Distributed SGD for Matrix Factorization

Michal Minařík, Yasin Esfandiari, Yassir Janah
Optimization for Machine Learning

Abstract—**Low-rank matrix factorization is an effective tool for discovering and analyzing the interactions between two given entities. It has been successfully applied in recommendation systems (users and items) or news personalization (users and articles) [?]. In production systems, matrices with millions of rows and columns can be involved [?]. Therefore, efficient algorithms that approximate the solution in a reasonable time need to be implemented, often leveraging distributed computing [?], [?]. In this work, one such algorithm using distributed stochastic gradient descent [?] is implemented and analyzed. The analysis covers convergence time and reconstruction quality of the recommendation system task for different numbers of CPU nodes on both synthetic data and a public dataset (MovieLens [?]). Code can be found at https://github.com/yasin-esfandiari/Distributed-SGD-for-Matrix-Factorization**

I. Introduction

Matrix factorization (MF) consists of finding a low-rank approximation $\hat{\mathbf{Y}}$ of a given data matrix $\mathbf{Y} \in \mathbb{R}^{N \times M}$.
In other words, we are interested in the following approximation: $\mathbf{Y} \approx \mathbf{U}\mathbf{V}^T = \hat{\mathbf{Y}}$ where $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{M \times K}$ are latent factor matrices and $K \ll \min(N, M)$ is the number of latent factors.

In the context of recommender systems, MF assumes that each user $n \in \{1, \ldots, N\}$ is represented by a preference vector $\mathbf{U}_{n,*} \in \mathbb{R}^K$, and that each item $m \in \{1, \ldots, M\}$ is represented by feature vector $\mathbf{V}_{m,*} \in \mathbb{R}^K$ [?]. These two vectors belong to the same latent feature space $\mathbb{R}^K$. Moreover, the rating or the interaction strength between a user n and an item m is measured by the scalar product: $\hat{\mathbf{Y}}_{n,m} = \langle \mathbf{U}_{n,*}, \mathbf{V}_{m,*} \rangle = \sum_{k=1}^{K} u_{nk} v_{mk}$, where u_{nk} and v_{mk} are the coefficients of the $\mathbf{U}$ and $\mathbf{V}$ matrices. Once the approximation is computed, the matrix $\hat{\mathbf{Y}}$ is used to make predictions and recommendations to users.

Learning the latent models of users $\mathbf{U}$ and items $\mathbf{V}$ can be formulated as the following optimization problem:

$$(\mathbf{U}, \mathbf{V})^* = \underset{\mathbf{U}, \mathbf{V}}{\operatorname{argmin}}\, \mathcal{L}\left(\mathbf{Y}, \mathbf{U}\mathbf{V}^\top\right) + \lambda \Omega\left(\mathbf{U}, \mathbf{V}\right) \qquad (1)$$

where $\mathcal{L}\left(\mathbf{Y}, \mathbf{U}\mathbf{V}^\top\right)$ is a distance measure between the data matrix and its approximation, and $\lambda \Omega\left(\mathbf{U}, \mathbf{V}\right)$ is a penalty term that is used to avoid model overfitting.

Gradient Descent based methods are the most common approaches for solving this optimization problem [?]. Due to the huge volume of information in recommender systems, computing and updating these matrices may become very expensive and time-consuming for a single machine. To make MF scalable, it is necessary to parallelize these optimization algorithms among several machines [?], [?].
This work focuses on the Distributed Stochastic Gradient Descent (DSGD) [?], as opposed to the Stochastic Gradient Descent baseline method for matrix factorization[1].
The contributions in these aspects are: (1) an implementation of this technique in Python and (2) a performance comparison of our implementation against vanilla SGD as a baseline. We evaluate the performance of these methods on synthetic data and on the public dataset of MovieLens [?]. The evaluation and comparisons are made in terms of accuracy, convergence time, and adaptability to the number of machines in the cluster.

II. Methods

In an optimization problem, when the objective is differentiable, the simplest and most efficient methods in terms of computation time are certainly the gradient descent methods.

A. Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) performs an update after each randomly selected training point (n, m) from the training data.
A simple formulation of the factor update rules in the case of an objective function as in (1), with $\mathcal{L}\left(\mathbf{Y}, \mathbf{U}\mathbf{V}^\top\right)$ being the distance induced from the Frobenius norm and a regularization term $\lambda\Omega(U, V) = \lambda(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2)$, is the following:

$$\mathbf{U}_{n,*}^{\text{new}} \leftarrow \mathbf{U}_{n,*}^{\text{old}} - 2\rho\left(-\left(\mathbf{Y}_{n,m} - \left\langle \mathbf{U}_{n,*}^{\text{old}}, \mathbf{V}_{m,*}^{\text{old}} \right\rangle\right) \mathbf{V}_{m,*}^{\text{old}} + \lambda \mathbf{U}_{n,*}^{\text{old}}\right)$$
$$\mathbf{V}_{m,*}^{\text{new}} \leftarrow \mathbf{V}_{m,*}^{\text{old}} - 2\rho\left(-\left(\mathbf{Y}_{n,m} - \left\langle \mathbf{U}_{n,*}^{\text{old}}, \mathbf{V}_{m,*}^{\text{old}} \right\rangle\right) \mathbf{U}_{n,*}^{\text{old}} + \lambda \mathbf{V}_{m,*}^{\text{old}}\right)$$

where ρ denotes the learning rate.

Once all ratings $\mathbf{Y}_{n,m}$ are visited, one iteration of the SGD is completed. For very large datasets, this may result in a slow convergence time. To overcome this, one may consider the batch SGD, which iterates over a batch of training points $\mathcal{B}_i = \{(n_1, m_1) \ldots (n_s, m_s)\}$ before updating the factors. Although this technique is straightforward, its parallelization is challenging [?], [?]. If the updates for two training points sharing the same user or item are computed in parallel, this interdependency creates a problem while writing the factor update.

B. Distributed Stochastic Gradient Descent

To avoid the above-described confliction, a parallelization of the SGD called Distributed SGD (DSGD) was introduced by R. Gemulla et al. [?]. A first algorithm towards the full distribution of computations was given as the Stratified SGD (SSGD).

[1]S. Funk, Netflix Update: Try This at Home

1) Stratified SGD: Consider two *interchangeable* training points (n, m) and (n', m') where $n \neq n'$ and $m \neq m'$. Then, their corresponding SGD updates are independent and hence can be computed in parallel. This definition can be generalized to *block interchangeability* defined as follows: given two blocks of the rating matrix $\mathbf{Y}_{I \times J}$ and $\mathbf{Y}_{I' \times J'}$ where I, I' (and J, J') are subsets of row indices (column indices) of the rating matrix $\mathbf{Y}$, respectively. Then these blocks are interchangeable if and only if $I \cap I' = J \cap J' = \emptyset$.
Now, we can decompose the rating matrix into sets of interchangeable blocks (*strata*) and perform SGD updates on each stratum. Inside each stratum, there are no dependency problems, and all processors can simultaneously compute updates to their assigned factor blocks. When all strata are visited, one iteration of the SSGD is completed.

2) Distributed SGD: The DSGD specializes the SSGD by choosing a set of strata $\{S_1, ..., S_q\}$ that covers all indices $\{1, \ldots, N\} \times \{1, \ldots, M\}$ of the training set and each individual stratum is decomposed into d interchangeable blocks: $S_j = \{I_1 \times J_1, \ldots, I_d \times J_d\}$. The factor matrices $\mathbf{U}$ and $\mathbf{V}$ are blocked accordingly, and each block is assigned to a single machine in the cluster. In a subepoch, the machines will update their assigned blocks in parallel, and after each stratum computation the updated factor blocks will be *synchronized* to update the factors $\mathbf{U}$ and $\mathbf{V}$.

3) Convergence: The DSGD is guaranteed to converge if the stratification, the sequence of strata, the learning rate, and the loss are chosen such that the underlying SSGD satisfies the convergence conditions detailed in [**?**].

III. Experiments

A. Setting

Experiments were run to compare the efficiency of the algorithm based on the number of workers' and the SGD step size. For a fixed step size $\rho \in \{0.01, 0.001, 0.0001\}$, the algorithm was run for each number of workers $D \in \{1, 2, 4, 8\}$. Since the initial matrices are initialized randomly, each setup was run five times to gain some statistical data. Therefore, 60 runs were concluded in total, with 12 distinct parameter settings.

Each run was capped by the number of iterations, with early stopping implemented — once the loss stopped decreasing, the algorithm was also stopped. The value of the regularization parameter λ has been set to $\lambda = 0.02$.

The method was first tested on a synthetic matrix of size 16x16 with rank 2. Then a part of a larger public dataset of [**?**] was used, containing a matrix with dimensions 943x1682 with 80000 non-zero elements.

Because the algorithm for $D = 1$ is the same as a classical SGD, the results for $D = 1$ will serve as a baseline.

B. Results

For each of the 12 scenarios, multiple runs were concluded. The results are split by step size value and averaged for each of the workers number D.

1) Synthetic data: The loss function of DSGD on the synthetic data is shown on Fig. 1 ($\rho = 0.01$), Fig. 2 ($\rho = 0.001$) and Fig. 3 ($\rho = 0.0001$). The maximal number of iterations was 5000, but for better clarity, only 2000 values are shown.

The results for $\rho = 0.01$ are shown in Fig. 1 and Tab. I. Using smaller values resulted in divergence of the method used. For number of workers 1 and 2, the algorithm converges at around 300 iterations. When a larger number of workers is used, more iterations are needed to reach the result.

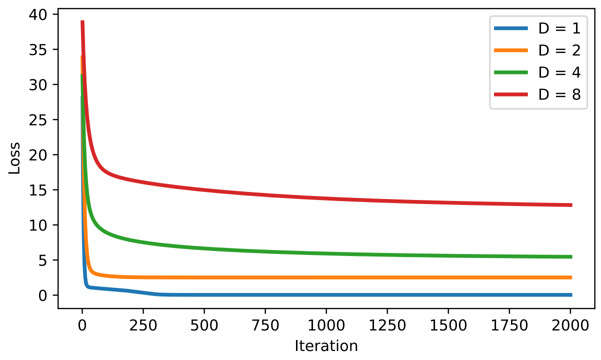

Fig. 1. Evolution of the loss function on the synthetic data for step size $\rho = 0.01$. Results are averaged over all 5 runs.

Workers	Loss	Iterations	Time per iteration [ms]
1	0.038	4716.0	19.38
2	1.469	3827.8	2.11
4	5.331	4204.0	2.74
8	9.962	2433.6	6.20

TABLE I
DSGD results on the synthetic data for step size $\rho = 0.01$. Results are averaged over all 5 runs.

The results for $\rho = 0.001$ are shown in Fig. 2 and Tab. II. Using a smaller value of ρ increases the number of iterations needed until convergence without significant decrease of the loss function.

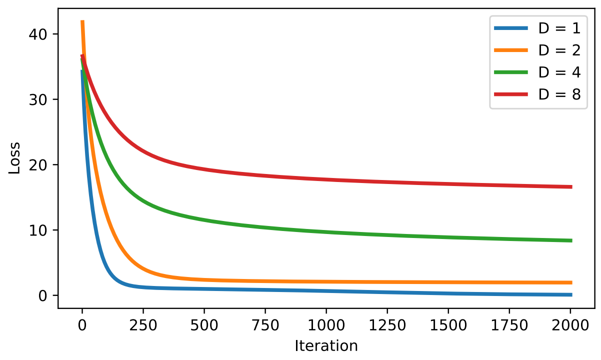

Fig. 2. Evolution of the loss function on the synthetic data for step size $\rho = 0.001$. Results are averaged over all 5 runs.

Decreasing the step size further to $\rho = 0.0001$ increases the number of iterations even more. The results for are shown in Fig. 3 and Tab. III.

Workers	Loss	Iterations	Time per iteration [ms]
1	0.039	4645.8	17.05
2	1.760	3202.0	2.13
4	6.257	4471.6	2.70
8	12.545	5000.0	6.20

TABLE II
DSGD results on the synthetic data for step size $\rho = 0.001$. Results are averaged over all 5 runs.

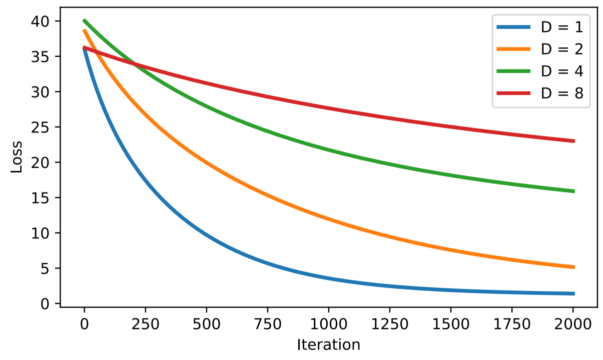

Fig. 3. Evolution of the loss function on the synthetic data for step size $\rho = 0.0001$. Results are averaged over all 5 runs..

Workers	Loss	Iterations	Time per iteration [ms]
1	0.923	5000.0	19.06
2	2.515	5000.0	2.13
4	9.080	5000.0	2.73
8	21.270	5000.0	6.19

TABLE III
DSGD results on the synthetic data for step size $\rho = 0.0001$. Results are averaged over all 5 runs.

2) MovieLens dataset: The loss function of DSGD on a part of the MovieLens dataset is shown on Fig. 4 ($\rho = 0.01$), Fig. 5 ($\rho = 0.001$) and Fig. 6 ($\rho = 0.0001$). Due to the fact that computers used had only 4 CPU cores available, we have decided to omit the test with $D = 8$

The maximal number of iterations was 1000, but for better clarity, only 200 values are shown. The results on the MovieLens dataset show a behavior similar to the synthetic data results.

Workers	Loss	Iterations	Time per iteration [ms]
1	59230.387	66.2	2927.84
2	81391.252	13.2	2769.85
4	117186.240	13.2	2033.50

TABLE IV
DSGD results on the MovieLens dataset for step size $\rho = 0.01$. Results are averaged over all 5 runs.

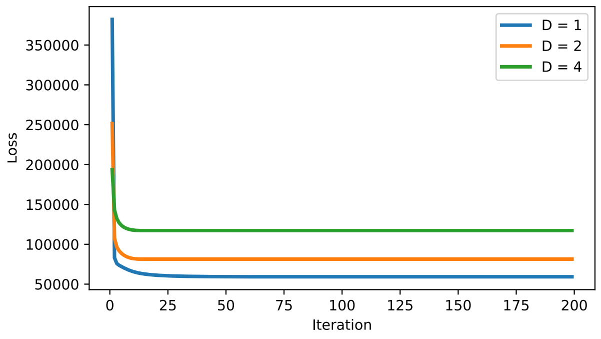

Fig. 4. Evolution of the loss function on the MovieLens dataset for step size $\rho = 0.01$. Results are averaged over all 5 runs.

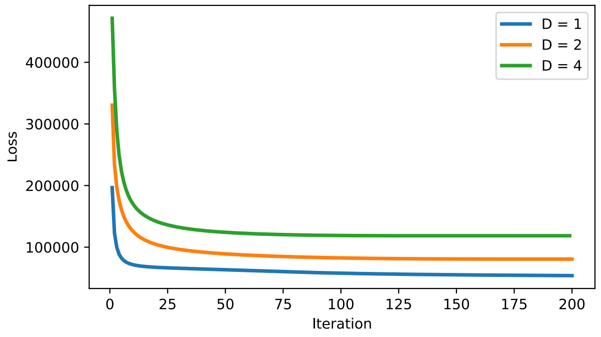

Fig. 5. Evolution of the loss function on the MovieLens dataset for step size $\rho = 0.001$. Results are averaged over all 5 runs.

Workers	Loss	Iterations	Time per iteration [ms]
1	50887.173	1000.0	2635.41
2	80601.377	176.6	2018.50
4	118526.314	127.4	1644.07

TABLE V
DSGD results on the MovieLens dataset for step size $\rho = 0.001$. Results are averaged over all 5 runs.

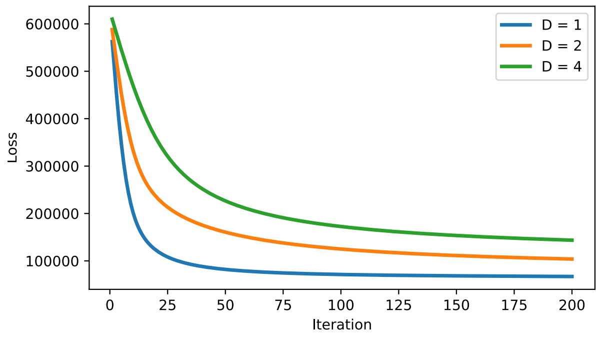

Fig. 6. Evolution of the loss function on the MovieLens dataset for step size $\rho = 0.0001$. Results are averaged over all 5 runs.

IV. Discussion and Conclusion

We have implemented and tested distributed stochastic gradient descent method for low-rank matrix factorization. The results show that the loss function converges to a higher value when the number of workers is increased (Fig. 1, 2 and 3).

Workers	Loss	Iterations	Time per iteration [ms]
1	58067.001	1000.0	2670.14
2	82554.619	1000.0	1935.14
4	119339.594	1000.0	1229.17

TABLE VI
DSGD RESULTS ON THE MOVIELENS DATASET FOR STEP SIZE $\rho = 0.0001$. RESULTS ARE AVERAGED OVER ALL 5 RUNS.

This agrees with the results found by [**?**], but the differences between the loss values are higher in our case. There exist more possibilities which could lead to such discrepancy.

In the original paper [**?**], the used loss function was different (no regularization was used), and the step size was adjusted dynamically based on the change of the loss value (contrary to our implementation, which uses constant step size). Finally, there exists a possibility of error in the implementation.

The advantage of the tested method can be found in the time needed for one iteration. Increasing the number of workers significantly decreases the time needed for one iteration, as can be seen in Tab. I, II and III. In our case, this holds true for smaller number of workers. Raising the number of workers to 8, the time for one iteration is increased again. The reason for this increase is the fact that the benchmark was run on a system where only 4 CPU cores were available. Therefore, all 8 tasks can not be run truly parallel.

In our case, setting the number of workers to 2 significantly speeds up the computation while keeping the value of the loss function close to baseline SGD. Choosing a larger value of D does not lead to further improvements.