giannakis, Roumeliotis, & Ribeiro, 2008; Schizas, Ribeiro, & Giannakis, 2008) in generic terms, and their influence on our algorithms is beyond the scope of this paper.

6.1. Toy problem

In this problem, the input data is embedded in a 2-dimensional space, i.e. $D = 2$, as e.g. when every node in a sensor network is equipped with two sensors. Meanwhile the subspace is 1-dimensional, $P = 1$. A distributed data set of 400 samples is drawn from a mixture of 4 Gaussians with random mean (over the range $[(0, 0), (3, 3)]$) and random covariance matrix (with variance over the range $[0, 9]$), are randomly allocated among the nodes. Every node has taken a different and random number of samples, between zero and four each, $t_n \in [0, 4]$. A small number of samples in each node allows us to highlight the performance of the algorithm even in such a low-dimensional problem.[1]

In Fig. 3, the performance of the CB-DPCA algorithm is shown with each node functioning in isolation (b); and with cooperation, after 10 consensus rounds (c). In both diagrams the subspace found by the traditional (centralized) PCA algorithm, which has access to the entire data set, is shown as a broken line. It can be seen clearly that the cooperation among the nodes which is built into the CB-DPCA algorithm leads not only to significant alignment among the nodes subspaces, but also to the centralized solution.

A natural performance/cost trade-off exists in every consensus based algorithm, due to its iterative nature. The effects of this trade-off are shown in Fig. 4. As expected, the communication cost grows linearly with the number of iterations of the consensus algorithm, simply because each iteration demands the exchange of a given volume of data among all the neighbors in the network.

[1] In general, when each node has very many samples, they are *independently* able to estimate the subspace well, making collaboration redundant. In our extremely simple toy example, we need ∼2 samples/node to keep in a range where cooperation is meaningful, for illustration purposes.

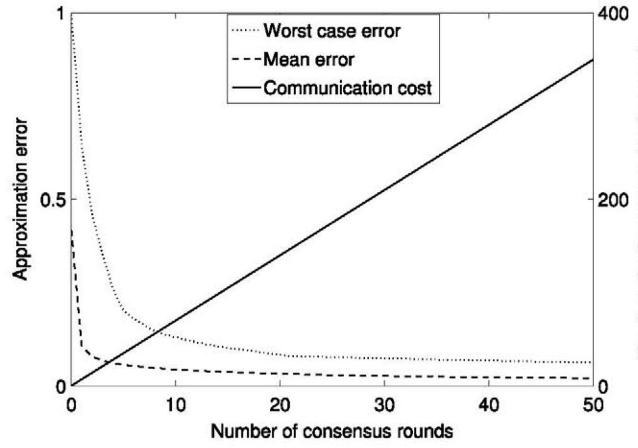


Fig. 4. Inherent cost-performance tradeoff for CB-DPCA, cost being the volume of information transmitted, and performance represented by the reduction of the angle between the locally-estimated and the central (optimal) subspace.

On the other hand, the angle between the resulting subspaces and the centralized solution drops monotonically (in the mean) with each iteration. In Fig. 4 we show both the mean and maximum angle for the whole network. The inverse of this angle can be seen as a metric of performance of the CB-DPCA algorithm. In other words, the subspaces computed in all the nodes in the network approximate the centralized solution more closely with each additional consensus iteration.

6.2. Image compression experiments

We use the Lena image at 512×512 resolution and 8 bit grayscale. The image has been divided into 441 blocks of 24×24 pixels. Those blocks have been vectorized (aggregating columns) as points lying in a 576 dimensional subspace. Each node randomly draws an arbitrary number of samples over the range $[1, 6]$. As explained in Section 5.1, every node first computes its own sample covariance matrix that is then shared iteratively with its neighbors a given number of consensus rounds. After the iterative collaboration, each node will compute, again in isolation, the 20 first principal components based on its resulting estimate of the global sample covariance matrix.

Fig. 5(a) displays the original Lena image, while Fig. 5(b) shows the compression that a fusion center would achieve in a centralized manner. Fig. 5(c) and (d) show the compression that two random nodes could achieve in isolation, i.e. with no cooperation with their neighbors. Then in Fig. 5(e) and (f) we show the noticeable improvement achieved when nodes share their estimates even for

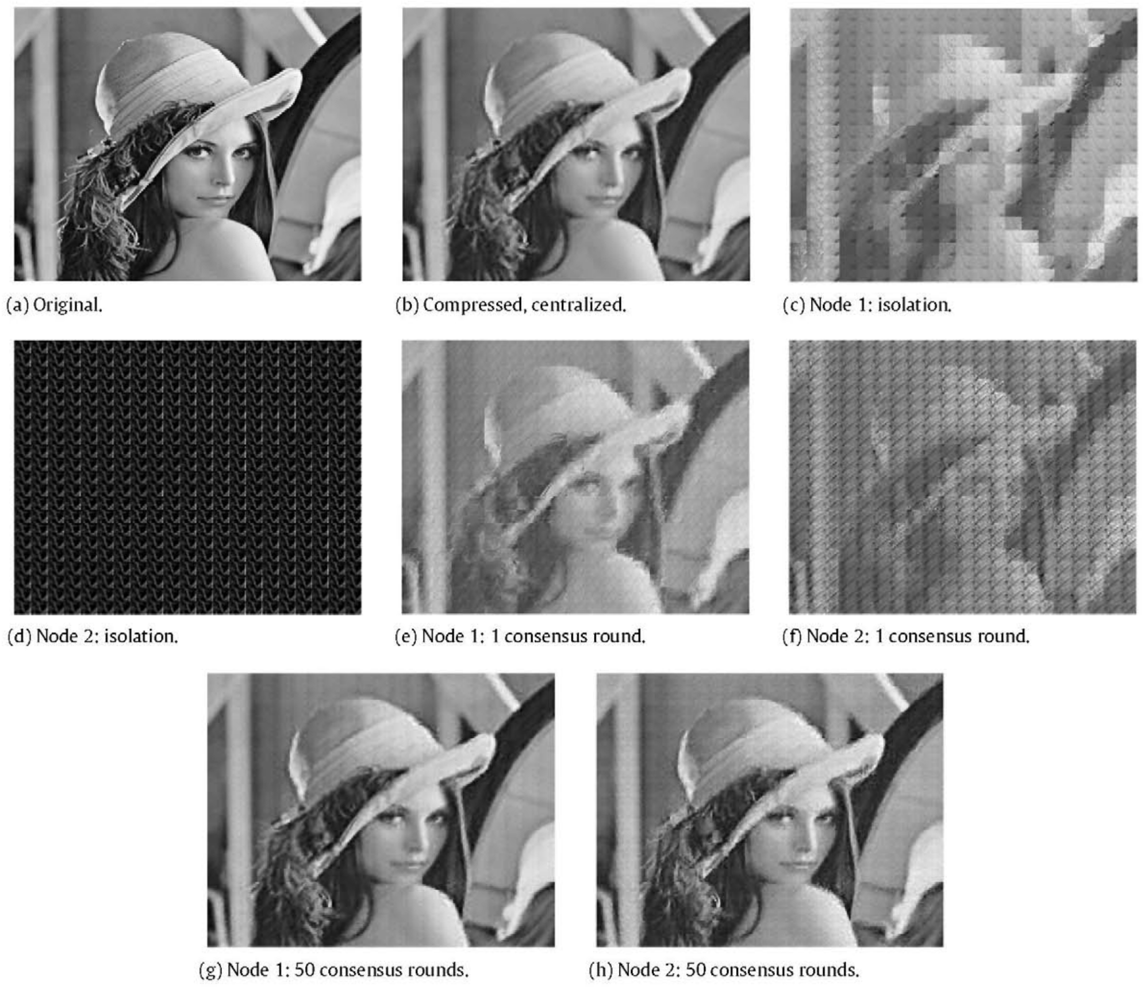

(a) Original.
(b) Compressed, centralized.
(c) Node 1: isolation.
(d) Node 2: isolation.
(e) Node 1: 1 consensus round.
(f) Node 2: 1 consensus round.
(g) Node 1: 50 consensus rounds.
(h) Node 2: 50 consensus rounds.

Fig. 5. Performance of PCA in isolation and with cooperation via consensus using CB-DPCA. Images (c), (e), and (g) illustrate how a node asymptotically converges to the centralized solution given in image (b). Images (d), (f), and (h) show the same for another node in the network.

just 1 consensus round. Finally, Fig. 5(g) and (h) show how both nodes asymptotically converge towards the same global solution after 50 consensus rounds.

7. Conclusions

In this article we presented a toolkit for creating distributed versions of centralized algorithms using average consensus. We also showed how this toolkit can be used to build two novel, fully distributed algorithms to perform PCA. Both algorithms require no fusion center and rely only on low-volume local (1-hop) communications. This in turn makes them very efficient, scalable, and robust. Both also offer guaranteed convergence to the centralized solution. Furthermore, we gave two extensions of the presented algorithms to distribute two closely related techniques: FA and PPCA. We also illustrated the performance of our algorithms on two simulated examples, with a discussion on the convergence and the inherent cost-performance trade-off.

Acknowledgments

The authors would like to thank Silvana Silva Pereira and Adriano Pastore for their collaboration in showing the positive semi-definite property of the local intermediate estimates of the global covariance matrix in Section 4.3.

References

Ahn, J.-H., Oh, J.-H., & Choi, S. (2007). Learning principal directions: integrated-squared-error minimization. In *Neurocomputing, 14th European symposium on artificial neural networks* (pp. 1372–1381) *Vol. 70* (7–9).

Bai, Z., Chan, R.H., & Luk, F.T. (2005). Principal component analysis for distributed data sets with updating. In *Proceedings of international workshop on advanced parallel processing technologies*, APPT (pp. 471–483).

Bajovic, D., Jakovetic, D., Xavier, J., Sinopoli, B., & Moura, J. M. F. (2011). Distributed detection via Gaussian running consensus: large deviations asymptotic analysis. *IEEE Transactions on Signal Processing*, *59*(9), 4381–4396.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, *2*(1), 53–58.

Bertsekas, D. P., & Tsitsiklis, J. N. (1997). *Parallel and distributed computation: numerical methods*. Athena Scientific.

De la Torre, F. (2009). A unification of component analysis methods. In *Handbook of pattern recognition and computer vision* (4th ed.) (pp. 3–22). World Scientific Publishing.

Diamantaras, K. I., & Kung, S. Y. (1996). *Principal component neural networks: theory and applications*. John Wiley & Sons.

Forero, P., Cano, A., & Giannakis, G. (2011). Distributed clustering using wireless sensor networks. *IEEE Journal of Selected Topics in Signal Processing*, *5*(4), 707–724.

Forero, P., Cano, A., & Giannakis, G. (2010). Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, *11*, 1663–1707.

Garin, F., & Schenato, L. (2011). A survey on distributed estimation and control applications using linear consensus algorithms. In A. Bemporad, M. Heemels, & M. Johansson (Eds.), *Lecture notes in control and information sciences*: *Vol. 406*. *Networked control systems* (pp. 75–107). Berlin, Heidelberg: Springer.

Gastpar, M., Dragotti, P., & Vetterli, M. (2006). The distributed Karhunen–Loève transform. *IEEE Transactions on Information Theory*, *52*(12), 5177–5196.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417–441.

Jolliffe, I. T. (2002). *Springer series in statistics, Principal component analysis* (2nd ed.). Springer.

Kargupta, H., Huang, W., Sivakumar, K., & Johnson, E. (2001). Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, *3*, 422–448.

Kar, S., & Moura, J. (2009). Distributed consensus algorithms in sensor networks with imperfect communication: link failures and channel noise. *IEEE Transactions on Signal Processing*, *57*, 355–369.

Le Borgne, Y.-A., Raybaud, S., & Bontempi, G. (2008). Distributed principal component analysis for wireless sensor networks. *Sensors*, *8*(8), 4821–4850.

Olfati-Saber, R. (2005). Distributed Kalman filter with embedded consensus filters. In *44th IEEE conference on decision and control* (pp. 8179–8184).

Olfati-Saber, R., & Murray, R. (2004). Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, *49*, 1520–1533.

Olshevsky, A., & Tsitsiklis, J. N. (2009). Convergence speed in distributed consensus and averaging. *SIAM Journal on Control and Optimization*, *48*, 33–55.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, *2*(6), 559–572.

Qi, H., & Wang, T. (2004). Global principal component analysis for dimensionality reduction in distributed data mining. In *Statistical data mining and knowledge discovery* (pp. 327–342). Chapman & Hall, CRC Press.

Reyment, R. A., & Jöreskog, K. G. (1996). *Applied factor analysis in the natural sciences* (2nd ed.). Cambridge University Press.

Roweis, S. (1998). EM algorithms for PCA and SPCA. *Advances in Neural Information Processing Systems*, *10*, 626–632.

Roweis, S., & Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural Computation*, *11*, 305–345.

Schizas, I., Giannakis, G., Roumeliotis, S., & Ribeiro, A. (2008). Consensus in ad hoc WSNs with noisy linkspart II: distributed estimation and smoothing of random signals. *IEEE Transactions on Signal Processing*, *56*, 1650–1666.

Schizas, I., Ribeiro, A., & Giannakis, G. (2008). Consensus in ad hoc WSNs with noisy linkspart I: distributed estimation of deterministic signals. *IEEE Transactions on Signal Processing*, *56*, 350–364.

Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, *61*, 611–622.

Tsitsiklis, J. N. (1993). Decentralized detection. In H. V. Poor, & J. B. Thomas (Eds.), *Advances in signal processing*: *Vol. 2* (pp. 297–344). JAI Press.

Valcarcel Macua, S., Belanovic, P., & Zazo, S. (2010). Consensus-based distributed principal component analysis in wireless sensor networks. In *Proceedings of the IEEE international workshop on signal processing advances for wireless communications*, SPAWC.

Valcarcel Macua, S., Belanovic, P., & Zazo, S. (2011). Distributed linear discriminant analysis. In *International conference on acoustics, speech and signal processing*, ICASSP (pp. 3288–3291).

van de Geijn, R. A. (1997). *Using PLAPACK*. MIT Press.

Xiao, L., & Boyd, S. (2004). Fast linear iterations for distributed averaging. *Systems and Control Letters*, *53*, 65–78.