Tighter Low-rank Approximation via Sampling the Leveraged Element[*]

Srinadh Bhojanapalli
The University of Texas at Austin
bsrinadh@utexas.edu

Prateek Jain
Microsoft Research, India
prajain@microsoft.com

Sujay Sanghavi
The University of Texas at Austin
sanghavi@mail.utexas.edu

Abstract

In this work, we propose a new randomized algorithm for computing a low-rank approximation to a given matrix. Taking an approach different from existing literature, our method first involves a specific biased sampling, with an element being chosen based on the leverage scores of its row and column, and then involves weighted alternating minimization over the factored form of the intended low-rank matrix, to minimize error only on these samples. Our method can leverage input sparsity, yet produce approximations in *spectral* (as opposed to the weaker Frobenius) norm; this combines the best aspects of otherwise disparate current results, but with a dependence on the condition number $\kappa = \sigma_1/\sigma_r$. In particular we require $O(nnz(M) + \frac{n\kappa^2 r^5}{\epsilon^2})$ computations to generate a rank-r approximation to M in spectral norm. In contrast, the best existing method requires $O(nnz(M) + \frac{nr^2}{\epsilon^4})$ time to compute an approximation in Frobenius norm. Besides the tightness in spectral norm, we have a better dependence on the error ϵ. Our method is naturally and highly parallelizable.

Our new approach enables two extensions that are interesting on their own. The first is a new method to directly compute a low-rank approximation (in efficient factored form) to the product of two given matrices; it computes a small random set of entries of the product, and then executes weighted alternating minimization (as before) on these. The sampling strategy is different because now we cannot access leverage scores of the product matrix (but instead have to work with input matrices). The second extension is an improved algorithm with smaller communication complexity for the distributed PCA setting (where each server has small set of rows of the matrix, and want to compute low rank approximation with small amount of communication with other servers).

1 Introduction

Finding a low-rank approximation to a matrix is fundamental to a wide array of machine learning techniques. The large sizes of modern data matrices has driven much recent work into efficient (typically randomized) methods to find low-rank approximations that do not exactly minimize the residual, but run much faster / parallel, with fewer passes over the data. Existing approaches involve either intelligent sampling of a few rows / columns of the matrix, projections onto lower-dimensional spaces, or sampling of entries followed by a top-r SVD of the resulting matrix (with unsampled entries set to 0).

[*]Extended abstract to appear in the proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA15).

We pursue a different approach: we first sample entries in a specific biased random way, and then minimize the error on these samples over a search space that is the factored form of the low-rank matrix we are trying to find. We note that this is different from approximating a 0-filled matrix; it is instead reminiscent of matrix completion in the sense that it only looks at errors on the sampled entries. Another crucial ingredient is how the sampling is done: we use a combination of ℓ_1 sampling, and of a distribution where the probability of an element is proportional to the sum of the leverage scores of its row and its column.

Both the sampling and the subsequent alternating minimization are naturally fast, parallelizable, and able to utilize sparsity in the input matrix. Existing literature has either focused on running in input sparsity time but approximation in (the weaker) Frobenius norm, or running in $O(n^2)$ time with approximation in spectral norm. Our method provides the best of both worlds: it runs in input sparsity time, with just two passes over the data matrix, and yields an approximation in spectral norm. It does however have a dependence on the ratio of the first to the r^{th} singular value of the matrix.

Our alternative approach also yields new methods for two related problems: directly finding the low-rank approximation of the product of two given matrices, and distributed PCA.

Our contributions are thus three new methods in this space:

- **Low-rank approximation of a general matrix:** Our first (and main) contribution is a new method (LELA, Algorithm 1) for low-rank approximation of any given matrix: first draw a random subset of entries in a specific biased way, and then execute a weighted alternating minimization algorithm that minimizes the error on these samples over a factored form of the intended low-rank matrix. The sampling is done with only two passes over the matrix (each in input sparsity time), and both the sampling and the alternating minimization steps are highly parallelizable and compactly stored/manipulated.

  For a matrix M, let M_r be the best rank-r approximation (i.e. the matrix corresponding to top r components of SVD). Our algorithm finds a rank-r matrix $\widehat{M_r}$ in time $O(nnz(M) + \frac{n\kappa^2 r^5}{\epsilon^2})$, while providing approximation in spectral norm: $\|M - \widehat{M_r}\| \leq \|M - M_r\| + \epsilon\|M - M_r\|_F$, where $\kappa = \sigma_1(M)/\sigma_r(M)$ is the condition number of M_r. Existing methods either can run in input sparsity time, but provide approximations in (the weaker) Frobenius norm (i.e. with $||\cdot||$ replaced by $||\cdot||_F$ in the above expression), or run in $O(n^2)$ time to approximate in spectral norm, but even then with leading constants larger than 1. See Table 1 for a detailed comparison to existing results for low-rank approximation.
- **Direct approximation of a matrix product:** We provide a new method to directly find a low-rank approximation to the product of two matrices, without having to first compute the product itself. To do so, we first choose a small set of entries (in a biased random way) of the product that we will compute, and then again run weighted alternating minimization on these samples. The choice of the biased random distribution is now different from above, as the sampling step does not have access to the product matrix. However, again both the sampling and alternating minimization are highly parallelizable.
- **Distributed PCA:** Motivated by applications with really large matrices, recent work has looked at low-rank approximation in a distributed setting where there are s servers – each have small set of rows of the matrix – each of which can communicate with a central processor charged with coordinating the algorithm. In this model, one is interested in find good approximations while minimizing both computations and the communication burden on the center.

  We show that our LELA algorithm can be extended to the distributed setting while guaranteeing small communication complexity. In particular, our algorithm guarantees the same error bounds as that of our non-distributed version but guarantees communication complexity of $O(ds + \frac{nr^5\kappa^2}{\epsilon^2}\log n)$ real numbers for computing rank-r approximation to $M \in \mathbb{R}^{n\times d}$. For $n \approx d$ and large s, our analysis guarantees significantly lesser communication complexity than the state-of-the-art method [15], while providing tighter spectral norm bounds.

Notation: Capital letter M typically denotes a matrix. M^i denotes the i-th row of M, M_j denotes the j-th column of M, and M_{ij} denotes the (i,j)-th element of M. Unless specified otherwise, $M \in \mathbb{R}^{n\times d}$ and M_r is the best rank-r approximation of M. Also, $M_r = U^*\Sigma^*(V^*)^T$ denotes the

Reference	Frobenius norm	Spectral norm	Computation time
BJS14 (Our Algorithm)	$(1+\epsilon)\|\Delta\|_F$	$\|\Delta\| + \epsilon\|\Delta\|_F$	$O(nnz(M) + \frac{nr^5\kappa^2\log(n)}{\epsilon^2})$
CW13[5]	$(1+\epsilon)\|\Delta\|_F$	$(1+\epsilon)\|\Delta\|_F$	$O(nnz(M) + \frac{nr^2}{\epsilon^4} + \frac{r^3}{\epsilon^5})$
BG13 [3]	$(1+\epsilon)\|\Delta\|_F$	$c\|\Delta\| + \epsilon\|\Delta\|_F$	$O(n^2(\frac{r+\log(n)}{\epsilon^2}) + n\frac{r^2\log(n)^2}{\epsilon^4})$
NDT09[19]	$(1+\epsilon)\|\Delta\|_F$	$c\|\Delta\| + \epsilon\sqrt{n}\|\Delta\|$	$O(n^2\log(\frac{r\log(n)}{\epsilon}) + \frac{nr^2\log(n)^2}{\epsilon^4})$
WLRT08[21]	$(1+\epsilon)\|\Delta\|_F$	$\|\Delta\| + \epsilon\sqrt{n}\|\Delta\|$	$O(n^2\log(\frac{r}{\epsilon}) + \frac{nr^4}{\epsilon^4})$
Sar06[20]	$(1+\epsilon)\|\Delta\|_F$	$(1+\epsilon)\|\Delta\|_F$	$O(nnz(M)\frac{r}{\epsilon} + n\frac{r^2}{\epsilon^2})$

Table 1: Comparison of error rates and computation time of some low rank approximation algorithms. $\Delta = M - M_r$.

SVD of M_r. $\kappa = \sigma_1^*/\sigma_r^*$ denotes the condition number of M_r, where σ_i^* is the i-th singular value of M. $\|u\|$ denotes the L_2 norm of vector u. $\|M\| = \max_{\|x\|=1}\|Mx\|$ denotes the spectral or operator norm of M. $\|M\|_F = \sqrt{\sum_{ij} M_{ij}^2}$ denotes the Frobenius norm of M. Also, $\|M\|_{1,1} = \sum_{ij}|M_{ij}|$. $dist(X,Y) = \|X_\perp^T Y\|$ denotes the principal angle based distance between subspaces spanned by X and Y orthonormal matrices. Typically, C denotes a global constant independent of problem parameters and can change from step to step.

Given a set $\Omega \subseteq [n] \times [d]$, $P_\Omega(M)$ is given by: $P_\Omega(M)(i,j) = M_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise. $R_\Omega(M) = w. * P_\Omega(M)$ denotes the Hadamard product of w and $P_\Omega(M)$. That is, $R_\Omega(M)(i,j) = w_{ij}M_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise. Similarly let $R_\Omega^{1/2}(M)(i,j) = \sqrt{w_{ij}}M_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise.

2 Related results

Low rank approximation: Now we will briefly review some of the existing work on algorithms for low rank approximation. [11] introduced the problem of computing low rank approximation of a matrix M with few passes over M. They presented an algorithm that samples few rows and columns and does SVD to compute low rank approximation, and gave additive error guarantees. [7, 8] have extended these results. [1] considered a different approach based on entrywise sampling and quantization for low rank approximation and has given additive error bounds.

[14, 20, 9, 6] have given low rank approximation algorithms with relative error guarantees in Frobenius norm. [21, 19] have provided guarantees on error in spectral norm which are later improved in [13, 3]. The main techniques of these algorithms is to use a random Gaussian or Hadamard transform matrix for projecting the matrix onto a low dimensional subspace and compute the rank-r subspace. [3] have given an algorithm based on random Hadamard transform that computes rank-r approximation in time $O(\frac{n^2 r}{\epsilon^2})$ and gives spectral norm bound of $\|M - \widehat{M_r}\| \le c\|M - M_r\| + \epsilon\|M - M_r\|_F$. Recently [5] gave an algorithm using sparse subspace embedding that runs in input sparsity time with relative Frobenius norm error guarantees.

We presented some results in this area as a comparison with our results in table 1. This is a heavily subsampled set of existing results on low rank approximations. There is a lot of interesting work on very related problems of computing column/row based(CUR) decompositions, matrix sketching, low rank approximation with streaming data. Look at [18, 13] for more detailed discussion and comparison.

Distributed PCA: In distributed PCA, one wants to compute rank-r approximation of a $n \times d$ matrix that is stored across s servers with small communication between servers. One popular model is row partition model where subset of rows are stored at each server. Algorithms in [10, 17, 12, 16] achieve $O(\frac{dsr}{\epsilon})$ communication complexity with relative error guarantees in Frobenius norm, under this model.

Recently [15] have considered the scenario of arbitrary splitting of a $n \times d$ matrix and given an algorithm that has $O(\frac{dsr}{\epsilon})$ communication complexity with relative error guarantees in Frobenius norm.

3 Low-rank Approximation of Matrices

In this section we will present our main contribution: a new randomized algorithm for computing low-rank approximation of any given matrix. Our algorithm first samples a few elements from the given matrix $M \in \mathbb{R}^{n\times d}$, and then rank-$r$ approximation is computed using only those samples. Algorithm 1 provides a detailed pseudo-code of our algorithm; we now comment on each of the two stages:

Sampling: A crucial ingredient of our approach is using the correct sampling distribution. Recent results in matrix completion [4] indicate that a small number ($O(nr\log^2(n))$) of samples drawn in a way biased by leverage scores[1] can capture *all* the information in any exactly low-rank matrix. While this is indicative, here we have neither access to the leverage scores, nor is our matrix exactly low-rank. We approximate the leverage scores with the row and column norms ($||M^i||^2$ and $||M_j||^2$), and account for the arbitrary high-rank nature of input by including an L_1 term in the sampling; the distribution is given in eq. (2). Computationally, our sampling procedure can be done in two passes and $O(m\log n + nnz(M))$ time.

Weighted alternating minimization: In our second step, we express the low-rank approximation $\widehat{M_r}$ as UV^T and then iterate over U and V alternatingly to minimize the weighted L_2 error over the *sampled* entries (see Sub-routine 2). Note that this is different from taking principal components of a 0-filled version of the sampled matrix. The weights give higher emphasis to elements with smaller sampling probabilities. In particular, the goal is to minimize the following objective function:

$$Err(\widehat{M_r}) = \sum_{(i,j)\in\Omega} w_{ij}\left(M_{ij} - (\widehat{M_r})_{ij}\right)^2, \tag{1}$$

where $w_{ij} = 1/\hat{q}_{ij}$ when $\hat{q}_{ij} > 0$, 0 else. For initialization of the WAltMin procedure, we compute SVD of $R_\Omega(M)$ (reweighed sampled matrix) followed by a trimming step (see Step 4, 5 of Sub-routine 2). Trimming step sets $(\widehat{U}^0)^i = 0$ if $\|(\widehat{U}^0)^i\| \geq 4\|M^i\|/\|M\|_F$ and $(\widehat{U}^0)^i = (U^0)^i$ otherwise; and prevents heavy rows/columns from having undue influence.

We now provide our main result for low-rank approximation and show that Algorithm 1 can provide a tight approximation to M_r while using a small number of samples $m = \mathbb{E}[|\Omega|]$.

Theorem 3.1. *Let $M \in \mathbb{R}^{n\times d}$ be any given matrix ($n \geq d$) and let M_r be the best rank-r approximation to M. Set the number of samples $m = \frac{C}{\gamma}\frac{nr^3}{\epsilon^2}\kappa^2\log(n)\log^2(\frac{\|M\|}{\zeta})$, where $C > 0$ is any global constant, $\kappa = \sigma_1/\sigma_r$ where σ_i is the i-th singular value of M. Also, set the number of iterations of WAltMin procedure to be $T = \log(\frac{\|M\|}{\zeta})$. Then, with probability greater than $1-\gamma$ for any constant $\gamma > 0$, the output $\widehat{M_r}$ of Algorithm 1 with the above specified parameters m, T, satisfies:*

$$\|M - \widehat{M_r}\| \leq \|M - M_r\| + \epsilon\,\|M - M_r\|_F + \zeta.$$

That is, if $T = \log(\frac{\|M\|}{\epsilon\|M-M_r\|_F})$, we have:

$$\|M - \widehat{M_r}\| \leq \|M - M_r\| + 2\epsilon\,\|M - M_r\|_F.$$

3.1 Computation complexity:

In the first step we take 2 passes over the matrix to compute the sampling distribution (2) and sampling the entries based on this distribution. It is easy to show that this step would require $O(nnz(M) + m\log(n))$ time. Next, the initialization step of WAltMin procedure requires computing rank-r SVD of $R_{\Omega_0}(M)$ which has at most m non-zero entries. Hence, the procedure can be completed in $O(mr)$ time using standard techniques like power method. Note that we need top-r singular vectors of $R_{\Omega_0}(M)$ only upto constant approximation. Further t-th iteration of alternating minimization takes $O(2|\Omega_{2t+1}|r^2)$ time. So, the total time complexity of our method is $O(nnz(M) + mr^2)$. As shown in Theorem 3.1, our method requires

[1]If SVD of $M_r = U^*\Sigma^*(V^*)^T$ then leverage scores of M_r are $||(U^*)^i||^2$ and $||(V^*)^j||^2$ for all i, j.

Algorithm 1 LELA: Leveraged Element Low-rank Approximation

```
input  matrix: $M \in \mathbb{R}^{n\times d}$, rank: $r$, number of samples: $m$, number of iterations: $T$
1: Sample $\Omega \subseteq [n] \times [d]$ where each element is sampled independently with probability: $\hat{q}_{ij} = \min\{q_{ij}, 1\}$
       $$q_{ij} = m \cdot \left( \frac{\|M^i\|^2 + \|M_j\|^2}{2(n+d)\|M\|_F^2} + \frac{|M_{ij}|}{2\|M\|_{1,1}} \right). \quad (2)$$
   /*See Section 3.1 for details about efficient implementation of this step*/
2: Obtain $P_\Omega(M)$ using one pass over $M$
3: $\widehat{M}_r = \mathsf{WAltMin}(P_\Omega(M), \Omega, r, \hat{q}, T)$
output  $\widehat{M}_r$
```

Sub-routine 2 WAltMin: Weighted Alternating Minimization

```
input  $P_\Omega(M),\ \Omega,\ r,\ \hat{q},\ T$
1: $w_{ij} = 1/\hat{q}_{ij}$ when $\hat{q}_{ij} > 0$, 0 else, $\forall i, j$
2: Divide $\Omega$ in $2T+1$ equal uniformly random subsets, i.e., $\Omega = \{\Omega_0, \ldots, \Omega_{2T}\}$
3: $R_{\Omega_0}(M) \leftarrow w. * P_{\Omega_0}(M)$
4: $U^{(0)}\Sigma^{(0)}(V^{(0)})^T = SVD(R_{\Omega_0}(M), r)$ //Best rank-$r$ approximation of $R_{\Omega_0}(M)$
5: Trim $U^{(0)}$ and let $\widehat{U}^{(0)}$ be the output (see Section 3)
6: for $t = 0$ to $T-1$ do
7:     $\widehat{V}^{(t+1)} = \text{argmin}_V \|R^{1/2}_{\Omega_{2t+1}}(M - \widehat{U}^{(t)}V^T)\|_F^2$, for $V \in \mathbb{R}^{d\times r}$.
8:     $\widehat{U}^{(t+1)} = \text{argmin}_U \|R^{1/2}_{\Omega_{2t+2}}(M - U(\widehat{V}^{(t+1)})^T)\|_F^2$ , for $U \in \mathbb{R}^{n\times r}$.
9: end for
output  Completed matrix $\widehat{M}_r = \widehat{U}^{(T)}(\widehat{V}^{(T)})^T$.
```

$m = O(\frac{nr^3}{\epsilon^2}\kappa^2 \log(n) \log^2(\frac{\|M\|}{\epsilon\|M-M_r\|_F}))$ samples. Hence, the total run-time of our algorithm is: $O(nnz(M) + \frac{nr^5}{\epsilon^2}\kappa^2 \log(n) \log^2(\frac{\|M\|}{\epsilon\|M-M_r\|_F}))$.

3.2 Proof Overview:

The key steps in our proof of Theorem 3.1 are to show that 1)the initialization procedure (step 4 of Sub-routine 2) provides an accurate enough estimate of M_r and then 2)at each step, we show a geometric decrease in distance to M_r. Our proof differs from the previous works on alternating minimization in two key aspects: a) existing proof techniques of alternating minimization assume that each element is sampled uniformly at random, while we can allow biased and approximate sampling, b) existing techniques crucially use the assumption that M_r is incoherent, while our proof avoids this assumption using the weighted version of AltMin. Complete proof is given in the full version [2].

3.3 Direct Low-rank Approximation of Matrix Product

In this section we present a new algorithm for the following problem: suppose we are given two matrices, and desire a low-rank approximation of their product AB; in particular, we are *not* interested in the actual full matrix product itself (as this may be unwieldy to store and use, and thus wasteful to produce in its entirety). One example setting where this arises is when one wants to calculate the joint counts between two very large sets of entities; for example, web companies routinely come across settings where they need to understand (for example) how many users both searched for a particular query and clicked on a particular advertisement. The number of possible queries and ads is huge, and finding this co-occurrence matrix from user logs involves multiplying two matrices – query-by-user and user-by-ad respectively – each of which is itself large.

We give a method that directly produces a low-rank approximation of the final product, and involves storage and manipulation of only the efficient factored form (i.e. one tall and one fat matrix) of the final intended low-rank matrix. Note that as opposed to the previous section, the matrix does not already exist and hence we do not have access to its row and column norms; so we need a new sampling scheme (and a different proof of correctness).

Algorithm: Suppose we are given an $n_1 \times d$ matrix A and another $d \times n_2$ matrix B, and we wish to calculate a rank-r approximation of the product $A \cdot B$. Our algorithm proceeds in two stages:

1. Choose a biased random set $\Omega \subset [n_1] \times [n_2]$ of elements as follows: choose an intended number m (according to Theorem 3.2 below) of sampled elements, and then independently include each $(i, j) \in [n_1] \times [n_2]$ in Ω with probability given by $\hat{q}_{ij} = \min\{1, q_{ij}\}$ where
$$q_{ij} \;:=\; m \cdot \left(\frac{\|A^i\|^2}{n_2\|A\|_F^2} + \frac{\|B_j\|^2}{n_1\|B\|_F^2} \right), \tag{3}$$
Then, find $P_\Omega(A \cdot B)$, i.e. only the elements of the product AB that are in this set Ω.
2. Run the alternating minimization procedure WAltMin$(P_\Omega(A \cdot B), \Omega, r, \hat{q}, T)$, where T is the number of iterations (again chosen according to Theorem 3.2 below). This produces the low-rank approximation in factored form.

The computation complexity of the algorithm is $O(|\Omega| \cdot (d + r^2)) = O(m(d + r^2)) = O(\frac{nr^3\kappa^2}{\epsilon^2} \cdot (d + r^2))$ (suppressing terms dependent on norms of A and B), where $n = \max\{n_1, n_2\}$. We now present our theorem on the number of samples and iterations needed to make this procedure work with at least a constant probability.

Theorem 3.2. *Consider matrices $A \in \mathbb{R}^{n_1 \times d}$ and $B \in \mathbb{R}^{d \times n_2}$ and let $m = \frac{C}{\gamma} \cdot \frac{(\|A\|_F^2 + \|B\|_F^2)^2}{\|AB\|_F^2} \cdot \frac{nr^3}{(\epsilon)^2}\kappa^2 \log(n) \log^2(\frac{\|A\|_F + \|B\|_F}{\zeta})$, where $\kappa = \sigma_1^*/\sigma_r^*$, σ_i^* is the i-th singular value of $A \cdot B$ and $T = \log(\frac{\|A\|_F + \|B\|_F}{\zeta})$. Let Ω be sampled using probability distribution* (3). *Then, the output $\widehat{AB}_r = WAltMin(P_\Omega(A \cdot B), \Omega, r, \hat{q}, T)$ of Sub-routine 2 satisfies (w.p. $\geq 1 - \gamma$): $\|A \cdot B - \widehat{AB}_r\| \leq \|A \cdot B - (A \cdot B)_r\| + \epsilon\|A \cdot B - (A \cdot B)_r\|_F + \zeta$.*

4 Distributed Principal Component Analysis

Modern large-scale systems have to routinely compute PCA of data matrices with millions of data points embedded in similarly large number of dimensions. Now, even storing such matrices on a single machine is not possible and hence most industrial scale systems use distributed computing environment to handle such problems. However, performance of such systems depend not only on computation and storage complexity, but also on the required amount of communication between different servers.

In particular, we consider the following distributed PCA setting: Let $M \in \mathbb{R}^{n \times d}$ be a given matrix (assume $n \geq d$ but $n \approx d$). Also, let M be row partitioned among s servers and let $M_{\mathbf{r_k}} \in \mathbb{R}^{n \times d}$ be the matrix with rows $\{r_k\} \subseteq [n]$ of M, stored on k-th server. Moreover, we assume that one of the servers act as Central Processor(CP) and in each round all servers communicate with the CP and the CP communicates back with all the servers. Now, the goal is to compute $\widehat{M}_r$, an estimate of M_r, such that the total communication (i.e. number of bits transferred) between CP and other servers is minimized. Note that, such a model is now standard for this problem and was most recently studied by [15].

Recently several interesting results [10, 12, 16, 15] have given algorithms to compute rank-r approximation of M, $\widetilde{M}_r$ in the above mentioned distributed setting. In particular, [15] proposed a method that for row-partitioned model requires $O(\frac{dsr}{\epsilon} + \frac{sr^2}{\epsilon^4})$ communication to obtain a relative Frobenius norm guarantee.

In contrast, a distributed setting extension of our LELA algorithm 1 has linear communication complexity $O(ds + \frac{nr^5}{\epsilon^2})$ and computes rank-r approximation $\widehat{M}_r$, with $||M - \widehat{M}_r|| \leq ||M - M_r|| + \epsilon||M - M_r||_F$. Now note that if $n \approx d$ and if s scales with n (which is a typical requirement), then our communication complexity can be significantly better than that of [15]. Moreover, our method provides spectral norm bounds as compared to relatively weak Frobenius bounds mentioned above.

Algorithm: The distributed version of our LELA algorithm depends crucially on the following observation: given V, each row of U can be updated independently. Hence, servers need to communicate rows of V only. There also, we can use the fact that each server requires only $O(nr/s \log n)$ rows of V to update their corresponding $U_{\mathbf{r_k}}$. $U_{\mathbf{r_k}}$ denote restriction of U to rows in set $\{r_k\}$ and 0 outside and similarly $\widehat{V}_{\mathbf{c_k}}, \widehat{Y}_{\mathbf{c_k}}$ denote restriction of $V, \widehat{Y}$ to rows $\{c_k\}$.

We now describe the distributed version of each of the critical step of LELA algorithm. See [2] for a detailed pseudo-code.

Sampling: For sampling, we first compute column norms $\|M_j\|, \forall 1 \le j \le d$ and communicate to each server. This operation would require $O(ds)$ communication. Next, each server (server k) samples elements from its rows $\{\mathbf{r_k}\}$ and stores $R_{\Omega^k}(M_{\mathbf{r_k}})$ locally. Note that because of *independence over rows*, the servers don't need to transmit their samples to other servers.

Initialization: In the initialization step, our algorithm computes top r right singular vector of $R_\Omega(M)$ by iterations $\widehat{Y}^{(t+1)} = R_\Omega(M)^T R_\Omega(M) Y^t = \sum_k R_{\Omega^k}(M)^T R_{\Omega^k}(M) Y^t$. Now, note that computing $R_{\Omega^k}(M)Y^t$ requires server k to access atmost $|\Omega^k|$ columns of Y^t. Hence, the total communication from the CP to all the servers in this round is $O(|\Omega|r)$. Similarly, each column of $R_{\Omega^k}(M)^T R_{\Omega^k}(M)Y^t$ is only $|\Omega^k|$ sparse. Hence, total communication from all the servers to CP in this round is $O(|\Omega|r)$. Now, we need constant many rounds to get a constant factor approximation to SVD of $R_\Omega(M)$, which is enough for good initialization in WAltMin procedure. Hence, total communication complexity of the initialization step would be $O(|\Omega|r)$.

Alternating Minimization Step: For alternating minimization, update to rows of U is computed at the corresponding servers and the update to V is computed at the CP. For updating $\widehat{U}^{(t+1)}_{\mathbf{r_k}}$ at server k, we use the following observation: updating $\widehat{U}^{(t+1)}_{\mathbf{r_k}}$ requires atmost $|\Omega^k|$ rows of $\widehat{V}^{(t)}$. Hence, the total communication from CP to all the servers in the t-th iteration is $O(|\Omega|r)$. Next, we make a critical observation that update $\widehat{V}^{(t+1)}$ can be computed by adding certain messages from each server. Message from server k to CP is of size $O(|\Omega^k|r^2)$. Hence, total communication complexity in each round is $O(|\Omega|r^2)$ and total number of rounds is $O(\log(\|M\|_F/\zeta))$.

We now combine the above given observations to provide error bounds and communication complexity of our distributed PCA algorithm:

Theorem 4.1. *Let the $n \times d$ matrix M be distributed over s servers according to the row-partition model. Let $m \ge \frac{C}{\gamma}\frac{nr^3}{\epsilon^2}\kappa^2 \log(n)\log^2(\frac{||M_r||}{\zeta})$. Then, the distributed LELA algorithm on completion will leave matrices $\widehat{U}^{(t+1)}_{\mathbf{r_k}}$ at server k and $\widehat{V}^{(t+1)}$ at CP such that the following holds (w.p. $\ge 1-\gamma$): $||M - \widehat{U}^{(t+1)}(\widehat{V}^{(t+1)})^T|| \le ||M - M_r|| + \epsilon||M - M_r||_F + \zeta$, where $\widehat{U}^{(t+1)} = \sum_k \widehat{U}^{(t+1)}_{\mathbf{r_k}}$. This algorithm has a communication complexity of $O(ds + |\Omega|r^2) = O(ds + \frac{nr^5\kappa^2}{\epsilon^2}\log^2(\frac{||M_r||}{\zeta}))$ real numbers.*

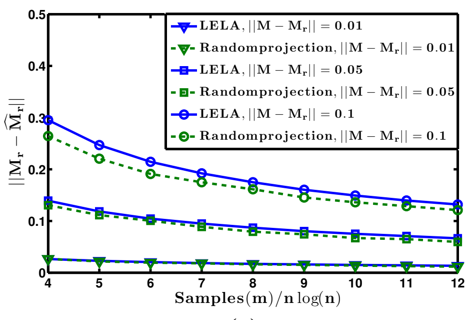
As discussed above, each update to $\widehat{V}^{(t)}$ and $\widehat{U}^{(t)}$ are computed exactly as given in the WAltMin procedure (Sub-routine 2). Hence, error bounds for the algorithm follows directly from Theorem 3.1. Communication complexity bounds follows by observing that $|\Omega| \le 2m$ w.h.p.

5 Simulations

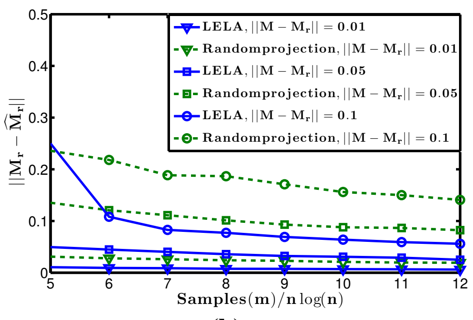
In this section we present some simulation results on synthetic data to show the error performance of the algorithm 1. We consider random matrices of size 1000 by 1000 and rank-5. M_r is a rank 5 matrix with all singular values 1. We consider two cases one in which M_r is incoherent and other in which M_r is coherent. Recall that a $n \times d$ rank-r matrix M_r is an incoherent matrix if $\|(U^*)^i\|^2 \le \frac{\mu_0 r}{n}, \forall i$ and $\|(V^*)^j\|^2 \le \frac{\mu_0 r}{d}, \forall j$, where SVD of M_r is $U^*\Sigma^*(V^*)^T$.

To generate matrices with varying incoherence parameter μ_0, we use the power law matrices model [4]. The input to algorithms is the matrix $M = M_r + Z$, where Z is a Gaussian noise matrix with $||Z|| = 0.01, 0.05$ and 0.1. Correspondingly Frobenius norm of Z is $||Z|| * \sqrt{1000}/2$, which is $0.16, 0.79$ and 1.6 respectively.

In the first plot we compare the error $||M_r - \widehat{M_r}||$ of our algorithm LELA 1 with the random projections based algorithm [13, 3]. We use the matrix with each entry an independent Gaussian

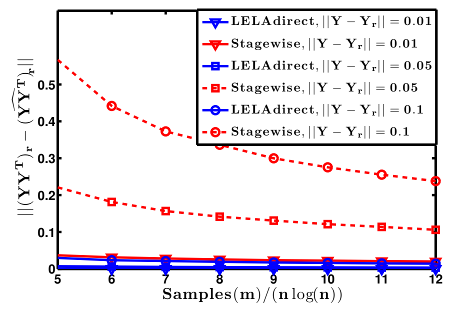


(a)

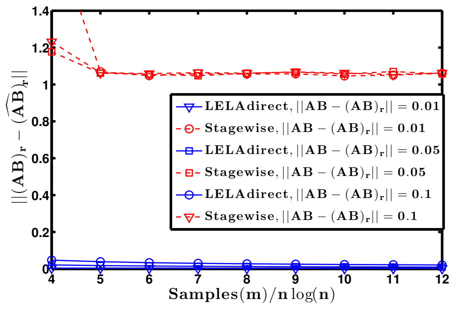


(b)

Figure 1: Figure plots how the error $||M_r - \widehat{M_r}||$ decreases with increasing number of samples m for different values of noise $||M - M_r||$, for incoherent and coherent matrices respectively. Algorithm LELA 1 is run with m samples and Gaussian projections algorithms is run with corresponding dimension of the projection $l = m/n$. Computationally LELA algorithms takes $O(nnz(M) + m\log(n))$ time for computing samples and Gaussian projections algorithm takes $O(nm)$ time. **(a)**:For same number of samples both algorithms have almost the same error for incoherent matrices. **(b):** For coherent matrices clearly the error of LELA algorithm (solid lines) is much smaller than that of random projections (dotted lines).



(a)



(b)

Figure 2: **(a)**:Figure plots the error $||(YY^T)_r - \widehat{(YY^T)}_r||$ for LELA direct 3.3 and Stagewise algorithm for incoherent matrices. Stagewise algorithm is first computing rank-r approximation $\widehat{Y}_r$ of Y using algorithm 1 and setting the low rank approximation $\widehat{(YY^T)}_r = \widehat{Y}_r\widehat{Y}_r^T$. Clearly directly computing low rank approximation of YY^T has smaller error. **(b):**Error $||(AB)_r - \widehat{(AB)}_r||$ for LELA direct 3.3 and Stagewise algorithm.

random variable as the sketching matrix, for the random projection algorithm. Other choices are Walsh-Hadamard based transform [21] and sparse embedding matrices [5].

We compare the error of both algorithms as we vary number of samples m for algorithm 1, equivalently varying the dimension of random projection $l = m/n$ for the random projection algorithm.

In figure 1 we plot the error $||M_r - \widehat{M_r}||$ with varying number of samples m for both the algorithms. For incoherent matrices we see that LELA algorithm has almost the same accuracy as the random projection algorithm 1(a). But for coherent matrices we notice that in figure 1(b) LELA has significantly smaller error compared to random projections.

Now we consider the setting of computing low rank approximation of YY^T given Y using algorithm LELA direct discussed in section 3.3 with sampling (3). In figure 2(a) we compare this algorithm with a stagewise algorithm, which computes low rank approximation $\widehat{Y}_r$ from Y first and the rank-r approximation is $\widehat{Y}_r\widehat{Y}_r^T$. As discussed in section 3.3 direct approximation of YY^T has less error than that of computing $\widehat{Y}_r\widehat{Y}_r^T$. Similarly in 2(b) we consider the case where A and B are two rank $2r$ matrices with AB being a rank r matrix. Here the top r dimensional row space of A is orthogonal to the top r dimensional column space of B. Hence stagewise algorithm, that computes rank r approximation of A and B first and then multiply will have high error as compared to that of LELA direct.

References

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 611–618. ACM, 2001.

[2] S. Bhojanapalli, P. Jain, and S. Sanghavi. Tighter low-rank approximation via sampling the leveraged element. *arXiv preprint arXiv:1410.3886*, 2014.

[3] C. Boutsidis and A. Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, 2013.

[4] Y. Chen, S. Bhojanapalli, S. Sanghavi, and R. Ward. Coherent matrix completion. In *Proceedings of The 31st International Conference on Machine Learning*, pages 674–682, 2014.

[5] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 81–90. ACM, 2013.

[6] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[7] P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. In *SODA*, volume 3, pages 223–232, 2003.

[8] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[9] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 316–326. Springer, 2006.

[10] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453. SIAM, 2013.

[11] A. Frieze, R. Kannan, and S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 370–378, Nov 1998.

[12] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *SODA*, pages 707–717. SIAM, 2014.

[13] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[14] S. Har-Peled. Low rank matrix approximation in linear time. *Manuscript. http://valis.cs.uiuc.edu/sariel/papers/05/lrank/lrank.pdf*, 2006.

[15] R. Kannan, S. S. Vempala, and D. P. Woodruff. Principal component analysis and higher correlations for distributed data. In *Proceedings of The 27th Conference on Learning Theory*, pages 1040–1057, 2014.

[16] Y. Liang, M.-F. Balcan, and V. Kanchanapally. Distributed pca and k-means clustering. In *The Big Learning Workshop at NIPS*, 2013.

[17] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[18] M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[19] N. H. Nguyen, T. T. Do, and T. D. Tran. A fast and efficient algorithm for low-rank approximation of a matrix. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 215–224. ACM, 2009.

[20] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

[21] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.