Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Polynomial Optimization Methods for Matrix Factorization

Po-Wei Wang
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
poweiw@cs.cmu.edu

Chun-Liang Li
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
chunlial@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
zkolter@cs.cmu.edu

Abstract

Matrix factorization is a core technique in many machine learning problems, yet also presents a nonconvex and often difficult-to-optimize problem. In this paper we present an approach based upon polynomial optimization techniques that both improves the convergence time of matrix factorization algorithms and helps them escape from local optima. Our method is based on the realization that given a joint search direction in a matrix factorization task, we can solve the "subspace search" problem (the task of jointly finding the steps to take in each direction) by solving a bivariate quartic polynomial optimization problem. We derive two methods for solving this problem based upon sum of squares moment relaxations and the Durand-Kerner method, then apply these techniques on matrix factorization to derive a direct coordinate descent approach and a method for speeding up existing approaches. On three benchmark datasets we show the method substantially improves convergence speed over state-of-the-art approaches, while also attaining lower objective value.

1 Introduction

Matrix factorization has become a foundational operation in many machine learning tasks. Although there are several variants of matrix factorization, one of the most common is the collaborate filtering setting, where we wish to find a low-rank reconstruction of some matrix $S \in \mathbb{R}^{M\times N}$, where we only observe the entries S_{ij} at some relatively small number of points $(i,j) \in \mathcal{S}$. The goal is then to find some $A \in \mathbb{R}^{M\times K}$ and $B \in \mathbb{R}^{N\times K}$ such that

$$a_i^T b_j \approx S_{ij}, \quad \forall (i,j) \in \mathcal{S} \tag{1}$$

where a_i denotes the ith row of A and b_j denotes the jth row of B. Although many methods to perform this factorization are possible, a natural approach is to solve the optimization problem

$$\underset{A,B}{\text{minimize}} \;\; L(A,B), \tag{2}$$

where

$$L(A,B) \equiv \sum_{(i,j)\in\mathcal{S}} (a_i^T b_j - S_{ij})^2 + \lambda\left(\|A\|_F^2 + \|B\|_F^2\right),$$

in which A and B are the optimization variables and λ is a regularization parameter. This approach and extensions have been used in a number of collaborative filtering and recommender system applications (Bell and Koren 2007; Dror et al. 2012) and researchers have developed numerous optimization approaches to solving these problems, discussed in more detail below.

Despite it ubiquity, (2) is a non-convex optimization problem as it involves quartic terms in the A and B variables. A number of approaches can still be used to optimize the objective — two popular and straightforward approaches, for example, are to use alternating minimization, since the objective is convex in either A or B separately, or simple gradient descent in the non-convex objective — but the possibility of local optima is real and a non-trivial reality of matrix factorization methods. A low-dimension example of (2) is given in Appendix B, in which both methods stuck if not properly initialized.

In this paper we propose an approach that both *helps to mitigate the possibility of local optima and speeds up matrix factorization approaches above the current state of the art*. The basic idea is straightforward: suppose we want to optimize a *single pair* of coefficient a_{ik} and b_{kj}, or more generally suppose we have some joint search direction $(\Delta A, \Delta B)$ over A and B (the case of a single set of coordinates simply being when ΔA and ΔB have only one non-zero component). Then we can consider the problem of optimizing the objective *just* over this subspace

$$\underset{\alpha,\beta}{\text{minimize}}\, L(A + \alpha\Delta A, B + \beta\Delta B) \tag{3}$$

where $\alpha, \beta \in \mathbb{R}$ are our optimization variables. This problem becomes a quartic polynomial in the two variables, α and β, which can be solved exactly using sum of squares methods (Parrilo 2003; Blekherman, Parrilo, and Thomas 2013); alternatively, for this particular case we show the problem can be reduced to finding the root of a quintic polynomial in a single variable, and solved exactly using the Durand-Kerner method (also called the method of Weierstrass) (Durand 1959; Petkovi et al. 1995) (we show, this approach is often several orders of magnitude faster than the SOS-based methods). Using this "exact subspace search" technique, we both derive a pure coordinate descent approach to matrix factorization and modify existing optimization approaches to incorporate this subspace search when adjusting the parameters. For the case of the pure coordinate descent approach, we derive an efficient caching mechanism that lets us efficiently update individual coordinates with minimal passes over the training data. Our results show that our method improves substantially over the existing state of the art in terms of optimization speed, and also allows the methods to more robustly escape from local optima.

2 Background and related work

Several solution methods have been proposed to solve the optimization problem (2) and similar problems. We discuss the popular approaches briefly, and focus on two recent methods that are considered as the state of the art by some metrics: coordinate descent (Pilászy, Zibriczky, and Tikk 2010) and stochastic gradient descent (Koren, Bell, and Volinsky 2009).

As mentioned previously, problem (2) is nonconvex. However, when either A or B is fixed, the problem becomes an convex quadratic with an analytical solution. The alternating least square method (ALS)(Zhou et al. 2008) exploits this observation and iteratively solves the two least-squares problems. The ALS is strictly decreasing but has an $O(|\mathcal{S}|K^2 + (M+N)K^3)$ complexity per iteration, and thus is not efficient on a large dataset with non-trivial K.

Another increasingly popular method is stochastic gradient descent (SGD)(Koren, Bell, and Volinsky 2009). For each update, the SGD randomly selects a pair $(i,j) \in \mathcal{S}$ and updates the corresponding variables via

$$a_i := a_i - \eta(\lambda a_i + R_{ij} b_j), \quad b_j := b_j - \eta(\lambda b_j + R_{ij} a_i), \quad (4)$$

where $R_{ij} \equiv a_i^T b_j - S_{ij}$. The SGD enjoys a $O(|\mathcal{S}|K + (M+N)K)$ complexity per data pass, and several parallel algorithms have been developed (Gemulla et al. 2011; Recht and Ré 2013; Zhuang et al. 2013). However, it is not guaranteed to be strictly decreasing and requires fine-tuning the additional step-size parameter η in practice.

Updating one variable at a time results in the coordinate descent algorithm (Pilászy, Zibriczky, and Tikk 2010). With proper caching and update order, this method enjoys $O(|\mathcal{S}|K + (M+N)K)$ complexity and is guaranteed to be strictly decreasing. Surprisingly, the coordinate descent approach for matrix factorization is fully parallelizable. To see this, let us fix all the variables except for the variable a_{ik}, the analytical solution for this variable is

$$a_{ik} := \mathrm{CCD}(i, b_{:k}, R_{:/k}) = -\frac{\sum_{j|(i,j)\in\mathcal{S}} R_{ij/k} b_{jk}}{\lambda + \sum_{j|(i,j)\in\mathcal{S}} b_{jk}^2}, \quad (5)$$

where $R_{ij/k} = R_{ij} - a_{ik}b_{jk}$. Note that the update formula is independent of a_{ik} for all i, because $R_{ij/k}$ has no dependence on a_{ik} terms. Thus, the update on a_{ik} for all i can be performed concurrently. This extension is called CCD++ (Yu et al. 2012).

Background on polynomial optimization Although the nonconvexity of (2) can make it difficult to solve, not all nonconvex problems are intractable. Specifically, optimizing a polynomial function, including non-convex and multi-modal polynomials, is a well-studied numerical problem, and there exist multiple solution techniques that work well in practice. For example, one well-known approach to polynomial optimization is via the sum of squares (SOS)(Parrilo 2003; Blekherman, Parrilo, and Thomas 2013) and moment relaxation (Lasserre 2001; Laurent 2009) approach, which approximates the solution of a polynomial optimization problem by a semidefinite program. For bivariate quartic polynomials (the precise type that will appear in our subspace search method), the SOS relaxation is exact and we can recover the exact solution to the polynomial optimization problem. In addition, there are several approaches for root finding of single variable polynomials, and we will exploit one of these methods, the Durand-Kerner approach (Durand 1959), to develop an even faster method for solving the bivariate quartic optimization problem.

3 Polynomial optimization for matrix factorization

First, we formally define the subspace search problem (3). Consider an arbitrary pair of search directions $U \equiv \Delta A$ and $V \equiv \Delta B$. In the subspace search, we want to solve the optimization problem

$$\underset{\alpha,\beta}{\text{minimize}} \; L(A + \alpha U, B + \beta V). \quad (6)$$

Expanding the definition of L in (2) and then shifting and scaling (see Appendix A for details), solving this problem is equivalent to

$$\underset{\alpha,\beta}{\text{minimize}} \; F(\alpha, \beta), \quad (7)$$

where $F(\alpha,\beta)$ is defined as

$$\frac{1}{2}C_{22}\alpha^2\beta^2 + C_{11}\alpha\beta + \frac{1}{2}C_{20}\alpha^2 + C_{10}\alpha + \frac{1}{2}C_{02}\beta^2 + C_{01}\beta,$$

in which C_{20} and C_{02} are strictly positive. In particular, as we show in Appendix A, the coefficients on any possible $\alpha^2\beta$ and $\alpha\beta^2$ terms can be removed. When $C_{22} = 0$, the problem can be solved by a linear equation by taking derivatives on α and β. Otherwise, with proper scaling, we can assume that $C_{22} = 1$ without loss of generality.

3.1 Solving polynomial optimization

The subspace search problem (7) is in general not convex and admits local minima. However, it can be solved efficiently via polynomial optimization techniques. We here demonstrate two methods that solves the bivariate quartic problem globally.

Moment relaxation SDP formulation The moment relaxation approach to solving the polynomial optimization problem, which is the dual of the SOS relaxation, is based upon forming a positive semidefinite *moment matrix* that consists of various polynomial degree terms of α and β. Specifically, define $\phi = (1 \quad \alpha \quad \beta \quad \alpha\beta)^\top$ and let

$$Y = E[\phi\phi^T] = \begin{bmatrix} 1 & y_{01} & y_{10} & y_{11} \\ y_{01} & y_{02} & y_{11} & y_{12} \\ y_{10} & y_{11} & y_{20} & y_{21} \\ y_{11} & y_{12} & y_{21} & y_{22} \end{bmatrix}, \quad (8)$$

where the expectation is over a properly constructed measure (Parrilo 2003). Then we can solve the semidefinite program

$$\begin{aligned} \underset{Y \succeq 0}{\text{minimize}} \;\; & \frac{1}{2}y_{22} + C_{11}y_{11} + \frac{1}{2}C_{20}y_{20} \\ & + C_{10}y_{10} + \frac{1}{2}C_{02}y_{02} + C_{01}y_{01}, \end{aligned} \tag{9}$$

with the guarantee that we are able to reconstruct α^* and β^* of (7) from the optimal solution Y^* of (9). This result is important theoretically, because it provides a basis for the global solvability of the such polynomial (bivariate quartic) optimization problems.

Durand-Kerner method In practice, even for a relatively small polynomial problem, solving the SDP as (9) is slow. Therefore, instead of solving (7) directly through solving a SDP, in this case we can transform it to a univariate polynomial and solve this directly using a root-finding method. Specifically, considering the derivatives of the objective

$$\nabla_\alpha F(\alpha,\beta) = \beta(\alpha\beta + C_{11}) + C_{20}\alpha + C_{10} \tag{10}$$

$$\nabla_\beta F(\alpha,\beta) = \alpha(\alpha\beta + C_{11}) + C_{02}\beta + C_{01} \tag{11}$$

We can perform a symmetric linear transform and write these equations as

$$\begin{aligned} & \sqrt{C_{02}} \cdot \nabla_\alpha F(\alpha,\beta) + \sqrt{C_{20}} \cdot \nabla_\beta F(\alpha,\beta) \\ & = D(t+D)x + D\Delta_+ = 0, \end{aligned} \tag{12}$$

$$\begin{aligned} & \sqrt{C_{02}} \cdot \nabla_\alpha F(\alpha,\beta) - \sqrt{C_{20}} \cdot \nabla_\beta F(\alpha,\beta) \\ & = D(t-D)y + D\Delta_- = 0, \end{aligned} \tag{13}$$

where we introduce the new variables t, x, and y

$$\begin{aligned} & t = \alpha\beta + C_{11}, \\ & x = \sqrt{C_{20}}\alpha + \sqrt{C_{02}}\beta, \quad y = \sqrt{C_{20}}\alpha - \sqrt{C_{02}}\beta, \end{aligned} \tag{14}$$

and constants D, Δ_+, and Δ_-

$$\begin{aligned} & D = \sqrt{C_{20}C_{02}}, \\ & \Delta_+ = \frac{C_{10}}{\sqrt{C_{20}}} + \frac{C_{01}}{\sqrt{C_{02}}}, \quad \Delta_- = \frac{C_{10}}{\sqrt{C_{20}}} - \frac{C_{01}}{\sqrt{C_{02}}}. \end{aligned} \tag{15}$$

Note that C_{02} and C_{20} are positive.

Starting now with the equality $t - C_{11} = \frac{x^2-y^2}{4D}$, and multiplying both sides by $(t+D)^2(t-D)^2$ we get

$$\begin{aligned} & (t-C_{11})(t+D)^2(t-D)^2 \\ & = \frac{1}{4D}((t+D)^2x^2)(t-D)^2 - \frac{1}{4D}((t-D)^2y^2)(t+D)^2. \end{aligned} \tag{16}$$

Applying (12), (13) to the above equality and rearrange the terms, we can finally write our gradient conditions in terms of a *single* quintic polynomial equation

$$\begin{aligned} f(t) \equiv & (t-C_{11})(t+D)^2(t-D)^2 \\ & - (\frac{1}{4D}\Delta_+^2)(t-D)^2 + (\frac{1}{4D}\Delta_-^2)(t+D)^2 = 0. \end{aligned} \tag{17}$$

At this point, we can apply the Durand-Kerner method to solve (17), which takes the form

1. Initialize t_i, $i = 1, \ldots, 5$ on a complex disk that will contain all the roots.
2. While $|f(t_i)| \geq \epsilon$ for any i: repeat

$$t_i \leftarrow t_i - \frac{f(t_i)}{\prod_{j \neq i}(t_i - t_j)}, \; i = 1, \ldots, 5. \tag{18}$$

The Durand-Kerner method generates all 5 (complex) roots of (17), including all the local minima. We choose the real root that minimize the subspace search. To guarantee that our initial disk contains all the roots requires potentially restarting the Durand-Kerner method multiple times, but in practice we are able to choose a disk size based upon the highest magnitude coefficient C and have yet to observe divergence (and importantly, lack of divergence for the Durand-Kerner method signifies that we have obtained the exact solution).

3.2 Polynomial coordinate descent

Using this exact bivariate subspace search procedure, we develop two approaches to solving the matrix factorization problem. The first is the direct approach, where we extend a coordinate descent approach to simultaneously optimize a pair of coefficients a_{ik} and b_{jk}, where (i,j) will cycle through the entries of S. In this setting, our optimization over α and β becomes

$$\begin{aligned} F(\alpha,\beta) = & \frac{1}{2}\alpha^2\beta^2 + R_{ij/k}\alpha\beta \\ & + \frac{1}{2}C_{20}^{i/j}\alpha^2 + C_{10}^{i/j}\alpha + \frac{1}{2}C_{02}^{j/i}\beta^2 + C_{01}^{j/i}\beta, \end{aligned} \tag{19}$$

where

$$\begin{aligned} & C_{20}^{i/j} = C_{20}^i - b_{jk}^2, \quad C_{10}^{i/j} = C_{10}^i - R_{ij/k}b_{jk}, \\ & C_{02}^{j/i} = C_{02}^j - a_{ik}^2, \quad C_{01}^{j/i} = C_{01}^i - R_{ij/k}a_{ik}. \end{aligned} \tag{20}$$

$$C_{20}^i = \lambda + \sum_{q|i,q\in\mathcal{S}} b_{qk}^2, \quad C_{10}^i = \sum_{q|i,q\in\mathcal{S}} R_{iq/k}b_{qk}, \tag{21}$$

$$C_{02}^j = \lambda + \sum_{p|p,j\in\mathcal{S}} a_{pk}^2, \quad C_{01}^j = \sum_{p|p,j\in\mathcal{S}} R_{pj/k}a_{pk} \tag{22}$$

Although the notation is somewhat cumbersome, these terms just signify the total C coefficients with the coefficients involves the k terms subtracted out. The advantage to maintaining the coefficients in this form, is that after updating a single a_{ik}, b_{jk} pair, we can update all the coefficients in $O(1)$ time, by simply adding back the new values. This bookkeeping allows us to perform a pass over *all* $(i,j) \in \mathcal{S}$ (for a fixed k) in $O(|S|)$ time. The full algorithm with the relevant updating and downdating, is given in Algorithm 1.

3.3 Polynomial subspace search

Although the direct coordinate descent method is efficient asymptotically, it still requires substantial computational work, as it needs to perform polynomial root finding once for each $(i,j) \in \mathcal{S}$, and thus the constant term on the $O(|\mathcal{S}|)$ complexity is quite large. An alternative which often works

```
for $k = 1, \ldots, K$ do
    $R_{ij/k} = R_{ij} - a_{ik}b_{jk}, \ \forall (i,j) \in \mathcal{S}$;
    Construct $C_{01}^{j}, C_{02}^{j}$ using (22) for all $j$;
    for $i = 1, \ldots, M$ do
        Update $C_{01}^{j/i}, C_{02}^{j/i}$ from $C_{01}^{j}, C_{02}^{j}$ via (20) for
          all $j$;
        Construct $C_{10}^{i}, C_{20}^{i}$ by (21);
        for $j \mid (i,j) \in \mathcal{S}$ do
            Update $C_{10}^{i/j}, C_{20}^{i/j}$ from $C_{10}^{i}, C_{20}^{i}$ via (20);
            Solve $a_{ik}, b_{jk} = \arg\min_{\alpha,\beta} F(\alpha,\beta)$ with
              coefficient $C_{20}^{i/j}, C_{10}^{i/j}, C_{02}^{j/i}, C_{01}^{j/i}$ in (19);
            Update $C_{10}^{i}, C_{20}^{i}$ from $C_{10}^{i/j}, C_{20}^{i/j}$ via (20);
        end
        Update $C_{01}^{j}, C_{02}^{j}$ from $C_{01}^{j/i}, C_{02}^{j/i}$ via (20) for
          all $j$;
    end
    $R_{ij} = R_{ij/k} + a_{ik}b_{jk}, \ \forall (i,j) \in \mathcal{S}$;
end
```

Algorithm 1: PolyMF-CD

```
for $k = 1, \ldots, K$ do
    Remove coordinate $k$ by
      $R_{ij/k} = R_{ij} - a_{ik}b_{jk}, \ \forall (i,j) \in \mathcal{S}$;
    $old\_a_{:k} = a_{:k}, \ old\_b_{:k} = b_{:k}$;
    max_func_decr $= 0$;
    for $iter$=$1, \ldots, T$ do
        foreach $j = 1, \ldots, N$ do
          $b_{jk} :=$ CCD$(j, a_{:k}, R_{:/k})$ by (5) ;
        foreach $i = 1, \ldots, M$ do
          $a_{ik} :=$ CCD$(i, b_{:k}, R_{:/k})$ by (5) ;
        func_decr :=
          function decrease in this inner iteration;
        max_func_decr :=
          max(max_func_decr, func_decr);
        if func_decr $< 10^{-8}$max_func_decr then break
          ;
    end
    $U = a_{:k} - old\_a_{:k}, \ V = b_{:k} - old\_b_{:k}$;
    Solve subspace search
      minimize$_{\alpha,\beta}$ $L(A + \alpha U, B + \beta V)$;
    $a_{:k} := a_{:k} + \alpha U, \ b_{:k} := b_{:k} + \beta V$;
    Update and restore coordinate $k$ by
      $R_{ij} = R_{ij/k} + a_{ik}b_{jk}, \ \forall (i,j) \in \mathcal{S}$;
end
```

Algorithm 2: PolyMF-SS

better in practice is to use an alternative matrix factorization algorithm to obtain search directions for an entire column/row of A and B respectively, and use the subspace search to find the adjustment of A and B in this direction. The subspace search could be applied to virtually any matrix factorization approach, including gradient descent, ALS, SGD, and CCD++. In this paper, we adopt CCD++ for the subspace it outperforms other algorithms in our benchmark test. Concretely, for a given k we use CCD++ to compute the coordinate descent direction of an entire row/column of A and B respectively, and use our polynomial optimization to find the exact minimum over this joint search direction. The algorithm is described in Algorithm 2.

Finally, we note that a few technical issues come up when applying this subspace search to CCD++. We keep tracking of the function decrease in inner iterations, and skip the inner iteration early when the function decrease is too small, as proposed in (Yu et al. 2014). Also, we skip the line search procedure in the first iteration because the first few steps of CCD++ is usually very large and avoiding subspace search provides better initialization. The whole algorithm is also fully parallelizable like CCD++, and we study the speedup of parallelism in Section 3.4.

3.4 Convergence

In this section we prove that the PolyMF-CD and PolyMF-SS (with CCD++ search direction) algorithms converge to a stationary point. Although this a relatively weak form a convergence, we know of no previous such guarantees for coordinate descent methods for matrix factorization (similar results exist for the cases of alternating least squares and multiplicative updates).

Theorem 1. *Suppose $\lambda > 0$. Then every limit point of the PolyMF with coordinate descent direction and CCD direction is a stationary point.*

Proof. Let A^i, B^i be the solution at i-th outer iteration. Because the method is strictly decreasing, we have

$$\lambda(\|A^i\|^2 + \|B^i\|^2) \le L(A^i, B^i) \le L(A^0, B^0), \ \forall i = 0, 1, \ldots. \quad (23)$$

Thus, the sequence $\{(A^i, B^i)\}$ is bounded. By Weierstrass's sequential compactness theorem, every bounded sequence in a Euclidean space has an converging subsequence. Let (A^*, B^*) be any of the limit point of the converging subsequence.

Now we demonstrate how to use the optimality condition for subproblem to construct a global condition for a stationary point. For the PolyMF with coordinate descent direction, (A^*, B^*) is a fix point means that, taking derivative on (19),

$$\begin{aligned} &\nabla_\alpha F(a_{ik}^*, b_{jk}^*) \\ &= a_{ik}^* b_{jk}^{*2} + R_{ij/k} b_{jk}^* + C_{20}^{i/j} a_{ik}^* + C_{10}^{i/j} = 0, \ \forall (i,j) \in \mathcal{S}. \end{aligned} \quad (24)$$

Expanding the definition of $C_{10}^{i/j}$, $C_{20}^{i/j}$, and $R_{ij/k}$, we obtain

$$\lambda a_{ik}^* + \sum_{q \mid i,q \in \mathcal{S}} R_{iq} b_{qk}^* = 0, \ \forall i, k. \quad (25)$$

By the same reasoning on $\nabla_\beta F$, there is

$$\lambda b_{jk}^* + \sum_{p \mid p,j \in \mathcal{S}} R_{pj} a_{pk}^* = 0, \ \forall j, k. \quad (26)$$

Equations (25) and (26) means that the gradient of the full problem (2) vanishes on (A^*, B^*), because the definition of

	M	N	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}_{\text{test}}\rvert$	k	λ
Movielens10m	71,567	65,133	9,301,274	698,780	5	0.01
Netflix	2,649,429	17,770	99,072,112	1,408,395	10	0.05
Yahoo-Music	1,000,990	624,961	252,800,275	4,003,960	10	1

Table 1: The statistics and parameters for the datasets.

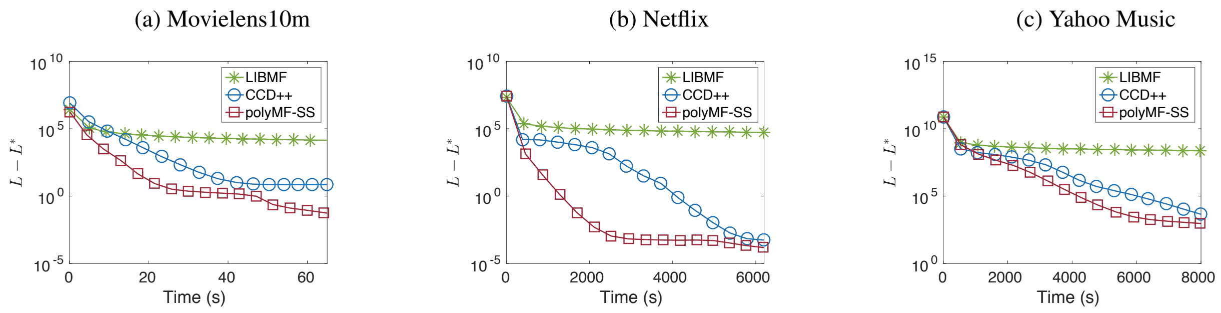

Figure 1: The comparison of $L - L^*$ in terms of time.

the gradient is

$$\nabla_{a_i} L(A, B) = \lambda a_i + \sum_{j|(i,j)\in\mathcal{S}} R_{ij} b_j = 0,\ \forall i, \tag{27}$$

$$\nabla_{b_j} L(A, B) = \lambda b_j + \sum_{i|(i,j)\in\mathcal{S}} R_{ij} a_i = 0,\ \forall j. \tag{28}$$

Thus, (A^*, B^*) is a stationary point. The same reasoning can be also be applied to the proof for CCD++ steps. □

4 Experiments

We evaluate the PolyMF algorithm (both the coordinate descent and subspace search approaches) on several benchmark problems in recommender systems. In particular, we consider three datasets: Movielens10m, Netflix (Bell and Koren 2007), and Yahoo Music (Dror et al. 2012), from smaller to larger. See Table 1 for the dimensions and parameters for each dataset. We first analyze the optimization performance of our methods versus existing state-of-the-art approaches in terms of minimizing the matrix factorization objective, and we then discuss analyses of different additional aspects of the problem. We choose regularization parameter λ as shown in Table 1, and focus our analysis on the optimization performance compared to the following algorithms:

LIBMF This package (Zhuang et al. 2013) implements a parallel SGD algorithm for matrix factorization with adaptive gradient steps (Duchi, Hazan, and Singer 2011). We use its default parameters for learning rate and step-size because the package is already fine-tuned for the above datasets.

CCD++ This package (Yu et al. 2012) implements a parallel coordinate descent method for matrix factorization. We use its default parameters because there is no need to tune the step-size by the nature of coordinate descent.

polyMF-CD This is the pair-wise coordinate descent method proposed in Algorithm 1, in which we perform an optimal subspace search on each pair of coordinate (a_{ik}, b_{jk}).

polyMF-SS This is the subspace search method proposed in Algorithm 2, in which we perform an optimal subspace-search on the CCD++ directions.

We conduct the experiments on a Intel Core i7-4790 machine with 32 GB memory. All experiments except the parallel speedup utilize 4 cores of the total 8 cores available.

4.1 Results on matrix factorization benchmarks

Figures 1 and 2 show the evolution of the objective function $L - L^*$ (where L^* is measured as the best objective found by any of the methods) as a function of running time and iteration count. For all three data sets, the PolyMF-SS algorithm outperforms LIBMF and CCD++ in terms of final objective value and running time versus objective. For example, in the Movielens10m dataset, as shown in Figure 1(a), the PolyMF-SS algorithm actually escapes the local optima that CCD++ converges to, and reaches a substantially lower final objective. In all examples, the convergence to this objective is faster, with the largest gains present on the Netflix example. The PolyMF-CD algorithm here is substantially slower by total time (the time needed to perform a polynomial optimization over each coordinate is prohibitively large), but we show performance per iteration on the smaller Movielens10m data set.

4.2 Comparative analysis

Finally, we analyze a few additional aspects of the algorithm, such as the comparison between the SDP polynomial solver, the data efficiency of PolyMF-CD and the parallel scaling performance of the different algorithms.

Comparison of Kerner Method and SDP We test the two polynomial solvers over 1 million iterations of the inner

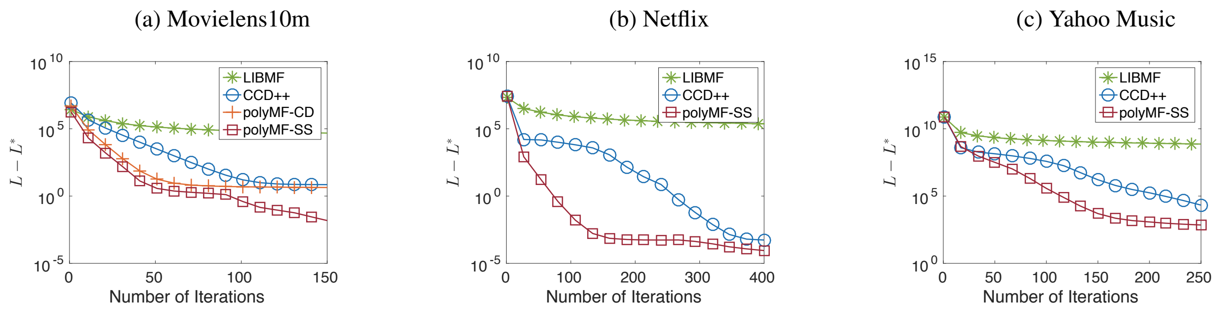

Figure 2: The comparison of $L - L^*$ in terms of the number of iterations.

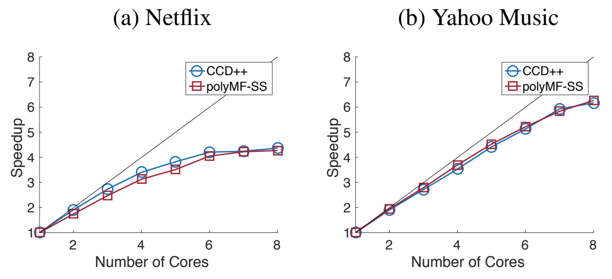

Figure 3: Parallel Speedup

polynomial solve. The moment-based SDP approach takes $1.430 \cdot 10^{-4}s \pm 3.4611 \cdot 10^{-6}s$ (average and standard deviation) to solve each problem, while the Durand-Kerner takes $2.0406 \cdot 10^{-6} \pm 5.1434 \cdot 10^{-7}s$, an improvement about 70 times.

Data efficiency of PolyMF-CD The implementation of CCD++ requires 4 data pass per iteration. Our subspace search method, polyMF-SS, only requires 1 more data pass than CCD++ but saves more iterations. However, both methods require maintaining the transpose of the score matrix, or more precisely, iterating over rows and columns of S. This might be cubersome when the data is too large to be store in the main memory. On the other hand, the PolyMF-CD algorithm requires similar number of data pass and do not need to maintain the transpose matrix.

Parallel performance Finally, we emphasize that because PolyMF-SS spends the bulk of its running time in computing the search directions U and V (using any desired method, though here we use CCD++), the PolyMF-SS inherits the computational advantages of the underlying search direction algorithm. Thus, when applying our PolyMF-SS algorithm to the Netflix and Yahoo Music domains, the speedup of the methods with additional cores is roughly identical to that of CCD++, as shown in Figure 3 (results on Movielens10m are not shown, as the dataset is too small to obtain substantial speedup).

5 Conclusion and future work

In this paper, we have derived and presented a polynomial optimization approach to matrix factorization. Specifically, we have shown that one can perform an exact subspace search over joint update directions in both A and B terms, by solving (exactly) a quartic polynomial optimization problem. We have discussed several approaches for applying these methods in either a direct coordinate descent fashion or in a subspace search fashion to speed up existing matrix factorization approaches. We show that the resulting methods attain faster objective convergence, while also being able to escape local optima that trap alternate approaches.

In future work, we aim to extend these approaches beyond the case of simple matrix factorization to richer settings such as tensor factorization approaches (Kolda and Bader 2009) or factorization machines (Rendle 2012). Both these methods have received significant attention in recent years, yet they suffer from similar local optima problems as our matrix factorization setting, and thus seem ripe for similar approaches.

References

Bell, R. M., and Koren, Y. 2007. Lessons from the netflix prize challenge. *SIGKDD Explorations*.

Blekherman, G.; Parrilo, P. A.; and Thomas, R. R. 2013. *Semidefinite optimization and convex algebraic geometry*. SIAM.

Dror, G.; Labs, Y.; Koenigstein, N.; Koren, Y.; and Weimer, M. 2012. The yahoo! music dataset and kddcup'11. In *Proceedings of KDD Cup 2011 Competition*.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul):2121–2159.

Durand, É. 1959. Solutions numeriques des equations algebriques. tome i: Equations du type $f(x)$.

Gemulla, R.; Nijkamp, E.; Haas, P. J.; and Sismanis, Y. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Kolda, T. G., and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM review* 51(3):455–500.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer*.

Lasserre, J. B. 2001. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*.

Laurent, M. 2009. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*.

Parrilo, P. A. 2003. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*.

Petkovi, M. S.; Carstensen, C.; Trajkov, M.; et al. 1995. Weierstrass formula and zero-finding methods. *Numerische Mathematik*.

Pilászy, I.; Zibriczky, D.; and Tikk, D. 2010. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*.

Recht, B., and Ré, C. 2013. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*.

Rendle, S. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3(3):57.

Yu, H.-F.; Hsieh, C.-J.; Si, S.; and Dhillon, I. S. 2012. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *IEEE International Conference of Data Mining*.

Yu, H.-F.; Hsieh, C.-J.; Si, S.; and Dhillon, I. S. 2014. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*.

Zhou, Y.; Wilkinson, D.; Schreiber, R.; and Pan, R. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*.

Zhuang, Y.; Chin, W.-S.; Juan, Y.-C.; and Lin, C.-J. 2013. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*.

A Proof on transformation of (7)

In this section, we will show that any subspace search problem can be reduced to (7), with the coefficients of the α^2 and β^2 terms being positive. Consider subspace search on arbitrary pair of direction U and V,

$$\underset{\alpha,\beta}{\text{minimize}} \quad L(A+\alpha U,\ B+\beta V). \tag{29}$$

Expand the definition of L in (2), we have

$$\begin{aligned}
&\frac{1}{2}L(A+\alpha U,\ B+\beta V) &(30)\\
&=\frac{\lambda}{2}\Big(\sum_{i=1}^{M}\|a_i+\alpha u_i\|^2+\sum_{j=1}^{N}\|b_j+\beta v_j\|^2\Big)\\
&\quad+\frac{1}{2}\sum_{(i,j)\in\mathcal{S}}\big((a_i+\alpha u_i)^\top(b_j+\beta v_j)-S_{ij}\big)^2 &(31)\\
&=\frac{1}{2}C_{22}\alpha^2\beta^2+C_{21}\alpha^2\beta+C_{12}\alpha\beta^2+C_{11}\alpha\beta\\
&\quad+\frac{1}{2}C_{20}\alpha^2+C_{10}\alpha+\frac{1}{2}C_{02}\beta^2+C_{01}\beta, &(32)
\end{aligned}$$

where

$$\begin{aligned}
C_{22}&=\sum\nolimits_{\mathcal{S}}\,(u_i^\top v_j)^2,\\
C_{21}&=\sum\nolimits_{\mathcal{S}}\,(u_i^\top v_j)(u_i^\top b_j),\\
C_{11}&=\sum\nolimits_{\mathcal{S}}\,(u_i^\top v_j)R_{ij}+(u_i^\top b_j)(v_j^\top a_i),\\
C_{12}&=\sum\nolimits_{\mathcal{S}}\,(u_i^\top v_j)(v_j^\top a_i),\\
C_{20}&=\sum\nolimits_{\mathcal{S}}\,(u_i^\top b_j)^2+\lambda\sum_{i=1}^{M}\|u_i\|^2,\\
C_{10}&=\sum\nolimits_{\mathcal{S}}\,R_{ij}(u_i^\top b_j)+\lambda\sum_{i=1}^{M}u_i^\top a_i,\\
C_{02}&=\sum\nolimits_{\mathcal{S}}\,(v_j^\top a_i)^2+\lambda\sum_{j=1}^{N}\|v_j\|^2,\\
C_{01}&=\sum\nolimits_{\mathcal{S}}\,R_{ij}(v_j^\top a_i)+\lambda\sum_{j=1}^{N}v_j^\top b_j. &(33)
\end{aligned}$$

Note that we denote $\sum_{(i,j)\in\mathcal{S}}$ as $\sum_{\mathcal{S}}$ for simplicity. If $C_{22}=0$, we have $u_i^\top v_j=0$ for all $(i,j)\in\mathcal{S}$. This case, $C_{21}=C_{12}=0$ and the minimization of L can be solved by a linear equation. Otherwise, we show that we can remove C_{21} and C_{12} by some transformation, and the resulting equation will satisfy the constraint of (7).

Let $\hat{\alpha}=\alpha+C_{12}$, $\hat{\beta}=\beta+C_{21}$. Normalize every constant $C_{\cdot,\cdot}$ by C_{22} and denote them as $\hat{C}_{\cdot,\cdot}$, we have

$$\begin{aligned}
\frac{1}{2C_{22}}L=&\frac{1}{2}\hat{\alpha}^2\hat{\beta}^2+(\hat{C}_{11}-2\hat{C}_{21}C_{12})\hat{\alpha}\hat{\beta}\\
&+\frac{1}{2}(\hat{C}_{20}-\hat{C}_{21}^2)\hat{\alpha}^2+\frac{1}{2}(\hat{C}_{02}-\hat{C}_{12}^2)\hat{\beta}^2\\
&+(\hat{C}_{10}+2\hat{C}_{12}\hat{C}_{21}^2-\hat{C}_{11}\hat{C}_{21}-\hat{C}_{12}\hat{C}_{20})\hat{\alpha}\\
&+(\hat{C}_{01}+2\hat{C}_{21}\hat{C}_{12}^2-\hat{C}_{11}\hat{C}_{12}-\hat{C}_{21}\hat{C}_{02})\hat{\beta}\\
&+\text{constant}.
\end{aligned}$$

This way, the problem has been reduced to (7). Now we examine the positivity of the coefficient for $\hat{\alpha}^2$. Observe that

$$\hat{C}_{20} - \hat{C}_{21}^2 = \frac{\sum_{(i,j)\in\mathcal{S}}(u_i^\top b_j)^2 + \lambda \sum_{i=1}^{M} \|u_i\|^2}{\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)^2} - \left(\frac{\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)(u_i^\top b_j)}{\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)^2}\right)^2. \quad (34)$$

By Cauchy inequality, we have

$$\left(\sum_{(i,j)\in\mathcal{S}}(u_i^\top b_j)^2\right)\left(\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)^2\right) \geq \left(\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)(u_i^\top b_j)\right)^2. \quad (35)$$

Thus,

$$\hat{C}_{20} - \hat{C}_{12}^2 \geq \frac{\lambda \sum_{i=1}^{M} \|u_i\|^2}{\sum_{(i,j)\in\mathcal{S}}(u_i^\top v_j)^2} > 0. \quad (36)$$

Similarly, the coefficient $\hat{C}_{02} - \hat{C}_{12}^2$ for $\hat{\beta}^2$ is also positive.

B Example of local minima

Consider a toy problem of 2 scores, $\{10, -10\}$, with $A \in \mathbb{R}^{1\times 1}$ and $B \in \mathbb{R}^{2\times 1}$.

$$L(A, B) = (a^2 + b_1^2 + b_2^2)/2 + (ab_1 - 10)^2 + (ab_2 + 10)^2.$$

Fixing $b_2 = 1$, we can see that the function plot for (a, b_1) contains exactly two local minima (Figure 4), and only one of which is the global minimum.

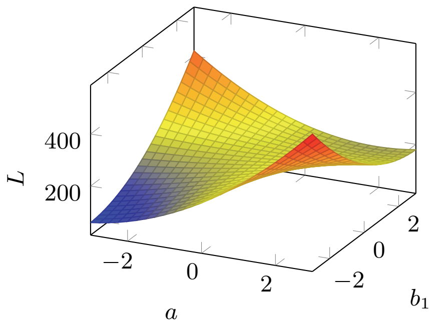

Figure 4: plot of $L(a, (b_1, 1))$

Further, the difference between the two minima can be arbitrary large by scaling the scores. Thus, if not properly initialized, both CCD++ and SGD may stuck at the local minima, but our polyMF method will always converge to global optima because we did an exact subspace search. This is one scenario that our method performs better than the other state-of-the-art methods.