Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla
Max-Planck-Institut für Informatik
rgemulla@mpi-inf.mpg.de

Peter J. Haas
IBM Almaden Research
peterh@us.ibm.com

Yannis Sismanis
IBM Almaden Research
syannis@us.ibm.com

Christina Teflioudi
Max-Planck-Institut für Informatik
chteflio@mpi-inf.mpg.de

Faraz Makari
Max-Planck-Institut für Informatik
fmakari@mpi-inf.mpg.de

Abstract

We provide a novel algorithm to approximately factor large matrices with millions of rows, millions of columns, and billions of nonzero elements. Our approach rests on stochastic gradient descent (SGD), an iterative stochastic optimization algorithm. Based on a novel "stratified" variant of SGD, we obtain a new matrix-factorization algorithm, called DSGD, that can be fully distributed and run on web-scale datasets using, e.g., MapReduce. DSGD can handle a wide variety of matrix factorizations and has good scalability properties.

1 Introduction

As Web 2.0 and enterprise-cloud applications proliferate, data mining algorithms need to be (re)designed to handle web-scale datasets. For this reason, low-rank matrix factorization has received much attention in recent years; it is fundamental to a variety of mining tasks that are increasingly being applied to massive datasets [1, 2, 3, 4, 5]. Specifically, low-rank matrix factorizations are effective tools for analyzing "dyadic data" in order to discover and quantify the interactions between two given entities. Successful applications include topic detection and keyword search (where the corresponding entities are documents and terms), news personalization (users and stories), and recommendation systems (users and items). In large applications, these problems can involve matrices with millions of rows (e.g., distinct customers), millions of columns (e.g., distinct items), and billions of entries (e.g., transactions between customers and items). At such massive scales, parallel [6, 7] and distributed [1, 8, 5, 9, 10] algorithms for matrix factorization are essential to achieving reasonable performance. In this paper—an extended abstract of [10]—we provide a novel, effective distributed factorization algorithm based on stochastic gradient descent. Our algorithm is amenable to MapReduce but is also effective in other distributed programming frameworks.

Given a large $m \times n$ matrix $\boldsymbol{V}$ and a small rank r, our goal is to find an $m \times r$ matrix $\boldsymbol{W}$ and an $r \times n$ matrix $\boldsymbol{H}$ such that $\boldsymbol{V} \approx \boldsymbol{WH}$. The quality of such an approximation is described in terms of an application-dependent loss function L, i.e., we seek to find $\operatorname{argmin}_{\boldsymbol{W},\boldsymbol{H}} L(\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{H})$. For example, matrix factorizations used in the context of recommender systems are based on the *nonzero squared loss* $L_{\mathrm{NZSL}} = \sum_{i,j:\boldsymbol{V}_{ij}\neq 0}(\boldsymbol{V}_{ij} - [\boldsymbol{WH}]_{ij})^2$ and usually incorporate regularization terms, user and movie biases, time drifts, and implicit feedback. In the following, we restrict attention to loss functions that, like L_{NZSL}, can be decomposed into a sum of *local losses* over (a subset of) the entries in $\boldsymbol{V}_{ij}$. I.e., we require that the loss can be written as

$$L = \sum_{(i,j)\in Z} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}) \tag{1}$$

for some *training set* $Z \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\}$ and *local loss function* l, where $\boldsymbol{A}_{i*}$ and $\boldsymbol{A}_{*j}$ denote row i and column j of matrix $\boldsymbol{A}$, respectively. Many loss functions used in practice—such as squared loss, generalized Kullback-Leibler divergence (GKL), and L_p regularization—can be decomposed in such a manner [11]. Note that a given loss function L can potentially be decomposed in multiple ways. In this paper, we focus primarily on the class of *nonzero decompositions*, in which $Z = \{(i,j) : \boldsymbol{V}_{ij} \neq 0\}$. Such decompositions naturally arise when zeros represent missing data as in the case of recommender systems. Our algorithms can handle other decompositions as well; see [10, 12].

2 Matrix Factorization with Stochastic Gradient Descent

The goal of SGD is to find the value $\theta^* \in \Re^k$ ($k \geq 1$) that minimizes a given loss $L(\theta)$. The algorithm makes use of noisy observations $\hat{L}'(\theta)$ of $L'(\theta)$, the function's gradient with respect to θ. Starting with some initial value θ_0, SGD refines the parameter value by iterating the stochastic difference equation $\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$, where n denotes the step number and $\{\epsilon_n\}$ is a sequence of decreasing step sizes. Since $-L'(\theta_n)$ is the direction of steepest descent, SGD constitutes a noisy version of gradient descent. To apply SGD to matrix factorization, we set $\theta = (\boldsymbol{W}, \boldsymbol{H})$ and decompose the loss L as in (1) for an appropriate training set Z and local loss function l. Denote by $L_z(\theta) = L_{ij}(\theta) = l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$ the local loss at position $z = (i,j)$. Then $L'(\theta) = \sum_z L'_z(\theta)$ by the sum rule for differentiation. We obtain a noisy gradient estimate by scaling up *just one* of the local gradients, i.e., $\hat{L}'(\theta) = N L'_z(\theta)$, where $N = |Z|$ and the training point z is chosen randomly from the training set. Algorithm 1 uses SGD to perform matrix factorization. Note that each step affects only a *single row* of $\boldsymbol{W}$ and a *single column* of $\boldsymbol{H}$.

Algorithm 1 SGD for Matrix Factorization

```
Require: A training set $Z$, initial values $\boldsymbol{W}_0$ and $\boldsymbol{H}_0$
  while not converged do {step}
    Select a training point $(i,j) \in Z$ uniformly at random.
    $\boldsymbol{W}'_{i*} \leftarrow \boldsymbol{W}_{i*} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{W}_{i*}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$
    $\boldsymbol{H}_{*j} \leftarrow \boldsymbol{H}_{*j} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{H}_{*j}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$
    $\boldsymbol{W}_{i*} \leftarrow \boldsymbol{W}'_{i*}$
  end while
```

3 Distributed Stochastic Gradient Descent

In general, distributing SGD is hard because the individual steps depend on each other: The parameter value of θ_n has to be known before θ_{n+1} can be computed. However, in the case of matrix factorization, the SGD process has some structure that we can exploit.

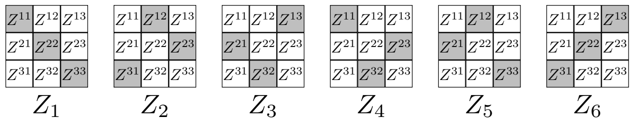
Definition 1 *Two training points $z_1, z_2 \in Z$ are* interchangeable *with respect to a loss function L having summation form* (1) *if for all $\theta \in H$, and $\epsilon > 0$,*

$$\begin{aligned} & L'_{z_1}(\theta) = L'_{z_1}(\theta - \epsilon L'_{z_2}(\theta)) \\ \textit{and} \quad & L'_{z_2}(\theta) = L'_{z_2}(\theta - \epsilon L'_{z_1}(\theta)). \end{aligned} \tag{2}$$

Two disjoint sets of training points $Z_1, Z_2 \subset Z$ are interchangeable with respect to L if z_1 and z_2 are interchangeable for every $z_1 \in Z_1$ and $z_2 \in Z_2$.

For matrix factorization, two training points $z_1 = (i_1, j_1) \in Z$ and $z_2 = (i_2, j_2) \in Z$ are interchangeable with respect to any loss function L having form (1) if they share neither row nor column, i.e., $i_1 \neq i_2$ and $j_1 \neq j_2$. It follows that if two blocks of $\boldsymbol{V}$ share neither rows or columns, then the sets of training points contained in these blocks are interchangeable.

Stratification. The key idea of DSGD is that we can swap the order of consecutive SGD steps that involve interchangeable training points without affecting the final outcome. This allows us to



Z^{11} Z^{12} Z^{13}
Z^{21} Z^{22} Z^{23}
Z^{31} Z^{32} Z^{33}
Z_1
Z_2
Z_3
Z_4
Z_5
Z_6

Figure 1: Strata for a 3×3 blocking of matrix $\boldsymbol{V}$

run SGD in parallel on any set of interchangeable sets of training points; see the discussion below. DSGD thus partitions the training set Z into a set of potentially overlapping "strata" $Z_1, \dots, Z_s$, where each stratum consists of d interchangeable subsets of Z; see Figure 1 for an example. The strata must *cover* the training set in that $\bigcup_{s=1}^{q} Z_s = Z$, but overlapping strata are allowed. The parallelism parameter d is chosen to be greater than or equal to the number of available processing tasks.

There are many ways to stratify the training set into interchangeable strata. In our current work, we perform *data-independent blocking*; more advanced strategies may improve the speed of convergence further. We first randomly permute the rows and columns of $\boldsymbol{V}$, and then create $d \times d$ blocks of size $(m/d) \times (n/d)$ each; the factor matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ are blocked conformingly. This procedure ensures that the expected number of training points in each of the blocks is the same, namely, N/d^2. Then, for a permutation $j_1, j_2, \dots, j_d$ of $1, 2, \dots, d$, we can define a stratum as $Z_s = Z^{1j_1} \cup Z^{2j_2} \cup \dots \cup Z^{dj_d}$, where the *substratum* Z^{ij} denotes the set of training points that fall within block $\boldsymbol{V}^{ij}$. In general, the set S of possible strata contains $d!$ elements, one for each possible permutation of $1, 2, \dots, d$. Note that different strata may overlap when $d > 2$. Also note that there is no need to materialize these strata: They are constructed on-the-fly by processing only the respective blocks of $\boldsymbol{V}$.

The SSGD algorithm. Given the set of strata, we decompose the loss into a weighted sum of per-stratum losses: $L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{s=1}^{q} w_s L_s(\boldsymbol{W}, \boldsymbol{H})$, where w_s is a weight associated with stratum Z_s. This decomposition underlies our stratified SGD (SSGD) algorithm, a novel variant of SGD that chooses training points from only a single stratum instead of the entire matrix $\boldsymbol{V}$. To establish convergence, SSGD switches strata from time to time so that, in the long run, the "time" spent on stratum Z_s is proportional to the stratum weight w_s. Gemulla et al. [10] provide sufficient conditions for convergence, as well as a proof; the conditions in [10] hold for most matrix factorization problems.

We use per-stratum losses of form $L_s(\boldsymbol{W}, \boldsymbol{H}) = c_s \sum_{(i,j) \in Z_s} L_{ij}(\boldsymbol{W}, \boldsymbol{H})$ where c_s is a stratum-specific constant. When running SGD on a stratum, we use the gradient estimate $\hat{L}'_s(\boldsymbol{W}, \boldsymbol{H}) = N_s c_s L'_{ij}(\boldsymbol{W}, \boldsymbol{H})$ of $L'_s(\boldsymbol{W}, \boldsymbol{H})$ in each step, i.e., we scale up the local loss of an individual training point by the size $N_s = |Z_s|$ of the stratum. For example, from the $d!$ strata described previously, we can select d disjoint strata $Z_1, Z_2, \dots, Z_d$ such that they cover Z; e.g., strata Z_1, Z_2, and Z_3 in Fig. 1. Then any given loss function L of form (1) can be represented as a weighted sum over these strata by choosing w_s and c_s subject to $w_s c_s = 1$. Recall that the weight w_s can be interpreted as the fraction of time spent on each stratum in the long run. A natural choice is to set $w_s = N_s/N$, i.e., proportional to the stratum size. This particular choice leads to $c_s = N/N_s$ and we obtain the standard SGD gradient estimator $\hat{L}'_s(\boldsymbol{W}, \boldsymbol{H}) = N L'_{ij}(\boldsymbol{W}, \boldsymbol{H})$.

The DSGD algorithm. The individual steps in DSGD are grouped into *subepochs*, each of which amounts to (1) selecting one of the strata and (2) running SGD (in parallel) on the selected stratum. In more detail, DSGD makes use of a sequence $\{(\xi_k, T_k)\}$, where ξ_k denotes the *stratum selector* used in the kth subepoch, and T_k the number of steps to run on the selected stratum. The $\{(\xi_k, T_k)\}$ sequence is chosen such that the DSGD factorization algorithm is guaranteed to converge. For the particular choice of stratification described above, we set $T_k = |N_{\xi_k}|$; see [10] for details. Once a stratum ξ_k has been selected, we perform T_k SGD steps on Z_{ξ_k}; this is done in a parallel and distributed way. DSGD is shown as Algorithm 2, where we define an *epoch* as a sequence of d subepochs. An epoch roughly corresponds to processing the entire training set once. Since, by construction, parallel processing within the kth selected stratum leads to the same update terms as for the corresponding sequential SGD algorithm on Z_{ξ_k}, we have established the connection between DSGD and SSGD. Thus the convergence of DSGD is implied by the convergence of the underlying SSGD algorithm.

Implementation. When executing DSGD on d nodes in a shared-nothing environment such as MapReduce, we only distribute the input matrix once. Then the only data that are transmitted between nodes during subsequent processing are (small) blocks of factor matrices. In our implementation, node i stores blocks $\boldsymbol{W}^i, \boldsymbol{V}^{i1}, \boldsymbol{V}^{i2}, \ldots, \boldsymbol{V}^{id}$ for $1 \leq i \leq d$; thus only matrices $\boldsymbol{H}^1, \boldsymbol{H}^2, \ldots, \boldsymbol{H}^d$ need be transmitted. (If the $\boldsymbol{W}^i$ matrices are smaller, then we transmit these instead.)

Algorithm 2 DSGD for Matrix Factorization

```
Require: $\boldsymbol{V}$, $\boldsymbol{W}_0$, $\boldsymbol{H}_0$, cluster size $d$
  Block $\boldsymbol{V}$ / $\boldsymbol{W}$ / $\boldsymbol{H}$ into $d \times d$ / $d \times 1$ / $1 \times d$ blocks
  while not converged do {epoch}
    Pick step size $\epsilon$
    for $s = 1, \ldots, d$ do {subepoch}
      Pick $d$ blocks $\{\boldsymbol{V}^{1j_1}, \ldots, \boldsymbol{V}^{dj_d}\}$ to form a stratum
      for $b = 1, \ldots, d$ do {in parallel}
        Read blocks $\boldsymbol{V}^{bj_b}$, $\boldsymbol{W}^b$ and $\boldsymbol{H}^{j_b}$
        Run SGD on the training points in $\boldsymbol{V}^{bj_b}$ (step size = $\epsilon$)
        Write blocks $\boldsymbol{W}^b$ and $\boldsymbol{H}^{j_b}$
      end for
    end for
  end while
```

4 Experimental Results

In Table 1, we report results of the performance of DSGD on the KDDCup 2011 dataset, which consists of 252M music ratings (1M users, 624k items) along with the time of rating; matrix factorization is used to predict missing ratings. The winning team [13] integrated more than 80 different matrix factorizations on this dataset. Here we show results for a "biased matrix factorization", in which user (u_i) and movie (m_j) preferences are integrated into the model, with weighted L_2-regularization (see [3]; $\lambda_W = \lambda_H = 1$, $\lambda_u = 2$, $\lambda_m = 0.1$). We used rank $r = 60$ and run 200 epochs. Each node in our cluster is equipped with an Intel Xeon 2.40GHz processor and 48GB of RAM. We implemented DSGD in C++ and used MPI for communication. Table 1 shows the time per DSGD epoch (excluding loss computations) as well as the achieved loss after 60 and 30 minutes of wall-clock time. Note that plain SGD performs slightly better than DSGD in terms of achieved loss after a fixed number of epochs; this is a result of the differences in training point selection. In terms of wall-clock time, however, DSGD significantly outperforms SGD. See [10] for experiments on other datasets, other distributed architectures (including Hadoop), and a comparison to alternative algorithms.

Table 1: Experimental results on the KDDCup 2011 dataset

Nodes x Threads	Method	Avg. Time per Epoch	Training loss ($\times 10^{11}$) 200 ep.	60min	30min	Validation RMSE 200 ep.	60min	30min
1x1	SGD	335.9s	**1.417**	1.683	1.803	**21.89**	22.76	23.04
1x7	DSGD	47.2s	1.430	1.461	1.516	22.04	22.43	22.91
2x7	DSGD	22.9s	1.430	1.432	1.468	22.06	**22.07**	22.53
4x7	DSGD	**12.7s**	1.427	**1.428**	**1.437**	22.08	22.09	**22.20**

5 Conclusion

We have developed DSGD, a distributed version of the classic SGD algorithm for matrix factorization. DSGD can efficiently handle web-scale matrices; our experiments indicate fast convergence and good scalability.

References

[1] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.

[2] Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[4] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[5] Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *WWW*, pages 681–690, 2010.

[6] Benjamin Recht and Christopher Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Optimization Online*, 2011.

[7] Feng Niu, Benajami Recht, Christopher Ré, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[8] Sudipto Das, Yannis Sismanis, Kevin S. Beyer, Rainer Gemulla, Peter J. Haas, and John McPherson. Ricardo: Integrating R and Hadoop. In *SIGMOD*, pages 987–998, 2010.

[9] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the Netflix Prize. In *AAIM*, pages 337–348, 2008.

[10] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proc. of SIGKDD*, pages 69–77, 2011.

[11] Ajit P. Singh and Geoffrey J. Gordon. A unified view of matrix factorization models. In *ECML PKDD*, pages 358–373, 2008.

[12] Rainer Gemulla, Peter J. Haas, Erik Nijkamp, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. Technical Report RJ10481, IBM Almaden Research Center, San Jose, CA, 2011. Available at www.almaden.ibm.com/cs/people/peterh/dsgdTechRep.pdf.

[13] Po-Lung Chen, Chen-Tse Tsai, Yao-Nan Chen, Ku-Chun Chou, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Yu-Cheng Chou, Chung-Yi Li, Wei-Shih Lin, Shu-Hao Yu, Rong-Bing Chiu, Chieh-Yen Lin, Chien-Chih Wang, Po-Wei Wang, Wei-Lun Su, Chen-Hung Wu, Tsung-Ting Kuo, Todd G. McKenzie, Ya-Hsuan Chang, Chun-Sung Ferng, Chia-Mau Ni, Hsuan-Tien Lin, Chih-Jen Lin, and Shou-De Lin. A linear ensemble of individual and blended models for music rating prediction. In *KDDCup 2011 Workshop*, 2011.