2016 IEEE 55th Conference on Decision and Control (CDC)
ARIA Resort & Casino
December 12-14, 2016, Las Vegas, USA

Scaled stochastic gradient descent for low-rank matrix completion

Bamdev Mishra[†] and Rodolphe Sepulchre[§]

***Abstract*— The paper looks at a scaled variant of the stochastic gradient descent algorithm for the matrix completion problem. Specifically, we propose a novel matrix-scaling of the partial derivatives that acts as an efficient preconditioning for the standard stochastic gradient descent algorithm. This proposed matrix-scaling provides a trade-off between local and global second order information. It also resolves the issue of scale invariance that exists in matrix factorization models. The overall computational complexity is linear with the number of known entries, thereby extending to a large-scale setup. Numerical comparisons show that the proposed algorithm competes favorably with state-of-the-art algorithms on various different benchmarks.**

I. Introduction

The problem of low-rank matrix completion amounts to completing a matrix from a small number of entries by assuming a low-rank model for the matrix. This problem has been addressed both from theoretical [1]–[3] as well as from algorithmic viewpoints [2], [4]–[18]. A standard way of approaching the problem is by casting it as a *fixed-rank optimization problem* with the assumption that the optimal rank r is known a priori, i.e.,

$$\begin{aligned} \min_{\mathbf{X}\in\mathbb{R}^{n\times m}} \quad & \tfrac{1}{2}\|\mathcal{P}_\Omega(\mathbf{X}) - \mathcal{P}_\Omega(\mathbf{X}^\star)\|_F^2 \\ \text{subject to} \quad & \operatorname{rank}(\mathbf{X}) = r, \end{aligned} \tag{1}$$

where $\mathbf{X}^\star \in \mathbb{R}^{n\times m}$ is a matrix whose entries are known for indices if they belong to the subset $(i,j) \in \Omega$ and Ω is a subset of the complete set of indices $\{(i,j) : i \in \{1,...,n\}$ and $j \in \{1,...,m\}\}$. The operator $\mathcal{P}_\Omega(\mathbf{X}_{ij}) = \mathbf{X}_{ij}$ if $(i,j) \in \Omega$ and $\mathcal{P}_\Omega(\mathbf{X}_{ij}) = 0$ otherwise is called the orthogonal sampling operator and is a mathematically convenient way to represent the subset of entries. $\|\cdot\|_F$ is the *Frobenius* norm and $|\Omega|$ is the number of known entries. The low-rank assumption on (1) implies that $r \ll \min(n,m)$. The rank constraint correlates the known entries with the unknown ones. Recent contributions provide bounds on $|\Omega|$ (linear in n and m) for which exact reconstruction is possible in certain conditions from entries sampled randomly [1], [2].

Problem (1) and its (many) variants find applications in control systems and system identification [19], machine learning [12], and information theory [20], to name a just few. A popular way to tackle the rank constraint in (1) is by using a factorization model. The earlier works [21], [22] discuss factorization models and show how to perform first and second order optimization with them in presence of *scale invariance*, which arises due to non-uniqueness of factorization models. [10], [14] exploit the *Riemannian* structure of rank constraint and provide a spectrum of algorithms. *Preconditioning* with rank constraint in the context of matrix completion is recently explored in [6], [11], [22], [23]. Alternating minimization algorithms that exploit the *least-squares structure* of the cost function of (1) are proposed in [9], [18]. The least-squares structure is also exploited in [2], [6] to develop algorithms on the *Grassmann* manifold. The Matlab toolbox Manopt contains various other implementations [24].

While all the earlier mentioned algorithms are sequential algorithms, the works [8], [12], [13], [16], [17] focus on parallel and stochastic versions. An alternating least-squares approach is proposed in [16] to learn the *rows* of a factorization model, where each subproblem has a closed-form solution. The paper [17] also exploits the least-squares structure, but at the level of the entries of the rows of factorization models. [8] focuses on learning r rank-1 factorizations cyclically, where each subproblem is solved using the algorithm of [17]. The stochastic gradient descent algorithm (SGD) proposed in [12] updates the factorization model as and when the known entries are observed. The specific focus there is on parallelization of the SGD algorithm. A distributed version of SGD is proposed in [13]. Another approach that is suitable in an online setup is proposed in [15], where the data is assumed to be streaming from low-dimensional subspaces. The works [14], [21] exploit the Riemannian structure of rank constraint to propose online algorithms for low-rank matrix completion.

Our focus in this paper is on a *scaled* variant of SGD that accelerates the standard SGD algorithm and respects the scale invariance property of the factorization model. To achieve this, we propose a novel matrix-scaling of the partial derivatives in Section II that combines *global and local* second order information. The computational cost of the algorithm per pass through $|\Omega|$ known entries is $O(|\Omega|(r^3/b + b_{\mathbf{L}}r^2/b + b_{\mathbf{R}}r^2/b + r + \log b))$, where b is the batch size of the entries that we pick and $b_{\mathbf{L}}$ and $b_{\mathbf{R}}$ are the rows of $\mathbf{L}$ and $\mathbf{R}$ that are updated. The computational cost is comparable to those of [8], [12], [16] for $r \ll \min(n,m)$. Our numerical comparisons in Section III suggest that the proposed scaled SGD algorithm competes favorably with state-of-the-art on a number of different benchmarks, especially outperforming others on ill-conditioned and scarcely sampled data.

[†]Amazon Development Centre India, Bangalore 560055, India (Bamdevm@amazon.com). Initial work was done while this author was with the Department of Electrical Engineering and Computer Science, University of Liège, 4000 Liège, Belgium and was visiting the Department of Engineering (Control Group), University of Cambridge, Cambridge CB2 1PZ, UK.

[§]University of Cambridge, Department of Engineering, Trumpington Street, Cambridge CB2 1PZ, UK (R.Sepulchre@eng.cam.ac.uk).

978-1-5090-1836-9/16/$31.00 ©2016 IEEE

TABLE I

PROPOSED STOCHASTIC GRADIENT DESCENT ALGORITHM FOR (1).

1) Pick b known entries with their indices.
2) Set up the completion subproblem by finding the indices corresponding to the submatrices $\mathbf{L}_b$ and $\mathbf{R}_b$, which need to be modified. Consequently, find the subset Ω_b of indices out of the total $b_{\mathbf{L}}b_{\mathbf{R}}$ indices.
3) Compute the residual $\mathbf{S}_b = \mathcal{P}_{\Omega_b}(\mathbf{L}_b\mathbf{R}_b^T - \mathbf{X}_b^\star)$.
4) Given a stepsize t, update $\mathbf{L}_b$ and $\mathbf{R}_b$ as

$$\mathbf{L}_{b+} = \mathbf{L}_b - t\mathbf{S}_b\mathbf{R}_b(\tfrac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R}) + (1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))^{-1}$$
$$\mathbf{R}_{b+} = \mathbf{R}_b - t\mathbf{S}_b^T\mathbf{L}_b(\tfrac{b\mu}{\max(m,n)}(\mathbf{L}^T\mathbf{L}) + (1-\mu)(\mathbf{L}_b^T\mathbf{L}_b))^{-1}.$$

5) Update $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$.
6) Repeat.

II. SCALED STOCHASTIC GRADIENT DESCENT

Given a matrix $\mathbf{X}$ of size $n \times m$ and rank r, it admits the factorization

$$\mathbf{X} = \mathbf{L}\mathbf{R}^T, \tag{2}$$

where $\mathbf{L} \in \mathbb{R}_*^{n\times r}$ and $\mathbf{R} \in \mathbb{R}_*^{m\times r}$, where $\mathbb{R}_*^{n\times r}$ is the set of $n \times r$ full column-rank matrices [25]. Consequently, the problem (1) boils down to

$$\min_{\mathbf{L}\in\mathbb{R}_*^{n\times r},\mathbf{R}\in\mathbb{R}_*^{m\times r}} \quad \tfrac{1}{2}\|\mathcal{P}_\Omega(\mathbf{L}\mathbf{R}^T) - \mathcal{P}_\Omega(\mathbf{X}^\star)\|_F^2. \tag{3}$$

Consider a stochastic gradient setup for solving (3), where we pick b known entries at a time and then take a gradient descent step that updates the matrices $\mathbf{L}$ and $\mathbf{R}$. Due to the cost function structure, we end up updating only a *maximum of* b rows of $\mathbf{L}$ and $\mathbf{R}$ at a time. Let $b_{\mathbf{L}}$ rows of $\mathbf{L}$ and $b_{\mathbf{R}}$ rows of $\mathbf{R}$ be updated when b known entries are picked, where $b_{\mathbf{L}} \leq b$ and $b_{\mathbf{R}} \leq b$. Let $\mathbf{L}_b$ be the corresponding submatrix of $\mathbf{L}$ with the $b_{\mathbf{L}}$ rows, i.e., its size is $b_{\mathbf{L}} \times r$. Similarly, let $\mathbf{R}_b$ be the submatrix of $\mathbf{R}$ with the $b_{\mathbf{R}}$ rows and size $b_{\mathbf{R}} \times r$. An interpretation is that, each time we pick b known entries, we have a subproblem of completing a matrix $\mathbf{X}_b^\star$ of size $b_{\mathbf{L}} \times b_{\mathbf{R}}$ with b known entries at indices Ω_b, which needs to be approximated by $\mathbf{L}_b\mathbf{R}_b^T$. If $\mathbf{S}_b$ is the residual matrix of this subproblem, then the partial derivatives at $(\mathbf{L}_b, \mathbf{R}_b)$ are $(\mathbf{S}_b\mathbf{R}_b, \mathbf{S}_b^T\mathbf{L}_b)$, where $\mathbf{S}_b = \mathcal{P}_{\Omega_b}(\mathbf{L}_b\mathbf{R}_b^T - \mathbf{X}_b^\star)$ is of size $b_{\mathbf{L}} \times b_{\mathbf{R}}$.

The proposed stochastic gradient descent updates are

$$\mathbf{L}_{b+} = \mathbf{L}_b - t\mathbf{S}_b\mathbf{R}_b(\tfrac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R}) + (1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))^{-1}$$
$$\mathbf{R}_{b+} = \mathbf{R}_b - t\mathbf{S}_b^T\mathbf{L}_b(\tfrac{b\mu}{\max(m,n)}(\mathbf{L}^T\mathbf{L}) + (1-\mu)(\mathbf{L}_b^T\mathbf{L}_b))^{-1}, \tag{4}$$

where t is the step size, $b/\max(m,n)$ is a normalization constant, and μ is a nonnegative scalar in $[0,1]$ that weighs $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}_b^T\mathbf{L}_b$ differently. The term $b/\max(m,n)$ ensures that the Frobenius norm of $b(\mathbf{L}^T\mathbf{L})/\max(m,n)$ and $(\mathbf{L}_b^T\mathbf{L}_b)$ are of the same order. Similarly, the terms $\mathbf{R}^T\mathbf{R}$ and $\mathbf{R}_b^T\mathbf{R}_b$. It should be stated that $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$ can be either be computed after every update or after a certain number of updates (e.g., one pass through data). The stepsize t can be modified, e.g., using the bold driver protocol [13], [26] or the exponential decay protocol [12, Section 4.1]. The choice of μ depends on the problem.

The proposed algorithm is shown in Table I.

It should be noted that the terms $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$ capture a part of second order information of $\|\mathbf{L}\mathbf{R}^T - \mathbf{X}^\star\|_F^2/2$, which is related to the cost function of (3) and gives a simpler way to understand the behavior of the cost function. In fact, $\|\mathbf{L}\mathbf{R}^T - \mathbf{X}^\star\|_F^2/2$ is obtained by assuming that Ω is the full set of indices in (3). This relies on *strict convexity* of $\|\mathbf{L}\mathbf{R}^T - \mathbf{X}^\star\|_F^2/2$ with respect to the factors $\mathbf{L}$ and $\mathbf{R}$ individually. The block diagonal approximation of the Hessian of $\|\mathbf{L}\mathbf{R}^T - \mathbf{X}^\star\|_F^2/2$ with respect to $(\mathbf{L}, \mathbf{R})$ is $((\mathbf{R}^T\mathbf{R}) \otimes \mathbf{I}_n, (\mathbf{L}^T\mathbf{L}) \otimes \mathbf{I}_m)$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix and $\otimes$ is the Kronecker product of matrices. For example, this Hessian approximation is used in the works of [6], [11], [22], where the authors accelerate the convergence of algorithms (e.g., steepest descent, conjugate gradients, and trust-regions) by scaling the partial derivatives with respect to $\mathbf{L}$ and $\mathbf{R}$ with $(\mathbf{R}^T\mathbf{R})^{-1}$ and $(\mathbf{L}^T\mathbf{L})^{-1}$, respectively.

In our case, an additional motivation for a trade-off with μ in (4), i.e., between the terms $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$ on one hand and $\mathbf{L}_b^T\mathbf{L}_b$ and $\mathbf{R}_b^T\mathbf{R}_b$ on the other, comes from the following intuition. $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$ can be interpreted as capturing the global second order information as they contain knowledge of all the rows of $\mathbf{L}$ and $\mathbf{R}$ [6], [11], [22]. On the other hand in a stochastic setup we modify only $\mathbf{L}_b$ rows of $\mathbf{L}$ and $\mathbf{R}_b$ rows of $\mathbf{R}$. This leads to the argument that $\mathbf{L}_b^T\mathbf{L}_b$ in the term $\mathbf{L}^T\mathbf{L}$ should be given a higher weight. Similarly, the part $\mathbf{R}_b^T\mathbf{R}_b$ in $\mathbf{R}^T\mathbf{R}$ is given a *differentiated* weight. Overall the terms $(\frac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R})+(1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))^{-1}$ and $(\frac{b\mu}{\max(m,n)}(\mathbf{L}^T\mathbf{L})+(1-\mu)(\mathbf{L}_b^T\mathbf{L}_b))^{-1}$ which are multiplied to the partial derivatives $\mathbf{S}_b\mathbf{R}_b$ and $\mathbf{S}_b^T\mathbf{L}_b$, respectively, act as an *efficient preconditioner* for the standard stochastic gradient descent updates proposed in [12], [13].

Conceptually, our approach can also be connected to the recent work [27], which also combines local and global information, but in the context of *first order* information, i.e., gradient information. The resulting stochastic variance reduction algorithms have shown superior performance.

A. Computation cost. In Table I, Step 2 costs $O(b\log b)$ to identify the rows of $\mathbf{L}$ and $\mathbf{R}$ that need to be modified and costs $O(b)$ to find the subset Ω_b. It involves *sorting* (and hence, the cost is $O(b\log b)$) the row indices corresponding to the b known entries to find the unique rows of $\mathbf{L}$ and $\mathbf{R}$ that are required to be updated. Computation of the residual $\mathbf{S}_b$ costs $O(b)$ in Step 3. The updates in Step 4 costs $O(r^3+br)$. Step 5 costs $O(b_{\mathbf{L}}r^2 + b_{\mathbf{R}}r^2)$. Consequently, each time we pick any b entries, our proposed gradient descent step costs $O(b_{\mathbf{L}}r^2+b_{\mathbf{R}}r^2+br+b+b\log b+r^3)$, where $b_{\mathbf{L}} \leq b$ and $b_{\mathbf{R}} \leq b$. Equivalently, our algorithm costs $O(|\Omega|(r^3/b + b_{\mathbf{L}}r^2/b + b_{\mathbf{R}}r^2/b + r + \log b))$ after we have seen $|\Omega|$ entries.

Depending on b, the computational cost varies from $O(|\Omega|r^3)$ to $O(|\Omega|r^2)$. For $b = 1$, the inverse computation of $(\frac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R})+(1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))$ costs $O(r^2)$ as it requires only a rank-1 modification per update. Consequently, our algorithm costs $O(|\Omega|r^2)$ for $b = 1$. For $b \in (1, r)$,

the computation cost is upper bounded by $O(|\Omega|r^3)$ for a straightforward implementation of matrix inversion in (4). For $b \geq r$, the computational cost is $O(|\Omega|r^2)$. In particular if $b = |\Omega|$, then $b_{\mathbf{L}} = n$, $b_{\mathbf{R}} = m$, and the computational cost is $O(|\Omega|r + nr^2 + mr^2)$, which is same as the computational cost (per iteration) of most algorithms in the batch setup, e.g., the ones proposed in [2], [6], [9]–[11], [18], [22].

B. Scale invariance. The proposed updates (4) also resolve the issue of scale invariance arising from non-uniqueness of matrix factorization (2) as $\mathbf{X}$ remains unchanged under the action [21], [25]

$$(\mathbf{L}, \mathbf{R}) \mapsto (\mathbf{L}\mathbf{M}^{-1}, \mathbf{R}\mathbf{M}^T), \tag{5}$$

for all non-singular matrices $\mathbf{M} \in \mathrm{GL}(r)$, where $\mathrm{GL}(r)$ is the set of all non-singular matrices of size $r \times r$. Equivalently, $\mathbf{X} = \mathbf{L}\mathbf{R}^T = \mathbf{L}\mathbf{M}^{-1}(\mathbf{R}\mathbf{M}^T)^T$. The issue of scale invariance refers to the behavior of algorithms which behave *equivalently* when initialized, say, either with $(\mathbf{L}_0, \mathbf{R}_0)$ or with $(\mathbf{L}_0\mathbf{M}^{-1}, \mathbf{R}_0\mathbf{M}^T)$ for all non-singular matrices $\mathbf{M} \in \mathrm{GL}(r)$. For example, if $(\mathbf{L}_+, \mathbf{R}_+)$ is an update obtained from $(\mathbf{L}_0, \mathbf{R}_0)$, then a *scale-invariant algorithm* produces the update $(\mathbf{L}_+\mathbf{M}^{-1}, \mathbf{R}_+\mathbf{M}^T)$ starting from $(\mathbf{L}_0\mathbf{M}^{-1}, \mathbf{R}_0\mathbf{M}^T)$. It is the case for the algorithm proposed in Table I. It is straightforward to show that under the mapping (5), the proposed updates (4) lead to the transformation $(\mathbf{L}_{b+}, \mathbf{R}_{b+}) \mapsto (\mathbf{L}_{b+}\mathbf{M}^{-1}, \mathbf{R}_{b+}\mathbf{M}^T)$. On the other hand, the standard stochastic gradient descent updates [7], [13],

$$\begin{aligned} \mathbf{L}_{b+} &= \mathbf{L}_b - t\mathbf{S}_b\mathbf{R}_b \\ \mathbf{R}_{b+} &= \mathbf{R}_b - t\mathbf{S}_b^T\mathbf{L}_b, \end{aligned} \tag{6}$$

are not scale invariant. Here $\mathbf{L}_b$ and $\mathbf{R}_b$ are the submatrices of $\mathbf{L}$ and $\mathbf{R}$, respectively that are updated for the batch size b, t is the stepsize, and $\mathbf{S}_b$ is the residual matrix at $(\mathbf{L}_b, \mathbf{R}_b)$.

The benefits of scale invariant algorithms are discussed in [21], [23], [28]. We also show an example in Section III-B.

C. Choice of μ. A key observation is that the updates (4) with $\mu = 1$ are equivalent to the stochastic version of the updates proposed in [9], [18], [22]. For $\mu < 1$, we take the additional local information into account. On the other hand, $\mu = 0$ gives full weighting to the "local" second order information and should be used when $\mathbf{L}_b^T\mathbf{L}_b$ and $\mathbf{R}_b^T\mathbf{R}_b$ are positive definite. This holds true for $b_{\mathbf{L}} \geq r$ and $b_{\mathbf{R}} \geq r$, i.e., for a large enough batch size. For a smaller batch size, i.e., $b_{\mathbf{L}} < r$ or $b_{\mathbf{R}} < r$, a non-zero μ should be used. In problem instances where a large number of entries are already known, i.e., $|\Omega|$ is large, the influence of $\mu < 1$ is minimal. However, for *ill-conditioned data*, making use of local information is more critical, and a smaller value of μ is more appropriate, e.g., $\mu = 0.5$. These trade-offs are shown in Section III-C.

D. Choice of batch size b. For $b = |\Omega|$, the algorithm in Table I behaves like a batch gradient descent algorithm and with same computational cost as discussed Section II-A. The choice of $b = 1$ is more appropriate for a fully online system. Other choices depend on the problem size and set up. Section III-D shows the robust behavior of the algorithm with different choices of b.

E. Convergence. The convergence analysis of the proposed algorithm in Table I follows the discussion in [12], [13], [29] except for the (positive definite) matrix-scaling of the partial derivatives $(\mathbf{S}_b\mathbf{R}_b, \mathbf{S}_b^T\mathbf{L}_b)$ at $(\mathbf{L}_b, \mathbf{R}_b)$ by multiplying with $((\frac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R}) + (1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))^{-1}$, $(\frac{b\mu}{\max(m,n)}(\mathbf{L}^T\mathbf{L}) + (1-\mu)(\mathbf{L}_b^T\mathbf{L}_b))^{-1})$.

A different interpretation is that we endow the search space $\mathbb{R}^{n\times r} \times \mathbb{R}^{m\times r}$ with the *adaptive inner product*

$$\begin{aligned} &\langle \xi_x, \eta_x \rangle_{\text{adaptive}} = \\ &\mathrm{Trace}((\tfrac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R}) + (1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))\xi_{\mathbf{L}}^T\eta_{\mathbf{L}}) \\ &+\mathrm{Trace}((\tfrac{b\mu}{\max(m,n)}(\mathbf{L}^T\mathbf{L}) + (1-\mu)(\mathbf{L}_b^T\mathbf{L}_b))\xi_{\mathbf{R}}^T\eta_{\mathbf{R}}), \end{aligned} \tag{7}$$

which depends on $(\mathbf{L}, \mathbf{R})$ and $(\mathbf{L}_b, \mathbf{R}_b)$. Here x has the matrix representation $(\mathbf{L}, \mathbf{R})$ and ξ_x and η_x vectors in $\mathbb{R}^{n\times r} \times \mathbb{R}^{m\times r}$ with matrix representations $(\xi_{\mathbf{L}}, \xi_{\mathbf{R}})$ and $(\eta_{\mathbf{L}}, \eta_{\mathbf{R}})$, respectively. $\langle \xi_x, \eta_x \rangle_{\text{adaptive}}$ is the inner product between ξ_x and η_x and $\mathrm{Trace}(\cdot)$ is the matrix trace operator. Finally, computing the steepest descent directions with the inner product (7) leads to the updates (4). The proposed inner product (7) has the interpretation of a *Riemannian metric* in the framework of Riemannian optimization [28]. Consequently, the analysis of a Riemannian stochastic gradient descent algorithm presented in [30] is applicable to our case, e.g., under a decaying stepsize condition and in a compact region near the minimum.

While asymptotic convergence guarantees for the proposed algorithm are available, the rate of convergence analysis of the proposed algorithm in Table I is a challenging task and remains an open problem. We point to the recent work [31] that sheds more light on this for a similar problem.

III. Numerical Experiments

In this section, we compare the following algorithms.

(1) **Scaled-SGD:** the proposed scaled stochastic gradient descent algorithm. The computation cost is $O(|\Omega|(r^3/b + b_{\mathbf{L}}r^2/b + b_{\mathbf{R}}r^2/b + r + \log b))$ after we have seen all the $|\Omega|$ entries with b entries at a time. $\mathbf{L}^T\mathbf{L}$ and $\mathbf{R}^T\mathbf{R}$ are updated after every update. (2) **SGD:** the standard SGD implementation, where the updates result by imposing the Euclidean metric [7], [13]. The computational cost is $O(|\Omega|(r + \log b))$ when we sweep through $|\Omega|$ entries with b entries at a time. It should be stated that the cost is independent of b as the effect of $\log b$ is minimal. (3) **Grouse:** the algorithm proposed in [15]. Instead of learning $\mathbf{L}$ and $\mathbf{R}$ simultaneously, it learns, e.g., the rank-r left subspace spanned by $\mathbf{L}$ first by traversing through the columns of the incomplete matrix $\mathbf{X}^\star$. The $\mathbf{R}$ factor is computed by solving a least-squares problem in closed form once we have learned the subspace spanned by $\mathbf{L}$. The learning of $\mathbf{L}$ boils down to an optimization problem on the Grassmann manifold. Consequently, Grouse is a stochastic gradient descent algorithm on the Grassmann manifold. Its computational cost is $O(mnr)$ after sweeping through m columns, i.e., one pass through $\mathbf{X}^\star$. (4) **Loreta:** the algorithm proposed in [14], but modified to handle data as in [15], i.e., we sweep through the incomplete matrix $\mathbf{X}^\star$ column by column. The computational cost is $O(mnr)$

after one pass through $\mathbf{X}^\star$. (5) **ALS:** the standard alternating least-squares algorithm, where we update the low-rank factors $\mathbf{L}$ and $\mathbf{R}$ row-by-row by solving the least-squares subproblems in closed form [16]. The computational cost is $O(|\Omega|r^2 + (n+m)r^3)$ per update all the rows of $\mathbf{L}$ and $\mathbf{R}$. (6) **CCD++:** the algorithm proposed in [8], which learns r rank-1 factors sequentially. For learning a rank-1 factor, it uses T inner iterations of the algorithm proposed in [17]. Its computational cost is $O(|\Omega|r\mathrm{T})$ after one update of $\mathbf{L}$ and $\mathbf{R}$. As suggested in [8], T is set to 5.

The choice of the above algorithms is motivated by the fact that these algorithms can be readily adapted to an online setup. As the mentioned algorithms are well suited for different scenarios and have implementations in different programming languages, we use only their Matlab implementations (which we implement for all except Loreta and Grouse) and compare them on the behavior of the cost function against *iterations*. An iteration for Scaled-SGD, SGD, Grouse, and Loreta corresponds to one pass through $|\Omega|$ entries. For ALS and CCD++, an iteration corresponds to one update of the low-rank factors $\mathbf{L}$ and $\mathbf{R}$. During each iteration, the stepsize t is fixed for Scaled-SGD, SGD, and Loreta. The stepsize is then updated according to the bold driver heuristic as suggested in [13]. In the bold driver protocol [26], updating of the stepsize depends on the cost function. In case the cost increases after an iteration, the stepsize is reduced by 50%, else the stepsize is increased by 10%. The initial stepsize is computed using the approach by linearizing the cost function as done in [22]. For Grouse, we use the stepsize update proposed in [15]. ALS and CCD++ do not require any stepsize tuning. Additionally, during each iteration, the known entries are randomly chosen for Scaled-SGD and SGD with *uniform* probability and without replacement. Equivalently, each known entry is seen only once per iteration. Similarly, during an iteration, the columns (rows) are randomly and uniformly chosen for Grouse and Loreta (ALS) without replacement.

A. Experimentation setup and stopping criteria. All simulations are performed in Matlab and on a 2.7 GHz Intel Core i5 machine with 8 GB of RAM. For each example, an $n \times m$ random matrix of rank r is generated as in [4]. Two matrices $\mathbf{A} \in \mathbb{R}^{n\times r}$ and $\mathbf{B} \in \mathbb{R}^{m\times r}$ are generated according to a Gaussian distribution with zero mean and unit standard deviation. The matrix product $\mathbf{AB}^T$ gives a random rank-r matrix. A fraction of the entries are randomly removed with uniform probability. The dimensions of $n \times m$ matrices of rank r is $(n+m-r)r$. The over-sampling (OS) ratio determines the number of entries that are known. An OS of 6 implies that $6(n+m-r)r$ number of randomly and uniformly selected entries are known a priori out of the total nm entries. No regularization is used. The algorithms are stopped when either the mean square error $\|\mathcal{P}_\Omega(\mathbf{X}) - \mathcal{P}_\Omega(\mathbf{X}^\star)\|_F^2/|\Omega|$ is less than 10^{-8} or the relative residual $\|\mathcal{P}_\Omega(\mathbf{X}) - \mathcal{P}_\Omega(\mathbf{X}^\star)\|_F/\|\mathcal{P}_\Omega(\mathbf{X}^\star)\|_F$ is less than 10^{-4} or the number of iterations exceeds 100. The Matlab codes are available at http://bamdevmishra.com/codes/scaledSGD/.

B. Effect of scale invariance. The difference of the standard stochastic updates (6) with the proposed updates (4) is in the $r \times r$ matrices, e.g., $(\frac{b\mu}{\max(m,n)}(\mathbf{R}^T\mathbf{R}) + (1-\mu)(\mathbf{R}_b^T\mathbf{R}_b))$, that are inversely applied to (6). However, those extra (but minimal) computations make the proposed updates *invariant* to the transformation (5), which is not the case with the Euclidean updates (6). To illustrate this effect of scale invariance, we consider a problem instance with $n = m = 100$, $r = 5$, and $\mathrm{OS} = 8$. Both Scaled-SGD and SGD are run with batch size $b = 10$ and $\mathbf{L}$ and $\mathbf{R}$ are randomly initialized with balanced factors (solid line in Figure 1(a)) such that $\|\mathbf{L}_{\text{init}}\|_F \approx \|\mathbf{R}_{\text{init}}\|_F$. Additionally, we set μ to 0.5. The performance of both the algorithms is similar. However, the performance of SGD suffers drastically for unbalanced factors (dashed line), i.e., when $\|\mathbf{L}_{\text{init}}\|_F \approx 4\|\mathbf{R}_{\text{init}}\|_F$ as shown in Figure 1(a).

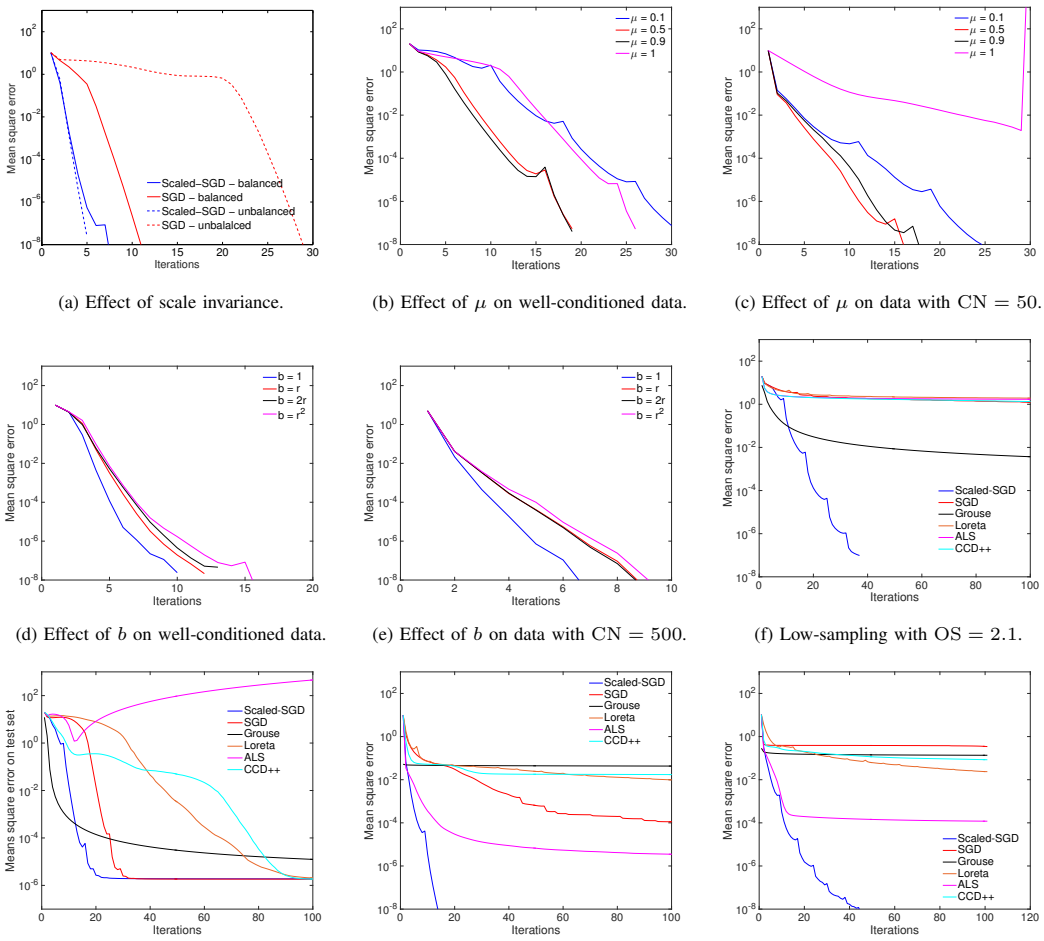
C. Effect of μ. We consider problem instances of size 5000×5000 and rank 10. In order to understand the influence of μ on Scaled-SGD, we consider two scenarios. The first scenario consists of an instance with over-sampling ratio 3. The second scenario considers an over-sampling ratio of 3 and ill-conditioned data with condition numbers (CN) 50, which is obtained by imposing an exponential decay of singular values (discussed in Section III-G). Figures 1(b) and 1(c) show the behavior of Scaled-SGD with four different values of μ. Scaled-SGD is run with batch size $b = 10$. Figures 1(b) shows that there exists values of μ, which show better performance. Figure 1(c) shows that relying solely on global information, i.e., $\mu = 1$, need not be better. In particular, Scaled-SGD with $\mu = 1$ diverges in 1(c). $\mu = 0.5$, on the other hand, shows a good performance in many instances.

D. Effect of batch size b. We consider problem instances of size 5000×5000 of rank 5. In order to understand the influence of b on Scaled-SGD, we consider two scenarios with $b = \{1, r, 2r, r^2\}$ with over-sampling ratio of 5. The first scenario consists of a well-conditioned instance. The second scenario consists of ill-conditioned data with condition number CN equal to 500. μ is set to 0.5 in Scaled-SGD. Figures 1(d) and 1(e) show the robust behavior of Scaled-SGD with different batch sizes b.

E. Low-sampling instances. We consider problem instances of size 5000×5000 and rank 10. Different over-sampling ratios of 4, 3, 2.5, and 2.1 are considered. Scaled-SGD and SGD are run with batch size $b = 10$. Additionally, we set μ to 0.5 in Scaled-SGD. While most algorithms perform well for larger OS values, Scaled-SGD particularly outperforms others for smaller OS values as shown in Figure 1(f).

F. Noisy instances. We consider the problem instance in Section III-E with $\mathrm{OS} = 3$. Additionally, noise is added to the known entries. As proposed in [15], noise for each entry is sampled from the Gaussian distribution with mean zero and standard deviations 10^{-4}. Figure 1(g) shows the performance of the algorithms on the test set that is held out, which is different from the training set Ω.

G. Ill-conditioned instances. We consider matrices of



(a) Effect of scale invariance. (b) Effect of μ on well-conditioned data. (c) Effect of μ on data with CN $= 50$.

(d) Effect of b on well-conditioned data. (e) Effect of b on data with CN $= 500$. (f) Low-sampling with OS $= 2.1$.

(g) Noisy instance. (h) Ill-conditioned data with CN $= 100$. (i) Rectangular instance.

Fig. 1. Scaled-SGD resolves scale-invariance in matrix factorization. Furthermore, it is robust to various choices of μ and b. In particular, a good trade-off between global and local information, i.e., for $\mu \in (0, 1)$, leads to improved performance in both well-conditioned and ill-conditioned data. Scaled-SGD outperforms others in lower sampling instances, i.e., with small OS values and ill-conditioned instances.

size 5000×5000 and rank 10 and impose an exponential decay of singular values. The ratio of the largest to the lowest singular value is known as the condition number (CN) of the matrix. For example, at rank 10 the singular values with condition number 100 is obtained using the Matlab function `logspace(-2,0,10)`. The over-sampling ratio for these instances is 3. μ is set to 0.5. Figure 1(h) shows the performance of various algorithms, where our proposed approach outperforms others for CN 100. Scaled-SGD shows a robust performance on ill-conditioned instances.

H. Rectangular instances. We consider rectangular matrices of size 1000×8000 of rank 10 and over sampling ratio 3. μ is set to 0.5. Most comparisons suggest that ALS and Grouse perform very well on those instances. However, even for slightly ill-conditioned data, the performance of ALS and Grouse degrade as shown in Figure 1(i). Scaled-SGD remains unaffected.

I. Jester dataset. We consider the Jester dataset 1 [32] consisting of ratings of 100 jokes by 24983 users. Each rating is between -10 and 10. Following the protocol in [5], we select $n = \{2000, 5000, 24983\}$ users randomly. We randomly extract two ratings per user as test data. The algorithms are run for ranks $\{5, 7\}$ with random initialization and for 100 iterations. Predictions are computed at the end of 100th iteration. The entire process is repeated ten times. No regularization is used for Scaled-SGD, SGD, Grouse, and Loreta. The performance of ALS and CCD++, however, *critically* depends on regularization for which we set the regularization parameter to 10 for $n = \{2000, 5000\}$ and 100 for $n = 24983$ after cross-validation. The batch size b is set to r for Scaled-SGD and SGD. μ is set to 0.5 in Scaled-SGD. Figure 2 shows the performance plots. Table II shows the *final* normalized mean absolute errors (NMAE) obtained by different algorithms on the test dataset averaged over *ten* runs. NMAE is defined as the mean absolute error (MAE) divided by spread of the ratings, i.e., the difference between the minimum and maximum ratings, i.e., NMAE is MAE/20. The standard deviation of the scores in Table II is $2 \cdot 10^{-3}$. Except Grouse, all other algorithms give similar NMAE scores on the test set. Scaled-SGD consistently outperforms SGD as shown in Figure 2.

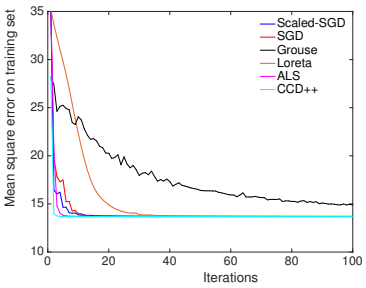


Fig. 2. Performance of the algorithms on the Jester dataset for $n = 2000$ and $r = 7$.

TABLE II

FINAL MEAN NMAES OBTAINED ON THE TEST SET OF THE JESTER DATASET.

Algorithm	$n = 2000$		$n = 5000$		$n = 24983$	
	$r = 5$	$r = 7$	$r = 5$	$r = 7$	$r = 5$	$r = 7$
Scaled-SGD	0.158	0.159	0.160	0.158	0.159	0.157
SGD	0.158	0.159	0.160	0.158	0.159	0.157
Grouse	0.165	0.165	0.179	0.177	0.168	0.166
Loreta	0.159	0.160	0.160	0.158	0.159	0.158
ALS	0.158	0.159	0.160	0.158	0.159	0.157
CCD++	0.158	0.159	0.160	0.157	0.159	0.157

IV. CONCLUSION

We have proposed a scaled variant of stochastic gradient descent algorithm for the low-rank matrix completion problem. It is based on a novel matrix-scaling of the partial derivatives with terms that combine both local and global second order information. This scaling is computationally cheap to implement and the proposed algorithm is potentially scalable to larger datasets. Initial results show a robust performance of the proposed algorithm on various benchmarks. At the conceptual level, this paper shows the complementary role of local and global second order information in a stochastic gradient setting.

REFERENCES

[1] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[2] R. H. Keshavan, A. Montanari, and S. Oh, "Matrix completion from a few entries," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2980–2998, 2010.

[3] K. Wei, J.-F. Cai, T. F. Chan, and S. Leung, "Guarantees of Riemannian optimization for low rank matrix recovery," arXiv preprint arXiv:1511.01562, Tech. Rep., 2015.

[4] J. F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[5] R. H. Keshavan, A. Montanari, and S. Oh, "Low-rank matrix completion with noisy observations: a quantitative comparison," in *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2009, pp. 1216–1222.

[6] N. Boumal and P.-A. Absil, "Low-rank matrix completion via preconditioned optimization on the Grassmann manifold," *Linear Algebra and its Applications*, vol. 475, pp. 200–239, 2015.

[7] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM Review*, vol. 53, no. 3, pp. 471–501, 2010.

[8] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon, "Parallel matrix factorization for recommender systems," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 793–819, 2014.

[9] J. Tanner and K. Wei, "Low rank matrix completion by alternating steepest descent methods," *Applied and Computational Harmonic Analysis*, 2015, doi: http://dx.doi.org/10.1016/j.acha.2015.08.003.

[10] B. Vandereycken, "Low-rank matrix completion by Riemannian optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1214–1236, 2013.

[11] T. T. Ngo and Y. Saad, "Scaled gradients on Grassmann manifolds for matrix completion," in *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012, pp. 1421–1429.

[12] B. Recht and C. Ré, "Parallel stochastic gradient algorithms for large-scale matrix completion," *Mathematical Programming Computation*, vol. 5, no. 2, pp. 201–226, 2013.

[13] C. Teflioudi, F. Makari, and R. Gemulla, "Distributed matrix completion," in *International Conference on Data Mining (ICDM)*, 2012, pp. 655–664.

[14] U. Shalit, D. Weinshall, and G. Chechik, "Online learning in the manifold of low-rank matrices," in *Neural Information Processing Systems conference (NIPS)*, 2010, pp. 2128–2136.

[15] L. Balzano, R. Nowak, and B. Recht, "Online identification and tracking of subspaces from highly incomplete information," in *The 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, June 2010, pp. 704–711.

[16] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the Netflix prize," in *International Conference on Algorithmic Aspects in Information and Management (AAIM)*, 2008, pp. 337–348.

[17] I. Pilászy, D. Zibriczky, and D. Tikk, "Fast als-based matrix factorization for explicit and implicit feedback datasets," in *ACM conference on Recommender systems (RecSys)*, 2010, pp. 71–78.

[18] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm," *Mathematical Programming Computation*, vol. 4, no. 4, pp. 333–361, 2012.

[19] I. Markovsky and K. Usevich, "Structured low-rank approximation with missing data," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 814–830, 2013.

[20] Y. Shi, Y. Zhang, Letaif, and K. B. and, "Low-rank matrix completion for topological interference management by Riemannian pursuit," *IEEE Transactions on Wireless Communications*, vol. 15, no. 7, pp. 4703–4717, 2016.

[21] G. Meyer, S. Bonnabel, and R. Sepulchre, "Linear regression under fixed-rank constraints: a Riemannian approach," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 545–552.

[22] B. Mishra, K. Adithya Apuroop, and R. Sepulchre, "A Riemannian geometry for low-rank matrix completion," arXiv preprint arXiv:1211.1550, Tech. Rep., 2012.

[23] B. Mishra and R. Sepulchre, "Riemannian preconditioning," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 635–660, 2016.

[24] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, "Manopt: a Matlab toolbox for optimization on manifolds," *Journal of Machine Learning Research*, vol. 15, no. Apr, pp. 1455–1459, 2014.

[25] R. Piziak and P. L. Odell, "Full rank factorization of matrices," *Mathematics Magazine*, vol. 72, no. 3, pp. 193–201, June 1999.

[26] R. Battiti, "Accelerated backpropagation learning: two optimization methods," *Complex Systems*, vol. 3, no. 4, pp. 331–342, 1989.

[27] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 315–323.

[28] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008.

[29] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning in Neural Networks*, D. Saad, Ed. Cambridge, UK: Cambridge University Press, 1998, pp. 9–42.

[30] S. Bonnabel, "Stochastic gradient descent on Riemannian manifolds," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2217–2229, 2013.

[31] C. D. Sa, K. Olukotun, and C. Ré, "Global convergence of stochastic gradient descent for some non-convex matrix problems," in *International Conference on Machine learning (ICML)*, 2015, pp. 2332–2341.

[32] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: a constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.