Fully Distributed Robust Singular Value Decomposition

István Hegedűs*, Márk Jelasity*†, Levente Kocsis*‡ and András A. Benczúr‡
*University of Szeged, Hungary
†MTA-SZTE Research Group on Artificial Intelligence, Hungary
‡Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)

***Abstract*—Low-rank matrix approximation is an important tool in data mining with a wide range of applications including recommender systems, clustering, and identifying topics in documents. The problem we tackle is implementing singular value decomposition (SVD)—a popular method for low rank approximation—in large fully distributed P2P systems in a robust and scalable manner. We assume that the matrix to be approximated is stored in a large network where each node knows one row of the matrix (personal attributes, documents, media ratings, etc). In our P2P model, we do not allow this personal information to leave the node, yet we want the nodes to collaboratively compute the SVD. Methods applied in large scale distributed systems such as synchronized parallel gradient search or distributed iterative methods are not preferable in our system model due to their requirements of synchronized rounds or their inherent issues with load balancing. Our approach overcomes these limitations with the help of a distributed stochastic gradient search in which the personal part of the decomposition remains local, and the global part (e.g., movie features) converges at all nodes to the correct value. We present a theoretical derivation of our algorithm, as well as a thorough experimental evaluation of real and synthetic data as well. We demonstrate that the convergence speed of our method is competitive while not relying on synchronization and being robust to extreme failure scenarios.**

***Keywords*—data mining; matrix factorization; online learning; stochastic gradient descent; singular value decomposition; privacy**

I. Introduction

Finding a low-rank decomposition of a matrix is an essential tool in data mining and information retrieval [1]. Prominent applications include recommender systems [2], information retrieval via Latent Semantic Indexing [3], [4], Kleinberg's HITS algorithm for graph centrality [5], clustering [6], [7], and learning mixtures of distributions [8], [9].

Collaborative filtering forms one of the most prominent applications of low rank matrix approximation. To implement a recommender algorithm, one can define a matrix A of dimensions $m \times n$ with one row assigned to one of m users, and one column assigned to one movie out of a set of n movies. The essence of the ratings matrix A is given by a low rank decomposition $A \approx XY^T$, where matrices X and Y^T are of dimensions $m \times k$ and $k \times n$, respectively, and where k is small [10]. A row of X can be interpreted as a compressed representation (features) of the corresponding user, and a column of Y^T contains features of a movie. This representation can be applied to calculate missing ratings to offer recommendations. In addition to recommender systems, low rank approximation finds applications in summarizing adjacency matrices for social network analysis or term-document matrices for text classification.

Data such as movie ratings, lists of friends, or personal documents is often very sensitive. It is argued in [11] that it is paramount for privacy that users keep their data (e.g. ratings and user factors) local. Instead of uploading such data to cloud servers, it is a natural idea to store it only on personal devices and process it in place in a fully distributed way suitable for P2P services. This would increase the level of privacy and remove any dependence on central infrastructure and services. However, such an algorithm has to be extremely robust as networks of personal devices are unreliable with users entering and leaving dynamically. In addition, the communication costs should remain affordable as well.

A. Contribution

To meet these goals, we implement singular value decomposition (SVD), an approach to low rank decomposition with the attractive property that the matrices X and Y in the decomposition consist of orthogonal vectors. We present a stochastic gradient descent (SGD) algorithm to find the SVD, where several instances of the matrix Y perform a random walk in the network visiting the data (the rows of A), while matrices A and X are stored in a fully distributed way with each row at different nodes. The rows of matrices A and X are accessed only locally by the node that stores them. Note that the public matrix Y carries no sensitive information. When Y visits a node, it gets updated based on the local row of A, and the local row of X gets updated as well.

To the best of our knowledge, we present the first fully distributed SGD algorithm that keeps X and A strictly local. As a special feature, our algorithm updates several versions of Y by sending them around over the network, resulting in a convergence much faster than a single instance of SGD for SVD. The algorithm is fully asynchronous: messages can be delayed or lost. We only rely on random walks that can perform every transition in a bounded time. We show that the only stable fix points of the search algorithm correspond to the SVD. In addition, we perform an experimental analysis and

978-1-4799-6201-3/14/$31.00

demonstrate the convergence speed and the scalability of the protocol.

B. Related work

Calculating the SVD is a well studied problem. One approach is based on considering it as an optimization problem (see Section II) and using *gradient search* to solve it [12]. In general, parallel versions of gradient search often assume the MapReduce framework [13] or a similar, less constrained, but still centralized model [14]. In these approaches, partial gradients are calculated over batches of data in parallel, and these are either applied to blocks of X and Y in the map phase or summed up in the reduce phase. Zinkevich et al. propose a different approach in which SGD is applied on batches of data and the resulting models are then combined [15]. Gemulla et al. [16] propose an efficient parallel technique in which blocks of X and Y are iteratively updated while only blocks of Y are exchanged. In contrast to these approaches, we work with fully distributed data: we do not wish to make X public, and we do not rely on central components or synchronization, a set of requirements ruling out the direct application of earlier approaches.

Another possibility is using fully distributed *iterative methods*. GraphLab [17] supports iterative methods for various problems including SVD. In these approaches, the communication graph in the network is defined by the non-zero elements of matrix A, in other words, A is stored as edge-weights in the network. This is feasible only if A is (very) sparse and well balanced; a constraint rarely met in practice. In addition, iterative methods need access to A^T as well, which violates our constraint that the rows of A are not shared. Using the same edge-weight representation of A, one can implement another optimization approach for matrix decomposition: an iterative optimization of subproblems over overlapping local subnetworks [18]. The drawback of this approach is, again, that the structure of A defines the communication network and access to A^T is required. The approach of Ling et al. [19] also needs global information: in each iteration step a global average needs to be calculated.

The first fully distributed algorithm for spectral analysis was given in [20] where data is kept at the compute nodes and partial results are sent through the network. That algorithm, however, only computes the user side singular vectors but not the item side ones and hence insufficient, for example, to provide recommendations. Similarly, [21] computes the low rank approximation but not the decomposition. This drawback is circumvented in [22] by assigning compute nodes not just for users but for items as well. In their gradient descent algorithm, item vectors are also stored at peers, which means that all items must know the ratings they receive, and this violates privacy.

We overcome the limitations of earlier fully distributed and gossip SVD algorithms by providing an algorithm without item-based processing elements, i.e. our algorithm is fully distributed in the sense that processing is exclusively done at the users, who may access their own ratings and approximations of the item factor vectors.

II. Problem Definition

A. Low–Rank and Singular Value Decomposition

The problem we tackle is rank–k matrix approximation. We are given a matrix $A \in \mathbb{R}^{m\times n}$. Matrix A holds our raw data, such as ratings of movies by users, or word statistics in documents. We are looking for matrices $X \in \mathbb{R}^{m\times k}$ and $Y \in \mathbb{R}^{n\times k}$ such that the error function

$$J(X,Y) = \frac{1}{2}\|A - XY^T\|_F^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - \sum_{l=1}^{k} x_{il}y_{jl})^2 \quad (1)$$

is minimized. We say that the matrix XY^T for which this error function is minimized is an optimal rank-k approximation of A. Clearly, matrices X and Y^T—and hence XY^T—have a rank of at most k. Normally we select k such that $k \ll \min(m,n)$ in order to achieve a significant compression of the data. As a result, matrices X and Y^T can be thought of as high level features (such as topics of documents, or semantic categories for words) that can be used to represent the original raw data in a compact way.

Singular value decomposition (SVD) is related to the above matrix decomposition problem. The SVD of a matrix $A \in \mathbb{R}^{m\times n}$ involves two orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T \quad (2)$$

where the columns of the matrices $U = [u_1 u_2 \cdots u_m]$ and $V = [v_1 v_2 \cdots v_n]$ are the left and right singular vectors, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix containing the singular values $\sigma_1, \sigma_2, \ldots, \sigma_r \geq 0$ of A ($r = \min(m,n)$). The relationship of SVD and low rank decomposition is that $U_k\Sigma_k V_k^T$ is an optimal rank-k approximation of A, where the matrices $U_k \in \mathbb{R}^{m\times k}$ and $V_k \in \mathbb{R}^{n\times k}$ are derived from U and V by keeping the first k columns, and $\Sigma_k \in \mathbb{R}^{k\times k}$ is derived from Σ by keeping the top left $k \times k$ rectangular area, assuming without loss of generality that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ [23].

Our goal in this paper is to find $X^* = U_k\Sigma_U$ and $Y^* = V_k\Sigma_V$ such that Σ_U and Σ_V are diagonal and $\Sigma_U\Sigma_V = \Sigma_k$. In other words, although X^* and Y^* are not uniquely defined, we require them to contain orthogonal columns that are scaled versions of left and right singular vectors of A, respectively.

B. System Model and Data Distribution

Having defined the computational problem, we need to elaborate on our assumptions on the network environment and the distribution of the data. As our system model we consider a network of nodes that are individual computational units (e.g., personal computers, smart phones, etc.) and have unique addresses. The number of nodes is potentially extremely large. Every node can communicate to other nodes via passing messages provided the address of the target node is known locally. We assume that there is a peer sampling service in

Algorithm 1 P2P low–rank factorization at node i

```
1: a_i                                              ▷ row i of A
2: initialize Y
3: initialize x_i                                   ▷ row i of X
4: loop
5:     wait(Δ)
6:     p ← selectPeer()
7:     send Y to p
8: end loop
9: procedure ONRECEIVEY(Ỹ)
10:    Y ← Ỹ
11:    (Y, x_i) ← update(Y, x_i, a_i)
12: end procedure
```

our system which can provide addresses of live nodes from the network selected uniformly at random (see e.g. [24] for a fully distributed implementation). In addition, messages can be lost or delayed and the nodes can leave the network and join again without prior notice. When a node rejoins the network it retains the state it had when leaving the network.

As for the data distribution, we assume that each node has exactly one row (but note that our algorithms can be applied—and in fact profit from it—if a node has several rows). Our data distribution model is motivated by application scenarios in which potentially sensitive data is available locally, such as private documents or ratings that naturally define rows of a matrix A, but where a decomposition needs to be found collectively without blatantly exposing this private data.

III. Algorithm

Our algorithm has its roots in the GoLF framework [25]. Algorithm 1 contains a version of the GoLF algorithm adopted to our problem. Each node i has its own approximation of the full matrix Y^* and an approximation of row i of X^*: x_i. Thus, the nodes collectively store one version of the matrix X (the approximation of X^*) distributed just like matrix A with each node storing one row. At the same time, at every point in time every node has a possibly different approximation of the full matrix Y^* locally.

These different approximations of Y^* perform random walks in the network and get updated at the visited nodes using the local data of the given node (a row of A and X). First, each node i in the network initializes Y and x_i uniformly at random from the interval $[0, 1]$. The nodes then periodically send their current approximation Y to a randomly selected peer from the network. The period is denoted by Δ. To select a random peer, we rely on a peer sampling service as mentioned in Section II-B. When receiving an approximation $\tilde{Y}$ (see procedure ONRECEIVEY) the node stores this approximation and subsequently it updates both Y and x_i using a stochastic gradient rule.

The algorithm involves periodic message sending at every node with a period of Δ. Later on, when we refer to one *iteration* or *round* of the algorithm, we simply mean a time interval of length Δ. Note that we do not require any synchronization of rounds over the network. Messages are sent independently, and they can be delayed or dropped as well. We require only that the delays are bounded and that the message drop probability is less than one. This allows the random walks to progress.

Algorithm 2 rank-k update at node i

```
1: η                                                ▷ learning rate
2: procedure UPDATE(Y, x_i, a_i)
3:     err ← a_i − x_i Y^T
4:     x'_i ← x_i + η · err · Y
5:     Y' ← Y + η · err^T · x_i
6:     return (Y', x'_i)
7: end procedure
```

Algorithm 1 requires an implementation of the procedure UPDATE. We will now elaborate on two different versions that implement different stochastic gradient update rules.

A. Update Rule for General Rank–k Factorization

Let us first consider the error function given in equation (1) and derive an update rule to optimize this error function. The partial derivatives of $J(X, Y)$ by X and Y are

$$\frac{\partial J}{\partial X} = (XY^T - A)Y, \qquad \frac{\partial J}{\partial Y} = (YX^T - A^T)X. \quad (3)$$

Since only x_i is available at node i, the gradient is calculated only w.r.t. x_i instead of X. Accordingly, the stochastic gradient update rule with a learning rate η can be derived by substituting x_i as shown in Algorithm 2. Although function J is not convex, it has been shown that all the local optima of J are also global [23]. Thus, for a small enough η, any stable fix point of the dynamical system implemented by Algorithm 1 with the update rule in Algorithm 2 is guaranteed to be a global optimum.

B. Update Rule for Rank–k SVD

Apart from minimizing the error function given in equation (1) let us now also set the additional goal that the algorithm converges to X^* and Y^*, as defined in Section II. This is indeed a harder problem: while (X^*, Y^*) minimizes (1), any pair of matrices (X^*R^{-1}, Y^*R^T) will also minimize it for any invertible matrix $R \in \mathbb{R}^{k\times k}$, so Algorithm 2 will not be sufficient.

From now on, we will assume that the non-zero singular values of A are all unique, and that the rank of A is at least k. That is, $\sigma_1 > \cdots > \sigma_k > 0$. This makes the discussion simpler, but these assumptions can be relaxed, and the algorithm is applicable even if these assumptions do not hold.

Our key observation is that any optimal rank-1 approximation $X_1 Y_1^T$ of A is such that $X_1 \in \mathbb{R}^{m\times 1}$ contains the (unnormalized) left singular vector of A that belongs to σ_1, the largest singular value of A. Similarly, Y_1 contains the corresponding right singular vector. This is because for any

Algorithm 3 rank-k SVD update at node i

η ▷ learning rate
procedure UPDATE(Y, x_i, a_i)
  $a'_i \leftarrow a_i$
  for ℓ = 1 to k **do** ▷ y_ℓ : column ℓ of Y
    $\text{err} \leftarrow a'_i - x_{i\ell} \cdot y_\ell^T$
    $x'_{i\ell} \leftarrow x_{i\ell} + \eta \cdot \text{err} \cdot y_\ell$
    $y'_\ell \leftarrow y_\ell + \eta \cdot \text{err}^T \cdot x_{i\ell}$
    $a'_i = a'_i - x_{i\ell} \cdot y_\ell^T$
  end for
  return (Y', x'_i)
end procedure

optimal rank-k approximation XY^T there is an invertible matrix $R \in \mathbb{R}^{k\times k}$ such that $X = X^*R$ and $Y^T = R^{-1}Y^{*T}$ [23]. For $k = 1$ this proves our observation because, as defined in Section II-A, $X^* \sim u_1$ and $Y^* \sim v_1$. Furthermore, for $k = 1$,

$$X_1 Y_1^T = X^* Y^{*T} = \sigma_1 u_1 v_1^T, \tag{4}$$

which means that (using equation (2)) we have

$$A - X_1 Y_1^T = \sum_{i=2}^{r} \sigma_i u_i v_i^T. \tag{5}$$

Thus, a rank-1 approximation of the matrix $A - X_1 Y_1^T$ will reveal the direction of the singular vectors corresponding to the second largest singular value σ_2. A naive approach based on these observations would be to first compute $X_1 Y_1^T$, a rank-1 approximation of A. After this has converged, we could compute a rank-1 approximation $X_2 Y_2^T$ of $A - X_1 Y_1^T$. This would give us a rank-2 approximation of A containing the first two left and right singular vectors, since according to the above observations $[X_1, X_2][Y_1, Y_2]^T$ is in fact such a rank-2 approximation. We could repeat this procedure k times to get the desired decomposition X^* and Y^* by filling in one column at a time sequentially in both matrices.

A more efficient and robust approach is to let all rank-1 approximations in this sequential naive approach evolve at the same time. Intuitively, when there is a reasonable estimate of the singular vector corresponding to the largest singular value, then the next vector can already start progressing in the right direction, and so on. This idea is implemented in Algorithm 3.

C. Synchronized Rank–k SVD

As a baseline method in our experimental evaluation, we will use a synchronized version of Algorithm 1 with the update rule in Algorithm 3. In this version (shown in Algorithm 4), the rank-1 updates are done over the entire matrix A at once, and there is only one central version of the approximation of Y as opposed to the several independent versions we had previously. As in Algorithm 1, the matrices X and Y are initialized with uniform random values from $[0, 1]$. Note that this algorithm listing uses a different notation: here x_ℓ denotes the ℓ-th column, not the ℓ-th row.

Algorithm 4 Iterative synchronized rank–k SVD

A ▷ The matrix to be factorized
η ▷ learning rate
initialize Y
initialize X
while not converged **do**
  $A' = A$
  for ℓ = 1 to k **do**
    $\text{err} \leftarrow A' - x_\ell \cdot y_\ell^T$ ▷ x_ℓ : column ℓ of X
    ▷ y_ℓ : column ℓ of Y
    $x'_\ell \leftarrow x_\ell + \eta \cdot \text{err} \cdot y_\ell$
    $y'_\ell \leftarrow y_\ell + \eta \cdot \text{err}^T \cdot x_\ell$
    $A' = A' - x_\ell \cdot y_\ell^T$
  end for
  $X = X'; \quad Y = Y'$
end while

TABLE I. THE MAIN PROPERTIES OF THE REAL DATA SETS

	Iris	Pendigits	Segmentation
Number of instances (m)	150	10992	2310
Number of features (n)	4	16	19
Minimal k such that $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2 > 0.9$	2	10	5

Note that—although it is formulated as a centralized sequential algorithm—this algorithm can easily be adopted for the MapReduce framework, where the mappers compute parts of the gradients (e.g., the gradients of the rows of A as in Algorithm 3) while the reducer sums up the components of the gradient and executes the update steps.

IV. EXPERIMENTS

Here we demonstrate various properties of our algorithm including convergence speed, scalability and robustness. Our testbed of matrices includes standard machine learning test data sets as well as synthetic matrices with controllable properties.

In case of the distributed algorithms the number of nodes in the network equals the number of rows of the matrix to be factorized, since every node has exactly one row of the matrix. We used the PeerSim [26] simulator with the event-based engine, and we implemented the peer sampling service by the NEWSCAST [24] protocol. Our implementation is publicly available.[1]

A. Algorithms

Here we provide names for the algorithms we include in our experiments. Algorithm 1 with the update rule in Algorithm 3 (our main contribution) will be referred to as Fully Distributed SVD (FUDISVD). Replacing Algorithm 3 with Algorithm 2 we get Fully Distributed Low Rank Decomposition (FUDILRD). Recall that this algorithm will converge

[1] http://rgai.inf.u-szeged.hu/download/p2p14/svdsrc.zip, https://github.com/RobertOrmandi/Gossip-Learning-Framework/tree/multicore

to a rank-k decomposition that is not necessarily the SVD of A. Algorithm 4 will be called Gradient SVD (GRADSVD). Recall that this algorithm can be parallelized: for example, the gradient can be calculated row by row and then summed up to get the full gradient.

Finally, we introduce a baseline algorithm, Stochastic Gradient SVD (SGSVD). This algorithm uses the update rule in Algorithm 3 but we have only a single approximation Y at any point in time, and there is only one process, which repeatedly gets random rows of A and then applies the update rule in Algorithm 3 to the current approximation Y and the corresponding row of X.

B. Error measures

a) Cosine similarity: To measure the difference between the correct and the approximated singular vectors, we used cosine similarity, because it is not sensitive to the scaling of the vectors (recall, that our algorithm does not guarantee unit-length columns in X and Y). The formula to measure the error of a rank-k decomposition XY^T is

$$\text{Error}(X,Y) = \frac{1}{2k}\sum_{i=1}^{k} 1 - \left|\frac{y_i^T v_i}{\|y_i\|}\right| + 1 - \left|\frac{x_i^T u_i}{\|x_i\|}\right|, \quad (6)$$

where x_i, y_i, u_i and v_i are column vectors of X, Y, U_k and V_k, respectively. Matrices U_k and V_k are orthogonal matrices defined in Section II.

b) Frobenius norm: Another measure of error is given by function $J(X,Y)$ defined in equation (1). The advantage of this measure is that it can be applied to Algorithm 2 as well. However, obviously, this error measure does not reflect whether the calculated matrices X and Y contain scaled singular vectors or not; it simply measures the quality of rank-k approximation. On the plots we call this error measure FNORM.

C. Data sets

c) Synthetic data: We included experiments on synthetic data so that we can evaluate the scalability and the fault tolerance of our method in a fully controlled way. We first generated random singular vectors producing matrices U and V with the help of the butterfly technique [27]. Since matrices to be decomposed often originate from graphs, and since the node degrees and the spectrum of real graphs usually follow a power law distribution [28], [29], the expected singular values in the diagonal of Σ were generated from a Pareto distribution with parameters $x_m = 1$ and $\alpha = 1$. This way we construct a matrix $A = U\Sigma V^T$ where we know and control the singular vectors and values.

d) Real data: These matrices were constructed from data sets from the well known UCI [30] machine learning repository. In these applications the role of SVD is dimensionality reduction and feature extraction. The selected data sets are the Iris, the Pendigits (Pen-Based Recognition of Handwritten Digits) and the Segmentation (Statlog (Image Segmentation)) data sets. Parameter k was set so that the approximation has at least 90% of the information in the data set, that is, we set the minimal k such that $\sum_{i=1}^{k}\sigma_i^2 / \sum_{i=1}^{n}\sigma_i^2 > 0.9$ [31]. Table I illustrates the main properties of the selected data sets. In order to be able to compute the error over these matrices, we computed the exact singular value decomposition using the Jama [32] library.

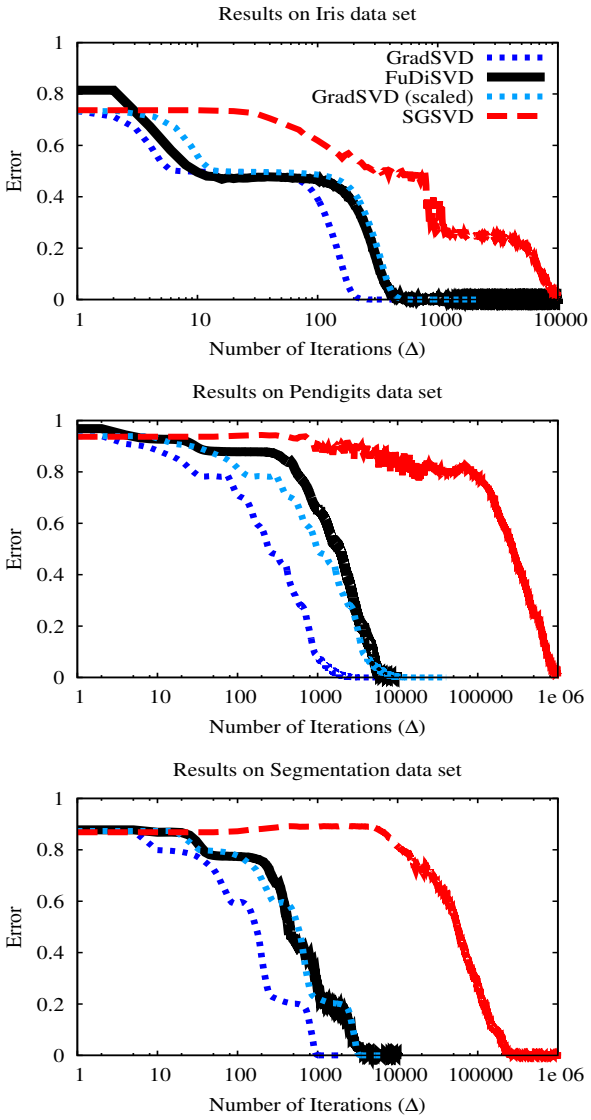

Fig. 1. Convergence on the real data sets. Error is based on cosine similarity. In the scaled version of GRADSVD the number of iterations is multiplied by $\log_{10} m$ (see text).

D. Convergence

The experimental results are shown in Figures 1 and 2. We tested the convergence speed of the algorithms over the real data sets with parameter k set according to Table I.

Figure 1 illustrates the deviation from the exact singular vectors. GRADSVD is the fastest to converge, however, it either needs a central database to store A, or it requires a synchronized master-slave communication model when parallelized. We also show a reference curve that is calculated by

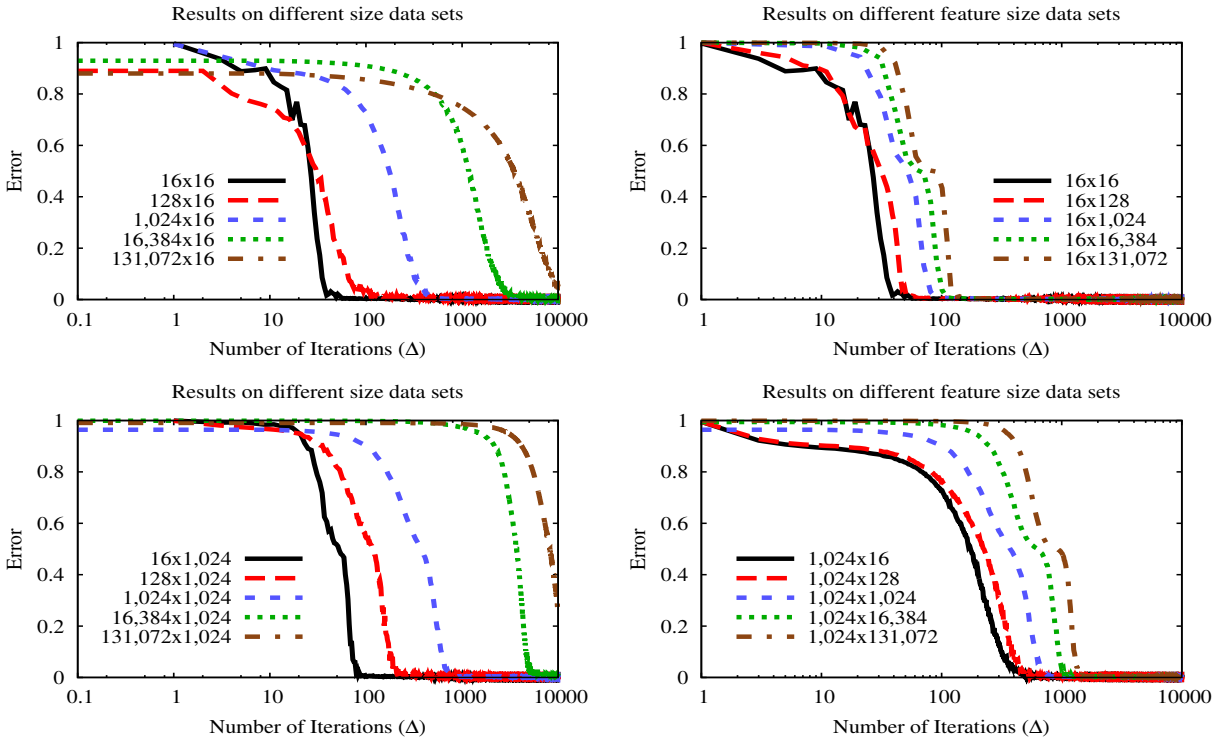

Fig. 3. Results on synthetic data sets using networks of different dimensions. We set $k = 1$, and all the matrices had a rank of 16.

scaling the number of iterations by $\log_{10} m$ for GRADSVD. The motivation is a more scalable potential implementation of GRADSVD in which there is a hierarchical communication structure in place where nodes propagate partial sums of the gradient up a tree (with a branching factor of 10) instead of each node communicating with a central server as in [33]. This curve almost completely overlaps with that of FUDISVD, our fully distributed robust algorithm.

We also illustrate the speedup w.r.t. SGSVD. The major difference between SGSVD and FUDISVD is that in FUDISVD there are m different approximations of Y all of which keep updating the rows of X simultaneously, while in SGSVD there is only one version of Y. Other than that, both algorithms apply exactly the same gradient update rule. In other words, in FUDISVD any row of X experiences m times as many updates in one unit of time.

Figure 2 illustrates the difference between FUDILRD and FUDISVD. Both optimize the same error function (that is shown on the plots), however, FUDISVD introduces the extra constraint of aiming for the singular vectors of A. Fortunately this extra constraint does not slow down the convergence significantly in the case of the data sets we examined, although it does introduce a bumpier convergence pattern. The reason is that the singular vectors converge in a sequential order, and the vectors that belong to smaller singular values might have to be completely re-oriented when the singular vectors that preceed them in the order of convergence have converged.

E. Scalability

For evaluating the scalability of FUDISVD we generated a range of synthetic matrices of various sizes using the method described earlier. Figure 3 shows the outcome of the experiments. Clearly, the method is somewhat more sensitive to changing the number of nodes (that is, to varying the first dimension m) than to varying the second dimension n (the length of a row). This is not surprising as the full row of A is always used at once in an update step, irrespective of its length, whereas a larger m requires visiting more nodes, which takes more iterations.

However, when m is large, we can apply *sampling techniques* [34]. That is, we can consider only a small sample of the network drawn uniformly or from an appropriately biased distribution and calculate a high quality SVD based on that subset. To illustrate such sampling techniques, we implemented uniform sampling. When we wish to select a given proportion p of the nodes, each node decides locally about joining the sample with a probability p. The size of the resulting sample will follow the binomial distribution $B(N, p)$.

Figure 4 shows our experimental results with $p = 1/2$ and $p = 1/3$. Clearly, while communication costs overall are decreased proportionally to the sample size, on our benchmark both precision and convergence speed remain almost unchanged. The only exception is the Iris data set. However, that is a relatively small data set over which the variance of our uniform sampling method is relatively high (note, for example, that the run with $p = 1/3$ resulted in a better performance than with $p = 1/2$).

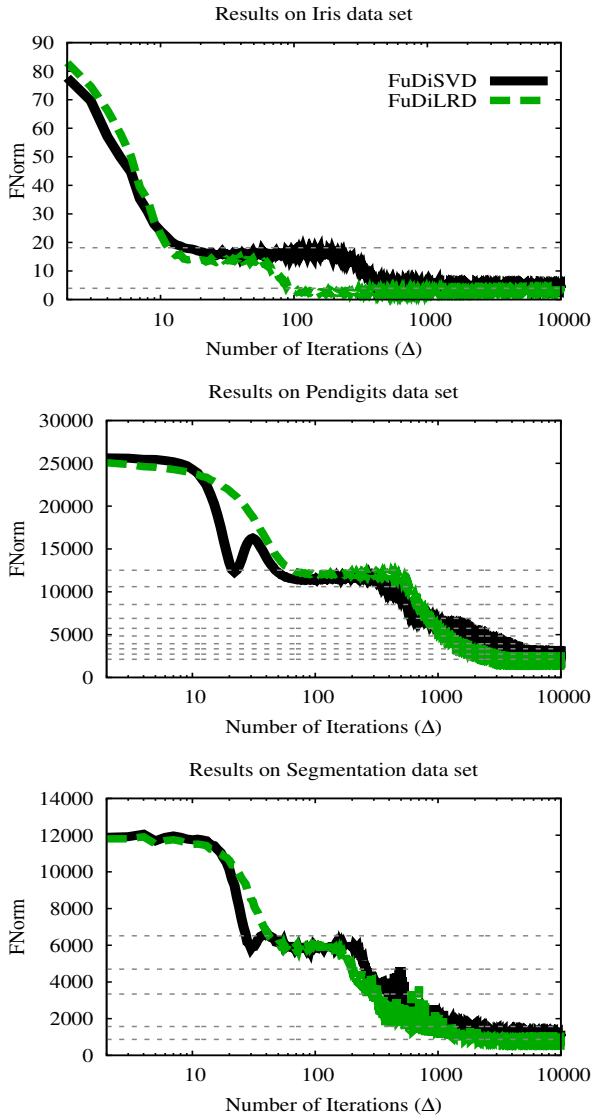

Fig. 2. Convergence on the real data sets. Error is based on the Frobenius norm. Horizontal dashed lines in top-down order show the FNORM value for the optimal rank-i approximations for $i = 1, \ldots, k$.

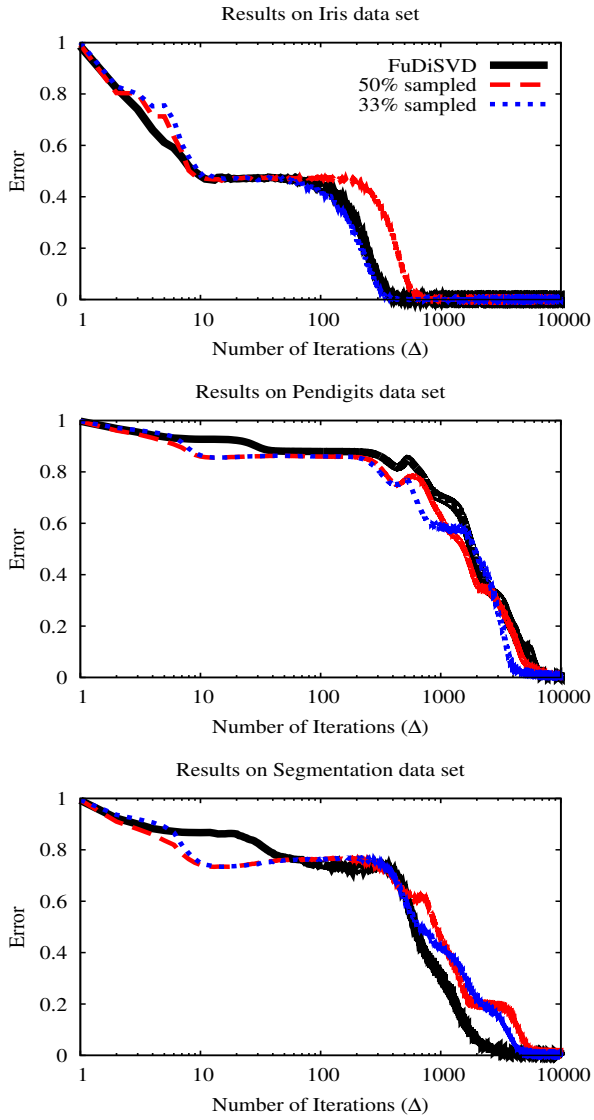

Fig. 4. Results when only the 50/33% randomly sampled instances were used from the data set.

F. Failure Scenarios

We used two different failure scenarios: a mild and a hard one. In the two scenarios message delay was drawn uniformly at random from between Δ and 5Δ or 10Δ time units, respectively. Messages were dropped with 20% or 50% probability, respectively.

Node churn was modeled taking into account statistics from a BitTorrent trace [35] as well as known empirical findings [36]. We draw the online session length for each node independently from a lognormal probability distribution with parameters $\mu = 5\Delta$ and $\sigma = 0.5\Delta$. Offline session lengths are determined implicitly by fixing the number of nodes that are offline at the same time. For the two scenarios 50% or 80% of the nodes were offline at the same time, respectively.

The results of the algorithms in these extreme failure scenarios can be seen in Figure 5. As we can see, the different types of failures (whether examined separately or combined together) result in only delay on the convergence rate, but the algorithm still converges. This is in sharp contrast with competing approaches that require synchronization, such as GRADSVD, for example.

An interesting observation is that when only churn is modeled, at the beginning of the simulation convergence is actually faster. This effect is due to the fact that since most of the nodes are offline, the effective network is smaller, but this small sample still allows for an approximation of the SVD. However, convergence eventually slows down as full precision cannot be reached without taking most of the nodes into account in the particular matrix we experiment with.

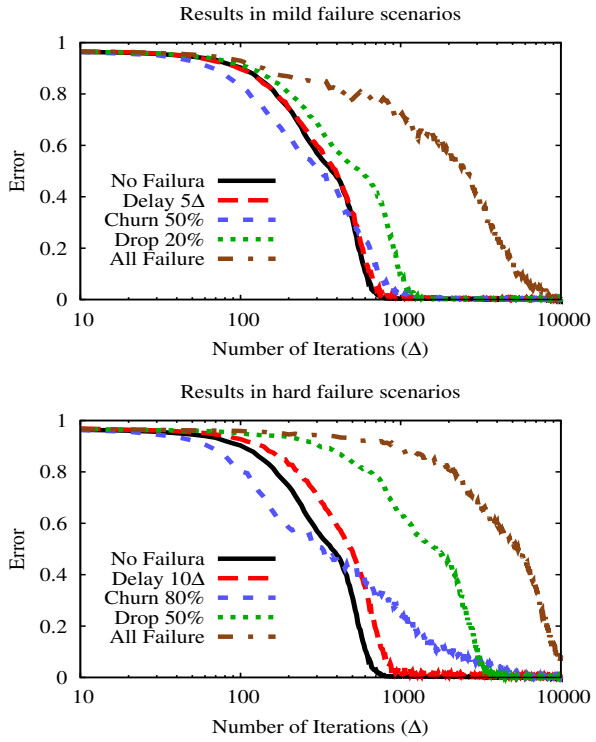

Fig. 5. Results in different failure scenarios using a 1024×1024 synthetic matrix with a rank of 16. We set $k = 1$.

V. Conclusion

In this paper we proposed an SGD algorithm with an update rule that has stable fix points only in the SVD of a matrix A. The output of the algorithm for rank k are two matrices X and Y that contain scaled versions of the first k left and right singular vectors of A, respectively. Matrices X and Y are unique apart from the scaling of the columns.

The most important feature of the algorithms is a P2P sense privacy preservation: we operate over fully distributed data where each network node stores one full row of A as its private data. The output matrix X is also fully distributed: node i that stores row i of A computes row i of X, the private model for the node. Matrices A and X are private in that only the node that stores a given row has access to it. A version of the matrix Y is available in full at all nodes.

Through experimental evaluation we studied the convergence speed of the algorithm, and showed that it is competitive with other gradient methods, that require more freedom for data access. We also demonstrated the remarkable robustness of the method in extreme failure scenarios.

Our future work includes addressing interesting challenges such as dynamically changing data, and investigating methods for further improving the efficiency of the protocol, for example, through sampling from the rows and/or columns of A using several sampling techniques.

Acknowledgments

M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013), by the grant OTKA NK 105645, and the "Momentum - Big Data" grant of the Hungarian Academy of Sciences.

References

[1] Y. Azar, A. Fiat, A. R. Karlin, F. McSherry, and J. Saia, "Spectral analysis of data," in *Proc. 33rd Symposium on Theory of Computing (STOC)*, 2001, pp. 619–626.

[2] P. Drineas, I. Kerenidis, and P. Raghavan, "Competitive recommendation systems," in *Proc. 34th Symposium on Theory of Computing (STOC)*, 2002, pp. 82–90.

[3] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, no. 4, pp. 573–595, 1995.

[4] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," *Journal of Computer and System Sciences*, vol. 61, no. 2, pp. 217–235, 2000.

[5] J. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[6] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering large graphs via the singular value decomposition," *Machine Learning*, pp. 9–33, 2004.

[7] F. McSherry, "Spectral partitioning of random graphs," in *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2001, pp. 529–537.

[8] R. Kannan, H. Salmasian, and S. Vempala, "The spectral method for general mixture models," in *Proc. 18th Annual Conference on Learning Theory (COLT)*, 2005, pp. 444–457.

[9] D. Achlioptas and F. McSherry, "On spectral learning of mixtures of distributions," in *Proc. 18th Annual Conference on Learning Theory (COLT)*, 2005, pp. 458–469.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[11] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *Proc. 2013 ACM SIGSAC Conf. on Comp. and Comm. Security (CCS'13)*. ACM, 2013, pp. 801–812.

[12] G. Gorrell, "Generalized hebbian algorithm for incremental singular value decomposition in natural language processing." in *Proc. 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, D. McCarthy and S. Wintner, Eds. The Association for Computer Linguistics, 2006.

[13] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 281–288.

[14] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *Proc. 29th International Conference on Machine Learning (ICML)*, J. Langford and J. Pineau, Eds. Omnipress, 2012, pp. 81–88.

[15] M. A. Zinkevich, A. Smola, M. Weimer, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, 2010, pp. 2595–2603.

[16] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2011, pp. 69–77.

[17] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in *Conf. on Uncertainty in Artif. Intel.*, 2010.

[18] Y. Liao, P. Geurts, and G. Leduc, "Network distance prediction based on decentralized matrix factorization," in *Proc. 9th Int. IFIP TC 6 Netw. Conf.*, ser. LNCS, M. Crovella, L. Feeney, D. Rubenstein, and S. Raghavan, Eds., vol. 6091. Springer, 2010, pp. 15–26.

[19] Q. Ling, Y. Xu, W. Yin, and Z. Wen, "Decentralized low-rank matrix completion," in *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 2925–2928.

[20] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," in *Proc. 36th Symposium on Theory of Computing (STOC)*. ACM, 2004, pp. 561–568.

[21] S. B. Korada, A. Montanari, and S. Oh, "Gossip pca," in *Proc. ACM SIGMETRICS joint int. conf. on Measurement and modeling of comp. sys.* ACM, 2011, pp. 209–220.

[22] S. Isaacman, S. Ioannidis, A. Chaintreau, and M. Martonosi, "Distributed rating prediction in user generated content streams," in *Proc. Fifth ACM Conf. on Rec. Sys.* ACM, 2011, pp. 69–76.

[23] N. Srebro and T. Jaakkola, "Weighted low-rank approximations," in *Proc. 20th International Conference on Machine Learning (ICML)*. AAAI Press, 2003, pp. 720–727.

[24] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Euro-Par 2009*, ser. LNCS, vol. 5704. Springer, 2009, pp. 523–534.

[25] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[26] A. Montresor and M. Jelasity, "Peersim: A scalable P2P simulator," in *Proc. 9th IEEE Int. Conf. on Peer-to-Peer Comp.* IEEE, 2009, pp. 99–100, extended abstract.

[27] A. Genz, "Methods for generating random orthogonal matrices," in *Proc. Monte Carlo and Quasi-Monte Carlo Methods (MCQMC 1998)*, H. Niederreiter and J. Spanier, Eds. Springer, 1999, pp. 199–213.

[28] M. Mihail and C. Papadimitriou, "On the eigenvalue power law," in *Rand. and Approx. Tech. in Comp. Sci.*, ser. LNCS, J. D. Rolim and S. Vadhan, Eds. Springer, 2002, vol. 2483, pp. 254–262.

[29] F. Chung, L. Lu, and V. Vu, "Eigenvalues of random power law graphs," *Annals of Combinatorics*, vol. 7, no. 1, pp. 21–33, 2003.

[30] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[31] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[32] T. Nis, "JAMA: A Java matrix package," 1999. [Online]. Available: http://math.nist.gov/javanumerics/jama

[33] A. R. Benson, D. F. Gleich, and J. Demmel, "Direct qr factorizations for tall-and-skinny matrices in mapreduce architectures," *CoRR*, 2013.

[34] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix," *SIAM J. Comput.*, vol. 36, no. 1, pp. 158–183, 2006.

[35] J. Roozenburg, "Secure decentralized swarm discovery in Tribler," Master's thesis, Parallel and Distributed Systems Group, Delft University of Technology, 2006. [Online]. Available: http://www.pds.ewi.tudelft.nl/~epema/MSc-theses/MSc-thesis-Roozenburg.pdf

[36] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. 6th ACM SIGCOMM conf. on Internet Measurement (IMC)*. ACM, 2006, pp. 189–202.