The AAAI-17 Workshop on
Distributed Machine Learning
WS-17-08

Parallel Higher Order Alternating Least Square for Tensor Recommender System

Romain Warlop
fifty-five
SequeL Team, Inria Lille
romain@fifty-five.com

Alessandro Lazaric
SequeL Team, Inria Lille
alessandro.lazaric@inria.fr

Jérémie Mary
Univ. Lille / Inria
jeremie.mary@inria.fr

Abstract

Many modern recommender systems rely on matrix factorization techniques to produce personalized recommendations on the basis of the feedback that users provided on different items in the past. The feedback may take different forms, such as the rating of a movie, or the number of times a user listened to the songs of a given music band. Nonetheless, in some situations, the user can perform several actions on each item, and the feedback is multidimensional (e.g., the user of an e-commerce website can either click on a product, add the product to her cart or buy it). In this case, one can no longer view the recommendation problem as a matrix completion, unless the problem is reduced to a series of multiple independent problems, thus loosing the correlation between the different actions. In this case, the most suitable approach is to use a tensor approach to learn all dimensions of the feedback simultaneously. In this paper, we propose a specific instance of tensor completion and we show how it can be heavily parallelized over both the dimensions (i.e., items, users, actions) and within each dimension (i.e., each item separately). We validate the proposed method both in terms of prediction accuracy and scalability to large datasets.

Introduction

The aim of personalized recommender systems (RSs) is to discover the preferences of each user and to predict which items, among those available in the system, she likes the most. The collaborative filtering approach (Hu, Koren, and Volinsky 2008; Zhou et al. 2008; Koren, Bell, and Volinsky 2009) relies only on the identifiers of users and items to infer implicit descriptions suitable for prediction. Since its success in the Netflix Prize competition (Bennett et al. 2007), one of the most popular collaborative-filtering methods is based on matrix factorization (MF) techniques for latent factor models. In this approach, the RS problem is reduced to the problem of completing a highly sparse user/item matrix (i.e., each entry records the feedback of the user for a specific item) under low-rank assumptions.

Despite its success, MF cannot always capture the full complexity of the problem. In several domains, users can perform several actions on the items. For instance, in e-commerce applications, while the user-item affinity is often predicted on the basis of the number of times the user clicked on the item, the action one would like to predict is whether the user will actually add the product to her cart and/or buy it. Unfortunately, these actions are much sparser than the clicks, thus far more difficult to predict, and they are not necessarily positively correlated with them, so predicting click will not make an effective RS. As a result, applying MF only on the click action may result in poor recommendation for the actual actions of interest (i.e., buy). Another scenario in which MF is not satisfactory is for RS where the recommendation needs to be related, e.g., with a specific moment of the day. For instance, previous studies on music recommendation (Baltrunas and Amatriain 2009) have shown that people listen to different artist depending on the moment of the day or of the week (while working, driving, resting at home). Again, ignoring this dimension and apply MF directly on the user-item matrix, may severely affect the final performance of the RS.

Related work. An effective approach in case of multidimensional feedback is to use tensor factorization (TF) techniques, which extends MF by taking into consideration all actions/contexts at the same time. In particular, the user-item matrix is expanded by adding one or more modes, which contains the additional type of actions available. TF is considerably more complicated than MF and it requires extending previous methods. Symeonidis, Nanopoulos, and Manolopoulos (2008) rely on a generalization of the singular value decomposition for tensors called *higher-order singular value decomposition* (HOSVD) introduced by Lathauwer, Moor, and Vandewalle (2000), part of the general family of Tucker decomposition. In HOSVD, the n-order tensor is unfolded into n matrices on which classic SVD is applied. Its decomposition is then used to reconstruct the full tensor factorization. An alternative approach is to perform tensor decomposition using PARAFAC (parallel factor analysis). For instance, Hu et al. (2013) performed cross-domain recommendation using an alternating least squares algorithm on the PARAFAC model called CP-ALS-R (PARAFAC is also called CANDECOMP, that is what the CP stand for here). However, recommendation problems always come with missing values (otherwise there would be nothing to learn) and this problem has not been taken into account neither in CP-ALS-R nor HOSVD. This problem has been studied for tensor decomposition for instance in the chemometric field for the PARAFAC decomposition (Acar

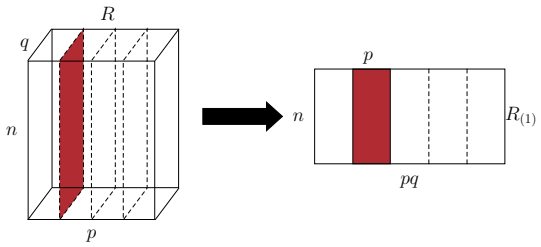

Figure 1: Mode matrix unfolding of a 3-order tensor.

et al. 2010; Tomasi and Bro 2005) and the Tucker decomposition (Filipović and Jukić 2013). Finally, Romera-Paredes and Pontil (2013) studied the tensor completion problem not with tensor factorization but working on the trace norm regularization and introduced a new convex relaxation method for tensor completion. Unfortunately, most of these techniques suffer from different shortcomings when applied to RS: **1)** they may not properly manage missing values (i.e., by filling unknown feedback with arbitrary values), **2)** they do not allow for modes with latent factors of different size (i.e., constraining to a single dimension for all factors), **3)** they do not scale to large tensors (both in size and number of modes).

Contributions. In this paper, we focus on an adaptation of the alternating least squares approach used in MF. In dealing with matrix factorization for RS, alternating least squares with λ regularization (ALS-WR) introduced by Zhou et al. (2008) proved to perform much better than SVD. While both approaches assume a low-rank constraint on the matrix, they differ in the way they deal with unknown values. While SVD implicitly sets all unknown values to 0, the ALS-WR minimizes the ℓ_2-norm error between the true feedback and the factorized matrix only for the available entries, so no assumption is made on the unobserved values. Furthermore, the ALS-WR also introduces a penalty term based on Tikhonov regularization which penalizes large parameters to prevent from overfitting. Building on this observation, we study a method called HOALS, for Higher Order Alternating Least Squares, that leverages alternating least-squares to perform tensor factorization. Beside being better designed for RS, we prove that HOALS can be easily parallelized both over tensor modes and within each dimension, thus allowing it to be applied to large RS. Finally, we show that the HOALS formulation allows to manage implicit feedback along the line of (Hu, Koren, and Volinsky 2008). To the best of our knowledge, HOALS is the only tensor factorization model that allows to perform tensor factorization for implicit feedback and that deal with missing data and it is scalable to large problems.

Preliminaries

RS as tensor completion. We define a recommender system as a tuple $\langle \mathcal{U}, \mathcal{I}, \mathcal{A} \rangle$, where $\mathcal{U} = (1, \dots, n) \doteq [n]$ is the set of users, $\mathcal{I} = (1, \dots, p) \doteq [p]$ is the set of items available in the system, and $\mathcal{A} = (1, \dots, q) \doteq [q]$ is the set of actions the user can take over items (e.g., a binary value such as click, add to cart (ATC), buy, or a number as in the case of a rating).[1] In the following we use u, i, and a to index the elements in the sets $\mathcal{U}$, $\mathcal{I}$, and $\mathcal{A}$ respectively. Each feedback is a scalar value $r_{uia} \in \mathbb{R}$, which denotes the feedback provided by user u on item i through action a. For instance, r_{uia} may be the rating provided by user u to the feature a of a restaurant i. Once organized along the sets $\mathcal{U}$, $\mathcal{I}$, $\mathcal{A}$, all the possible feedback take the form of a tensor (i.e., a multi-dimensional array) $\mathbf{R}$ of dimension $n \times p \times q$. Nonetheless, only a limited number of feedback is available to the system. We denote by $\mathcal{D} \subseteq \mathcal{U} \times \mathcal{I} \times \mathcal{A}$ the set of triples (user, item, action) for which a feedback is available. Starting from the feedback available for the tuples in $\mathcal{D}$, the objective of a RS is to complete the tensor $\mathbf{R}$ with all missing entries and propose the best recommendations for each user. While in the case of one single action, for any user u the RS simply recommends the item i with largest value r_{ui}, when multiple actions are available, the RS should combine different values r_{iua} to find a global index of preference. For instance, a RS can choose to recommended the restaurant with the best average rating over all features (i.e., average of ratings for service, food, and decoration) or one can introduce a filtering on the rating of different actions (e.g., filter out restaurants with decoration rated less than 7/10) and then pick the one according to a chosen feature (e.g., the one with the best rated food).

Tensor notation. In the following we use bold upper case letters, such as $\mathbf{X}$, to denote tensors, whereas matrices are denoted by upper case letters such as X. The pseudo-inverse of a matrix is denoted by X^*. The entry of a third-order tensor is written with a lower case (e.g., x_{i_1,i_2,i_3}) where first, second and third indices represents respectively row, column and depth (similar for matrices, X_{i_1,i_2} denotes the element in row i_1 and column i_2). In the general n-order tensor we may use *fiber* instead of row, column and depth, to denote one dimensional part of the tensor, which can not find equivalent when n is greater than 3. A particular fiber is called mode-i. For instance, for a third-order tensor the column correspond to mode-1. Moreover a two dimensional part of the tensor is called a *slice*. In the special case of matrices, we denote the uth row of X by X_u. We recall some basic definitions and operations on tensors. Let $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be a N-dimensional tensor, we define the n-mode product of tensor $\mathbf{X}$ by a matrix $U \in \mathbb{R}^{J_n \times I_n}$, denoted $\mathbf{X} \times_n U$, as the tensor of dimension $(I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N)$ obtained as

$$\left[\mathbf{X} \times_n U\right]_{i_1 \cdots i_{n-1} j_n i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} X_{i_1 \cdots i_n \cdots i_N} U_{j_n, i_n}.$$

The mode-n product is commutative when applied in distinct modes, i.e. for $m \neq n$,

$$(\mathbf{X} \times_n A) \times_m B = (\mathbf{X} \times_m B) \times_n A.$$

[1] In the following, we only consider only one action dimension, but in general other dimensions could be available. For instance, we could consider the rating attributed by a user to the food (i.e., the action) of a restaurant in a specific season (i.e., an additional dimension of interest).

Another important operation is the matrix unfolding of the tensor. There are N different ways to unfold a N-order tensor into a matrix called 1-mode, 2-mode,..., N-mode matrix unfolding. For instance for $N = 3$, one can stack the row, the column, or the depth as shown in Figure 1 for one mode. In general, the d-mode matrix unfolding results in a matrix $X_{(d)} \in \mathbb{R}^{I_d \times I_1 \cdots I_{d-1} I_{d+1} \cdots I_N}$.

Tensor factorization. Unlike matrix factorization, tensor decomposition does not have a unique formulation and many different methods exist to recover a tensor as a combination of multiple factors (see e.g., (Kolda and Bader 2009)). The two principals tensor decomposition formulation are the CANDECOMP-PARAFAC (*CP* for short) decomposition and the Tucker decomposition. The CP decomposition decomposes a n-order tensor into a sum of rank-one tensor. More precisely, a three-order tensor $\mathbf{X} \in \mathbb{R}^{N \times M \times P}$ is decomposed as

$$\mathbf{X} = [A, B, C] = \sum_{k=1}^{K} A_{:,k} \otimes B_{:,k} \otimes C_{:,k},$$

where $A \in \mathbb{R}^{N \times K}$, $B \in \mathbb{R}^{M \times K}$, $C \in \mathbb{R}^{P \times K}$, K is an arbitrarily chosen dimension, and $\otimes$ denotes the outer product. As a result, the entries of the tensor can be computed as $\mathbf{X}_{ijp} = \sum_{k=1}^{K} A_{i,k} B_{j,k} C_{p,k}$. The minimal value of K for which a decomposition exists defines the rank of the tensor. On the contrary, the Tucker decomposition assumes that the tensor $\mathbf{X}$ can be factorized as

$$\mathbf{X} = \mathbf{W} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$$

where $U^{(d)}$ contain orthonormal vectors called the d-mode singular values and $\mathbf{W}$ is the core tensor. In general, for any $d \in \{1, 2, 3\}$ the matrices are $U^{(d)} \in \mathbb{R}^{I_d \times k_d}$ and the core tensor is $\mathbf{W} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$, where k_1, k_2, and k_3 are dimensions arbitrarily chosen.

In general, computing the CP and the Tucker decompositions is a computationally challenging problem. In the next section we present an efficient parallel algorithm, called *higher order alternating least-squares* (HOALS), to perform the general tensor factorisation. In the following section we compare HOALS to other state of the art methods.

Higher Order Alternating Least Squares

HOALS is an algorithm for tensor completion through the factorization of the tensor into either a Tucker or CP decomposition. For sake of clarity, we detail the algotihm for the Tucker decomposition and we postpone the CP decomposition case in the appendix. Besides, while the algorithm applies for any n-order tensor, we describe it for $n = 3$ for the sake of simplicity. The main idea of HOALS is that the decomposition of an n-order tensor can be parallelized over each of its modes independently, thus reducing it to a series of matrix decomposition operations. Furthermore, each matrix decomposition can be further parallelized across each component. The resulting process can be easily split over multiple machines, whose jobs are completely independent and whose results are merged only at the last step.

A RS is provided with a tensor $\mathbf{R}$ which is only partially filled with the feedback in $\mathcal{D}$ (i.e., feedback for triples *(user, item, action)*) and the objective is to construct a full tensor $\widehat{\mathbf{R}}$ with entries that are as close as possible to the "true" ones (i.e., the feedback a user would give to an item for a specific action). Similar to the case of matrix completion, this problem is ill-posed (i.e., all tensors $\widehat{\mathbf{R}}$ with the same entries as $\mathbf{R}$ in $\mathcal{D}$ are equivalent) unless constraints on the structure of the tensor $\widehat{\mathbf{R}}$ are introduced. In particular, we introduce rank-constraints in the decomposition of $\widehat{\mathbf{R}}$ either following Tucker or CP.

Algorithm 1 Direct-HOALS for Tucker factorization.

Input: dataset $\mathcal{D}$, desired ranks $\{k_d\}_{d=1}^{3}$
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Initialize $\mathbf{W}$, I and A with small random numbers
while Not at convergence **do**
  Fix $\mathbf{W}$, I, A and update U
  Fix $\mathbf{W}$, U, A and update I
  Fix $\mathbf{W}$, U, I and update A
  Compute $\mathbf{W} \doteq \mathbf{R} \times_1 U^* \times_2 I^* \times_3 A^*$
end while
Return full tensor $\widehat{\mathbf{R}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A$

Following the definition of Tucker decomposition, we work on tensors $\widehat{\mathbf{R}}$ that can be decomposed as

$$\widehat{\mathbf{R}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A,$$

With this decomposition we have the flexibility of choosing the ranks k_U, k_I, and k_A of each of the latent factors independently. This is particularly important in RSs where the set of users, items, and actions have different complexity and require different degrees of freedom in their description. The objective is to find the tensor $\widehat{\mathbf{R}}$ that minimizes the squared loss with a penalty term related to the latent factors weighted by the number of observed interactions. We thus want to find the components latent factors U, I, and A and the core tensor W the minimize the following loss function

$$\mathcal{L}(W,U,I,A) = \sum_{u,i,a \in \mathcal{D}} \left(\mathbf{R}_{uia} - \widehat{\mathbf{R}}_{uia} \right)^2 + \lambda \left(\sum_{u=1}^{n} n_u^{\mathcal{U}} ||U_u||^2 + \sum_{i=1}^{p} n_i^{\mathcal{I}} ||I_i||^2 + \sum_{a=1}^{q} n_a^{\mathcal{A}} ||A_a||^2 \right), \quad (1)$$

where $n_u^{\mathcal{U}}$ (resp. $n_i^{\mathcal{I}}$, $n_a^{\mathcal{A}}$) is the number of known entries for user u (resp. item i, a) in $\mathcal{D}$ and λ is a regularization parameter.

Direct-HOALS. The loss function in Eq. 1 can be minimized with an alternating scheme similar to ALS-WR for MF (see Alg. 1). In this case, instead of repeating an alternate update on users and items only, also the action mode is updated while keeping the other two modes fixed. While a simple and smooth generalization of ALS to tensor factorization, this algorithm, called *Direct*-HOALS, increases the complexity of the original ALS-WR by adding an update over the action mode. This process may be computationally expensive and prevent from scaling this algorithm to tensors of higher modes.

Algorithm 2 Parallel-HOALS for Tucker factorization.

Input: dataset $\mathcal{D}$, desired ranks $\{k_d\}_{d=1}^3$
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Compute the three matrices unfolding $R_{(1)}$, $R_{(2)}$, $R_{(3)}$
In parallel apply [2]
  ALS-WR on $R_{(1)}$, $\widehat{R}_{(1)} = M^1 {V^1}^\top$ and set $U = M^1$
  ALS-WR on $R_{(2)}$, $\widehat{R}_{(2)} = M^2 {V^2}^\top$ and set $I = M^2$
  ALS-WR on $R_{(3)}$, $\widehat{R}_{(3)} = M^3 {V^3}^\top$ and set $A = M^3$
Compute $\mathbf{W} \doteq \mathbf{R} \times_1 U^* \times_2 I^* \times_3 A^*$
Return full tensor $\widehat{\mathbf{R}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A$

Parallel-HOALS. Instead of running ALS over all modes iteratively, we design an alternative algorithm, called *Parallel*-HOALS, which runs ALS separately on each mode and collecting its result to obtain the final tensor decomposition. The idea is to first construct the three modes $R_{(1)}$, $R_{(2)}$, and $R_{(3)}$ and perform ALS-WR on each of them to find a suitable factorization of the corresponding matrix. The first factors of each of these decompositions can be then collected to construct the factorization of the initial tensor. The resulting algorithm is illustrated in Alg. 2. Despite its difference with the sequential implementation of the ALS idea of *Direct-HOALS*, the following theorem shows that (the proof is reported in the appendix), *Parallel*-HOALS returns exactly the same tensor at the end.

Theorem 1 *Solving U (resp. I, A) with alternating least square and k_U (resp. k_I, k_A) latent factor on the tensor $\widehat{\boldsymbol{R}}$ is equivalent to applying alternating least square on $R_{(1)}$ (resp. $R_{(2)}$, $R_{(3)}$) with k_U (resp. k_I, k_A) latent factor and identify U (resp. I, A) as the left side matrix in the matrix factorization.*

Notice that since this result applies separately for each mode it can be extended to an arbitrary number of modes and it applies to both Tucker and CP decompositions. The biggest advantage of this version of HOALS is that the three applications of ALS-WR on $R_{(1)}$, $R_{(2)}$, and $R_{(3)}$ can be computed in parallel. Furthermore, since ALS itself is highly parallelizable (Zhou et al. 2008), each instance of ALS-WR associated to any mode can be further parallelized, thus making HOALS very efficient without suffering any major overhead from the number of modes and elements in each mode. The update complexity for each event (i.e., update of the latent factor for one user, one item, or one action) for matrix $R_{(d)}$ is $\mathcal{O}\left({f_d}^2 n_s^d + {f_d}^3\right)$, where n_s^d is the number of non-zero elements on row $R_{(d),s}$ and f^d is the rank constraint. Since in HOALS we can parallelize the computation of each matrix and within each matrix, the complexity of one update is due to the largest latent dimension over the three dimensions, i.e., $\mathcal{O}\left(\max_{d\in\{1,2,3\},s} ({f_d}^2 n_s^d + {f_d}^3)\right)$.

Additional advantages. Finally, we would like to emphasize some features that make the specific tensor decomposition in HOALS particularly suitable for RS, unlike other TF methods. First, our method use Tikhonov regularization with different weight on each mode that regularized appropriately each dimensions and is crucial for such application since the number of action provided by each user can be highly heterogeneous. Second, tensor decomposition may appears difficult in practice because of memory issues that can arise from the additional modes. However thanks to its appropriate treatment of missing values and the low rank decomposition, memory is not an issue in practice since HOALS never requires to store the full tensor.

Experiments

We tested both the accuracy and the scalability of our method on different types of datasets:

- rather small datasets for recommendation problem but still large for common machine learning problem. This datasets allowed us to make a lot of experiments on a substantial number of algorithms and tuning them quite effectively. These datasets also emphasize the benefit from using tensor decomposition instead of matrix decomposition. For the sake of space, we present the results on a e-commerce dataset in the main paper and on Last.fm in the appendix.
- a big dataset to test the scalability performance as a function of the dataset size and the number of machines used.

Testing accuracy

We report an empirical comparison of a number of matrix and tensor decomposition methods. The objective is twofold: **1)** investigate whether tensor completion enables an effective transfer across different actions that improves over matrix completion where each action is treated separately, **2)** compare HOALS to state-of-the-art methods.

E-commerce dataset (authors 2015): this dataset contains historical interactions (click, add to cart, buy) from a real e-commerce website of 1290 users and 390 items, resulting in around 120k *(user, item, action, value)* quadruplets. The *value* is the number of times the user performed the action on the item.

Algorithms. Beside HOALS, we tested four other state-of-the-art algorithms, two with different initialization

- **MI-SVD**: multiple independent SVD applied to each slice of the tensor,
- **HOSVD v0/vMean**: as proposed by (Symeonidis, Nanopoulos, and Manolopoulos 2008), filling missing values with either 0 or the mean of the modes,
- **MI-ALS**: the classic ALS-WR applied to each slice of the tensor, that is three independent ALS-WR,
- **CP-ALS-R v0/vMean** (aka PARAFAC-ALS) as in Alg.1 of (Hu et al. 2013), filling missing values with 0 or the mean of the modes. We used the cp_als python code from the sktensor package,
- **tucker/CP-HOALS**, our parallel algorithm described in Alg. 2. The implementation is available at (authors 2015).

The *vMean* versions had only been tested on the E-commerce dataset because filling with non zero values require much more memory, since sparse representations of the structures cannot be used any more. A modification of

the ALS-WR more suitable for implicit feedback datasets, like the Last.fm dataset, is proposed in (Hu, Koren, and Volinsky 2008). The algorithm is very similar to the classic ALS-WR but it penalizes each user-item affinity error by a weight depending on the true user-item value. In the following, we use exactly the same approach to obtain an implicit version of HOALS. Experiments test both classic and implicit methods for ALS-WR and HOALS, which are referred to *MI-ALS* and *HOALS* for the classic version and *imp-MI-ALS* and *imp-HOALS* for the implicit counterparts. Finally, we also tried the algorithms proposed by (Romera-Paredes and Pontil 2013) and (Acar et al. 2010) but their solutions could not scale to the size of our datasets.

Experimental protocol. We first executed a preprocessing step.

1. We filtered out items seen by less than 5 users,
2. We define a maximum threshold for each action used to cap all the values (i.e., $r_{uai} = \min\{r_{uai}, \mathit{threshold}_a\}$). This prevents from having outliers that tend to degrade the performance. The click threshold was set to 50, the ATC to 20 and buy to 10.
3. We scaled all values between 0 and 10 in order to have comparable slices. This is needed since click, ATC, and buy actions are at very different scales.

For this experiment we performed 5-cross validation. We first randomly split the users into five groups and at each round we selected four groups to form the *training* users and the remaining one as *testing* users. Then, we selected all the data belonging to a train user plus 70% at random from the test users to form the train set. The 30% of data left from the test users form the actual test set. Notice that this selection protocol for the test user is needed in order to have some feedback for each user, otherwise no prediction can be made for completely unknown users. For evaluating the recommendation performance, we use the rank-measure error introduced by Hu, Koren, and Volinsky (2008), which is more suitable for implicit feedback datasets. The rank-measure error is computed as

$$\overline{\rho_a} = \frac{\sum_{u,i} r_{u,i,a}\rho_{u,i,a}}{\sum_{u,i} r_{u,i,a}}, \tag{2}$$

where $\rho_{u,i,a}$ denotes the percentile-ranking of the item i among all items previously seen by user u for action a ordered by decreasing value of $r_{u,i,a}$. For instance, if item i is predicted to be the most (resp. the least) suitable for user u, then $\rho_{u,i} = 0\%$ (resp. $\rho_{u,i} = 100\%$). If the predictions are made at random, the expected rank-measure is 50%. Low values of $\overline{\rho}$ indicate that most of highly seen items have been predicted to be the most suitable and thus corresponds to a good recommendation.

Results. The results are reported in Table 1. For Tucker models we tried 10, 50, 100 and 200 number of latent vectors for each possible tuning combination for the user and item slice, and 2 latent vectors for the action slice, since the depth of the tensor is always 3 in our cases. For CP decompositions, given that the rank has to be the same for all modes, we tried $2, 3, 4, 5$ and 10 latent vectors for all modes.

model	dim 1	dim 2	action	$\overline{\rho_a}$
imp Tucker-HOALS	10	200	click	**28.79**
Tucker-HOALS	200	200	click	30.34
CP-HOALS	5	-	click	31.92
imp CP-HOALS	2	-	click	32.01
CP-ALS-R	2	-	click	32.20
MI-ALS	50	-	click	35.89
HOSVD v0	10	200	click	36.39
HOSVD vMean	10	100	click	36.42
imp MI-ALS	100	-	click	36.66
MI-SVD	10	-	click	37.11
imp Tucker-HOALS	10	200	atc	**28.58**
Tucker-HOALS	200	200	atc	31.63
imp CP-HOALS	2	-	atc	31.70
CP-HOALS	5	-	atc	31.91
CP-ALS-R	2	-	atc	32.11
HOSVD vMean	10	200	atc	34.94
HOSVD v0	10	200	atc	35.08
MI-SVD	10	-	atc	36.25
MI-ALS	10	-	atc	36.63
imp MI-ALS	100	-	atc	43.14
imp Tucker-HOALS	10	200	buy	**28.13**
CP-ALS-R	4	-	buy	31.19
imp CP-HOALS	2	-	buy	31.45
CP-HOALS	5	-	buy	31.74
Tucker-HOALS	10	200	buy	32.38
HOSVD vMean	10	100	buy	34.56
HOSVD v0	10	50	buy	34.88
MI-SVD	10	-	buy	35.49
MI-ALS	10	-	buy	36.79
imp MI-ALS	50	-	buy	41.96

Table 1: $\overline{\rho_a}$ measure (rank-measure) on the E-commerce dataset ordered by decreasing performance. For MF and CP, *dim*1 refers to the common latent space. For TF, *dim*1 and *dim*2 refers to the number of latent vectors for the user and item space respectively and *dim*3 is always equal to 2.

We were not able to test higher number of latent vectors because of memory issue when trying to factorize the third unfolding matrix of size $(q \times np)$, which illustrates one of the drawbacks of CP decompositions. Furthermore, notice that increasing the number of latent vectors for the CP decomposition is not really meaningful since the action mode only contains three different actions. Finally for matrix decomposition we tried 10, 50, 100 and 200 number of latent vectors for each slice. For the sake of space we do not provide the parameters tuning results for each combination (several hundreds) and only report the performance for each model in their best configuration for each action.

In e-commerce applications, recommendations are usually performed using only the click action because of the sparsity of the buy action. For instance, in this dataset the sparsity level for click, ATC, and buy are respectively 85%, 94% and 98%. However, the results in Table 1 show that Tucker HOALS (implicit or not) outperformed every other model obtaining a $\overline{\rho}$ measure around 28% for the implicit

version, even for the buy action, while other models are at around 35%. The implicit Tucker HOALS improves the classic Tucker HOALS performance by more than two points, which illustrates the benefit of such of modelisation. Even the CP models outperformed the other factorization methods with a $\overline{\rho}$ measure around 32%. This indicates that the HOALS is able to perform better recommendation for the optimization of the purchase and is more suitable than the other tensor factorization methods thanks to its ability to deal with missing values. Furthermore, tensor factorization methods achieve better results than their matrix factorization counterparts (HOALS vs ALS and HOSVD vs SVD), which shows the advantage of transferring information across actions within this dataset. However, the HOSVD improvement over MI-SVD is relatively small, about 1 point, whereas the tucker HOALS improvement over MI-ALS is around 5 points for the classic algorithm and 12 points for the implicit one. These results show the effectiveness of HOALS in overcoming the sparsity issue by sharing knowledge across the different actions. Finally, we see no real improvement in the initialisation of missing values with 0 or with mean values.

Testing scalability

For this experiment we used a real world dataset referred as **big E-commerce dataset** (authors 2015). The full dataset is composed of 100k users, around 25k items and 4 actions (view the product, click on the product, add it to cart or buy it) resulting in around 53M (user, item, action, value) quadruplets. The view action is a passive action, a user is exposed to an item by the website but cannot directly decide to see it or not. Thus this action is not really interesting to predict but can be very informative. Indeed a user may be more likely to click, add to her cart or buy a product if she sees it a lot. It is then a valuable context information that can be easily taken into account using tensor decomposition.

To test the scalability we measured the execution time when varying the size of the dataset and number of machines. For the size of the dataset we selected either $1k$, $2k$ or $3k$ users and all the items (since the files are split every 1k users), knowing that $1k$ users result in a dataset of around 9Mo and 530k lines, i.e., quadruplets. We performed our experiments on *Google Cloud* on the n1-standard-2 machine, which is standard 2-CPU machine with 2 virtual CPUs and 7.5 GB of memory. When running the three decompositions in three independent cluster, the execution time depends on the execution time of the longest decomposition. We then report here the execution time of the longest decomposition for dataset size. The code is available at (authors 2015) for reproducibility.

Results. Fig. 2 compares *Direct* and *Parallel*-HOALS. We see that the execution times decreases linearly as the number of workers increases. Doubling the number of workers does not divide the execution time by a factor of 2 (for instance 2 workers, 1k users need 1200 seconds and with 4 workers it need 780 seconds) because of the set up and the communication time. Adding 1CPU decreases the running time in average of 200 (resp. 300 and 400) seconds for 1k users (resp. 2k and 3k). Thus adding users affect mainly the set up and communication time but the parallelisation is more and more effective. We thus see that the HOALS algorithm scale easily for big dataset thanks to its high parallelisation power. Furthermore, this improvement would be greatly amplified as the number of modes increases. For instance, the performance of RS in e-commerce can be significantly improved by adding new contexts such as the day of the week and the season of the year, thus resulting in a 5-order tensor completion problem.

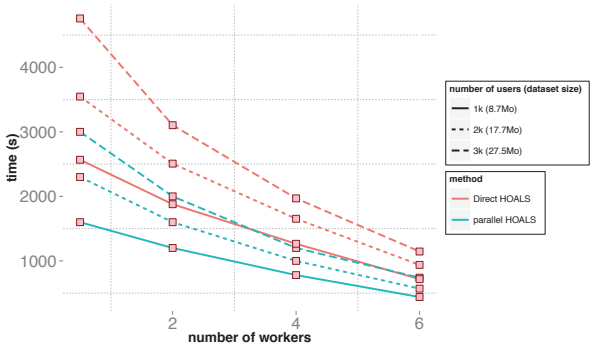

Figure 2: Result of the scalability test of parallel HOALS ran with the implicit version with rank constraints of 200 for users and items and 2 for actions. Each workers has 2CPUs.

Conclusion

In this paper we introduced a parallel tensor factorization algorithm for RSs built on an extension of alternating least squares. For e-commerce, the buy action is the most preferable action to predict but also the most difficult for matrix factorization given its high sparsity. However, our HOALS model proved to be effective to predict any action by leveraging knowledge from the other actions and thus make it an efficient transfer learning algorithm for RSs. An important advantage of HOALS over state-of-the-art methods such as CP-ALS-R and HOSVD is its high ability to be parallelized, and thus it is feasible for large RSs dealing with millions of users and items. For instance, for a n-order tensor, since CP-ALS-R computes matrices one after the other, HOALS is n times more parallelizable. We also extended the implicit feedback strategy introduced in Hu, Koren, and Volinsky (2008) to tensor factorization, making our algorithm a very effective approach to solve efficiently RSs with implicit and multiple feedbacks.

Acknowledgements

This work was supported by CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020, the French Ministry of Higher Education and Research, the French National Research Agency project ExTra-Learn (n.ANR-14-CE24-0010-01) and the ANRT through the CIFRE contract 2015/0326.

References

Acar, E.; Dunlavy, D. M.; Kolda, T. G.; and Mørup, M. 2010. Scalable tensor factorizations with missing data. In *SDM10: Proceedings of the 2010 SIAM International Conference on Data Mining*, 701–712.

authors, A. 2015. E-commerce dataset and hoals code. goo.gl/kyPlGx.

Baltrunas, L., and Amatriain, X. 2009. Towards time-dependant recommendation based on implicit feedback. In *In Workshop on context-aware recommender systems (CARS-09)*.

Bennett, J.; Eklan, C.; Liu, B.; Smyth, P.; and Tikk, D., eds. 2007. *KDD Cup and Workshop 2007*. ACM SIGKDD Explorations Newsletter.

Celma, O. 2009. Last.fm dataset-1k users. goo.gl/VWQKsp.

Filipović, M., and Jukić, A. 2013. Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional Systems and Signal Processing* 1–16.

Hu, L.; Cao, J.; Xu, G.; Cao, L.; Gu, Z.; and Zhu, C. 2013. Personalized recommendation via cross-domain triadic factorization. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, 595–606. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee.

Hu, Y.; Koren, Y.; and Volinsky, C. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, 263–272. Washington, DC, USA: IEEE Computer Society.

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* 42(8):30–37.

Lathauwer, L. D.; Moor, B. D.; and Vandewalle, J. 2000. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* 21(4):1253–1278.

Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2011. A three-way model for collective learning on multi-relational data. In Getoor, L., and Scheffer, T., eds., *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, 809–816. New York, NY, USA: ACM.

Romera-Paredes, B., and Pontil, M. 2013. A new convex relaxation for tensor completion. *CoRR* abs/1307.4653.

Symeonidis, P.; Nanopoulos, A.; and Manolopoulos, Y. 2008. Tag recommendations based on tensor dimensionality reduction. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, 43–50. New York, NY, USA: ACM.

Tomasi, G., and Bro, R. 2005. Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems* 75(2):163–180.

Zhou, Y.; Wilkinson, D.; Schreiber, R.; and Pan, R. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, 337–348. Berlin, Heidelberg: Springer-Verlag.

Proof of Theorem 1

Tucker Decomposition For sake of space, we use the following short summation notation

$$\sum_{m,l,s=1}^{k_U,k_I,k_A} \doteq \sum_{m=1}^{k_U}\sum_{l=1}^{k_I}\sum_{s=1}^{k_A}.$$

Computing the partial derivative of $\mathcal{L}$ in Eq.1 w.r.t. U_{u_0,k_0}, we get:

$$\frac{1}{2}\frac{\partial \mathcal{L}}{\partial U_{u_0,k_0}} = \sum_{(i,a)\in\mathcal{I}_{u_0}} \Big(\sum_{m,l,s=1}^{k_U,k_I,k_A} \mathbf{W}_{m,l,s} U_{u_0,m} I_{i,l} A_{a,s} - \mathbf{R}_{u_0,i,a} \Big) \times \sum_{l,s=1}^{k_I,k_A} \mathbf{W}_{k_0,l,s} I_{i,l} A_{a,s} + \lambda n_{u_0}^{\mathcal{U}} U_{u_0,k_0}, \quad (3)$$

where $\mathcal{I}_{u_0}$ denotes the items and actions seen by user u_0. Let us denote

$$\mathcal{V}_m^{i,a} = \sum_{l=1}^{k_I}\sum_{s=1}^{k_A} \mathbf{W}_{m,l,s} I_{i,l} A_{a,s} \quad (4)$$

and equalize the derivative above to 0, then we get the equation

$$\sum_{(i,a)\in\mathcal{I}_{u_0}} \Big(\sum_{m=1}^{k_U} U_{u_0,m} \mathcal{V}_m^{i,a} \mathcal{V}_{k_0}^{i,a} \Big) + \lambda n_{u_0}^{\mathcal{U}} U_{u_0,k_0} = \sum_{(i,a)\in\mathcal{I}_{u_0}} \mathcal{V}_{k_0}^{i,a} \mathbf{R}_{u_0,i,a} \quad (5)$$

We introduce the mapping g_1 function:

$$\begin{array}{rcl} g_1 : [p]\times[q] & \to & \{1,\cdots,pq\} \\ (i,a) & \mapsto & \text{1-mode unfolding} \\ & & \text{corresponding column index} \end{array}$$

which maps item and action indices to a common index used to refer to the column of the 1-mode unfolding of the tensor. We can use this function to re-index $R^1 \in \mathbb{R}^{n,pq}$ such that $R^1_{u,g_1(i,a)} = r_{u,i,a}$ and $\mathcal{V}^1 \in \mathbb{R}^{pq,k_U}$ such that $\mathcal{V}^1_{g_1(i,a),m} = \mathcal{V}^{i,a}_m$. Let $\mathcal{G}_{u_0} = \{g : 1 \le g \le pq, R^1_{u_0,g} \neq 0\}$ we can then rewrite the left side (LS) of Eq. 5 and obtain $\forall u_0, \forall k_0$:

$$\begin{aligned} (LS) &= \sum_{m=1}^{k_U}\sum_{g\in\mathcal{G}_{u_0}} \mathcal{V}^1_{g,k_0}\mathcal{V}^1_{g,m} U_{u_0,m} + \lambda n_{u_0}^{\mathcal{U}} E_{k_0} U_{u_0}^\top \\ &= \sum_{m=1}^{k_U} [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0,m} U_{u_0,m} + \lambda n_{u_0}^{\mathcal{U}} E_{k_0,:} U_{u_0}^\top \\ &= [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0} U_{u_0}^\top + \lambda n_{u_0}^{\mathcal{V}} E_{k_0} U_{u_0}^\top \end{aligned}$$

Similarly for the right side

$$(RS) = \sum_{g\in\mathcal{G}_{u_0}} R^1_{u_0,g}\mathcal{V}^1_{g,k_0} = {\mathcal{V}^1_{\mathcal{G}_{u_0},k_0}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top.$$

Thus $\forall u_0$, Eq. 5 becomes

$$\left({\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top \mathcal{V}^1_{\mathcal{G}_{u_0}} + \lambda n^{\mathcal{U}}_{u_0} E\right) U^\top_{u_0} \quad = \quad {\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top$$

And finally,

$$U^\top_{u_0} = \left({\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top \mathcal{V}^1_{\mathcal{G}_{u_0}} + \lambda n^{\mathcal{U}}_{u_0} E\right)^{-1} {\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top, \quad (6)$$

which is exactly the classic ALS-WR formula for matrix factorization applied to the 1-mode matrix unfolding (11), which concludes the proof. □

CP Decomposition Computing the partial derivative of $\mathcal{L}$ in Eq.1 w.r.t. U_{u_0,k_0}, we get:

$$\frac{1}{2}\frac{\partial \mathcal{L}}{\partial U_{u_0,k_0}} = \sum_{(i,a)\in\mathcal{I}_{u_0}} \Big(\sum_{k=1}^{K} U_{u_0,k} I_{i,k} A_{a,k} - \mathbf{R}_{u_0,i,a}\Big) \times I_{i,k} A_{a,k} + \lambda n^{\mathcal{U}}_{u_0} U_{u_0,k}, \quad (7)$$

where $\mathcal{I}_{u_0}$ denotes the items and actions seen by user u_0. AS previously, let us denote

$$\mathcal{V}^{i,a}_k = I_{i,k} A_{a,k} \quad (8)$$

and set the derivative above to 0, then we get the equation

$$\sum_{(i,a)\in\mathcal{I}_{u_0}} \Big(\sum_{k=1}^{K} U_{u_0,k} \mathcal{V}^{i,a}_k \mathcal{V}^{i,a}_{k_0}\Big) + \lambda n^{\mathcal{U}}_{u_0} U_{u_0,k_0} \quad (9)$$
$$= \sum_{(i,a)\in\mathcal{I}_{u_0}} \mathcal{V}^{i,a}_{k_0} \mathbf{R}_{u_0,i,a}$$

We use again the function g_1 to re-index $R^1 \in \mathbb{R}^{n,pq}$ such that $R^1_{u,g_1(i,a)} = r_{u,i,a}$ and $\mathcal{V}^1 \in \mathbb{R}^{pq,K}$ such that $\mathcal{V}^1_{g_1(i,a),k} = \mathcal{V}^{i,a}_k$. Let $\mathcal{G}_{u_0} = \{g : 1 \le g \le pq, R^1_{u_0,g} \neq 0\}$, we rewrite the left side (LS) of Eq. 9 and obtain $\forall u_0, \forall k_0$

$$\begin{aligned}(LS) &= \sum_{k=1}^{K} \sum_{g\in\mathcal{G}_{u_0}} \mathcal{V}^1_{g,k_0} \mathcal{V}^1_{g,k} U_{u_0,k} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0} U^\top_{u_0} \\ &= \sum_{m=1}^{k_U} [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0,k} U_{u_0,k} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0,:} U^\top_{u_0} \\ &= [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0} U^\top_{u_0} + \lambda n^{\mathcal{V}}_{u_0} E_{k_0} U^\top_{u_0}\end{aligned}$$

Similarly for the right side

$$(RS) = \sum_{g\in\mathcal{G}_{u_0}} R^1_{u_0,g} \mathcal{V}^1_{g,k_0} = {\mathcal{V}^1_{\mathcal{G}_{u_0},k_0}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top.$$

We can then conclude as previously. □

Related Work

In this section we give a detailed comparison between HOALS and state-of-the-art methods for RSs based on matrix factorization, tensor factorization and transfer learning. Some of the following methods are also compared in the experiments.

Matrix factorization. As illustrated in the previous section, ALS-WR is a matrix factorization method that computes a low-rank approximation of the feedback matrix $R \in \mathbb{R}^{n\times p}$ for each action separately. Under the assumption that R can be factorized in two matrices $U \in \mathbb{R}^{k\times n}$ and $V \in \mathbb{R}^{k\times p}$, matrix completion is performed by minimizing the loss

$$\mathcal{L}(U,V) = \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - U_u V_i^\top\right)^2 + \lambda\Big(\sum_u n^{\mathcal{U}}_u ||U_u||^2 + \sum_i n^{\mathcal{I}}_i ||V_i||^2\Big), \quad (10)$$

where $n^{\mathcal{U}}_u$ (resp. $n^{\mathcal{I}}_i$) is the number of known entries for user u (resp. item i). At each iteration, given a fixed V, the matrix U is updated for user u as

$${U_u}^\top = \left(V^\top_{\mathcal{G}_u} V_{\mathcal{G}_u} + \lambda n^{\mathcal{U}}_u E\right)^{-1} V^\top_{\mathcal{G}_u} {R_{u,\mathcal{G}_u}}^\top \quad (11)$$

where $\mathcal{G}_u$ denotes the indices of the known entries for user u, and E is the identity matrix in $\mathbb{R}^k$. The item update rule is obtained by inverting the role of U and V. Those updates can be perform in parallel for each user or each item thus allowing a good scalability to a large number of users and items.

PARAFAC decomposition.

In (Hu et al. 2013), the problem of finding a CP decomposition of a generic tensor $\mathbf{X}$ is restated as a least-squares problem, i.e., finding the factors A, B, C that minimize the loss

$$f(A,B,C) = \frac{1}{2}||\mathbf{X} - [A,B,C]||^2_F + \frac{\lambda_A}{2}||A||^2_F + \frac{\lambda_B}{2}||B||^2_F + \frac{\lambda_C}{2}||C||^2_F.$$

Computing the derivative and setting it to zero yields to a closed-form expression for each matrix, based on the pseudo-inverse property of Khatri-Rao product. One can then alternately learn each of the matrices until convergence. The resulting algorithm called **CP-ALS-R** has the advantage to be simple and easily implemented. However, this approach suffers from major limitations. First, the definition of the least-squares problem does not deal with missing values and the tensor is assumed to be full. In RS, this means that the initial sparse tensor is usually filled with zeros or the mean over each mode. Furthermore, CP-ALS-R directly inherits one of the major drawbacks of the CP decomposition: each rank constraint must be the same for each mode. When the dimension of the tensor are highly unbalanced, as in the case of e-commerce, where we may have millions of users, thousand of products but only few actions, the rank constraint is either too high for the action dimensions or too low for the other modes.

Tucker decomposition. One of the most popular approaches to perform Tucker decomposition is the generalization of SVD to n-order tensors, called **HOSVD** (Lathauwer, Moor, and Vandewalle 2000). We recall that the SVD of a matrix is the product of three matrices U, S and V, where U and V are orthonormal matrices of the left and right singulars values and S is the diagonal matrix of (ordered) singular

Algorithm 3 HOSVD for tensor factorization.

Input: dataset $\mathcal{D}$, desired rank $\{k_d\}_{d=1}^{3}$
Construct the initial (sparse) 3-order tensor $\mathbf{X}$ from $\mathcal{D}$
for $d \in \{1,2,3\}$ *(modes)* **do**
  Compute the matrix unfolding $X_{(d)}$
  Apply SVD on $X_{(d)}$ and select the k_d largest singular values

$$\hat{X}_{(d)} = U^{(d)} \tilde{S}_d V_d^\top$$

end for
Compute $\mathbf{W} \doteq \mathbf{X} \times_1 {U^{(1)}}^\top \times_2 {U^{(2)}}^\top \times_3 {U^{(3)}}^\top$
Return full tensor $\hat{\mathbf{X}} \doteq \mathbf{W} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$

values of X such that $X = USV^\top$. In RS, we preserve only the k largest singular value to get an approximation of X defined as $\hat{X} \doteq U\tilde{S}V^\top$, where $\tilde{S}$ is the truncated value of S where we replaced the $k+1, \cdots, \min(n,p)$ largest singular value by 0. In order to compute the Tucker decomposition, (Symeonidis, Nanopoulos, and Manolopoulos 2008) proposed to apply SVD on unfolded versions of the tensor as illustrated in Alg. 3. Similar to SVD for MF, this approach suffers from the fact that the missing values are implicitly translated into 0 entries or other arbitrary values. In many RS, this may significantly bias the result and return poor prediction performance.

Dealing with missing values. Both previous TF algorithms can be adapted to take into consideration missing values. Let $\mathbf{P}$ a binary 3-order tensor defined as (see (Acar et al. 2010), (Tomasi and Bro 2005))

$$\mathbf{P}_{ijk} = \begin{cases} 1, & \text{if } (i,j,k) \in \mathcal{D} \\ 0, & \text{else} \end{cases}$$

Using $\mathbf{P}$, (Acar et al. 2010) adapts the objective function of the CP decomposition as

$$\begin{aligned} f_{\mathbf{P}}(A,B,C) &= \sum_{i,j,k=1}^{I,J,K} \left(\mathbf{P}_{ijk} \left(\mathbf{X}_{ijk} - \sum_{r=1}^{R} A_{ir} B_{jr} C_{kr} \right) \right)^2 \\ &= ||\mathbf{Y} - \mathbf{Z}||^2 \end{aligned}$$

with $\mathbf{Y} = \mathbf{P} * \mathbf{X}$ and $\mathbf{Z} = \mathbf{P} * [A,B,C]$, where $*$ represent the element by element multiplication. They propose to use nonlinear conjugate gradient method after having vectorized the two tensors (by concatenating all the modes into a long vector). The resulting algorithm, called CP-WOPT, has the advantage to use a simple first-order optimization method. Nevertheless it suffers from different drawbacks. First, this approach can be really memory greedy since for each step of the algorithm we need to store very long vectors with all the known values while HOALS store only the values related to a particular user, or item, or action, and thus fail in large recommender systems. Indeed, they show experiments on a $64 \times 4392 \times 28$ tensor that is very small compared to state-of-the-art RSs like Amazon or Netflix. Second, the algorithm proceeds sequentially and it can not be parallelized. This is also a major drawback in RSs that deal with millions of user and entries that cannot be managed by a unique non-parallel method.

A similar approach can be followed for the Tucker decomposition by defining the objective

$$f_{\mathbf{P}}(\mathbf{W}, U, I, A) = \frac{1}{2} ||\mathbf{P} * (\mathbf{X} - \mathbf{W} \times_1 U \times_2 I \times_3 A)||_F^2$$

In order to learn the parameters, (Filipović and Jukić 2013) used nonlinear conjugate gradient with HOSVD initialization. The gradients of the objective function are (similar for I and A)

$$\begin{aligned} \nabla_U f_{\mathbf{P}} &= [\mathbf{P} * (\mathbf{W} \times_1 U \times_2 I \times_3 A - \mathbf{X})]_{(1)} \cdot \\ &\quad [(\mathbf{W} \times_2 I \times_3 A)_{(1)}]^\top \\ \nabla_{\mathbf{W}} f_{\mathbf{P}} &= \{\mathbf{P} * (\mathbf{W} \times_1 U \times_2 I \times_3 U - \mathbf{X})\} \times_1 \\ &\quad U^\top \times_2 I^\top \times_3 A^\top \end{aligned}$$

Although interesting, this method has several limitations when applied to RSs. First, it does not include any regularization in the objective function and this may lead to overfitting data. Second, as for the CP models, the algorithm cannot be parallelized and this hinders its applicability to large scale RSs.

Finally, when dealing with multi-relational data, the problem can be seen as a tensor completion problem where each slice is a squared matrix. (Nickel, Tresp, and Kriegel 2011) studied this specific problem by leveraging this particularity of the tensor in their RESCAL algorithm but we could not compare to them because of the squared matrix constraint that make no sense in our case.

Additional experiment

Dataset. For this second experiment to test the accuracy of our model, we use a real-world dataset:
Last.fm Dataset - 1K users (Celma 2009): it is formed of around 20M lines representing the track of an artist that a user is listening to at a given timestamp. We transformed the dataset in order to get the quadruplets *(user, artist, moment, value)*, where the *moment* represents the time of day clustered into three periods: *morning* from 6 a.m. to 2 p.m., *afternoon* from 2 p.m. to 10 p.m., and *night* from 10 p.m. to 6 a.m. The value counts the number of times the user listened to that artist in that period. The final dataset is composed of 510k quadruplets.
Results. The results are reported in Table 2. For this dataset (Table 2), tucker HOALS (implicit or not) obtains the best results with a $\overline{\rho}$ measure around 25%, two points better than HOSVD which is the next best model. Surprisingly, it is not the implicit version that achieve the best performance but the classic one, although both methods have really close performance (less than 0.6 point). Our CP method achieves relatively bad performance with a $\overline{\rho}$ measure around 32%. We think this can come from three reasons. The first one is due to the constraint of CP decomposition that imposes to have the same number of latent factors for each mode, which is a strong drawback for very imbalance tensor's dimension, as it is the case here. The second one is the low number of features for the users and items modes that we could not increase because of computational issues. Both these explanations are common to CP-ALS-R and CP-HOALS. Finally,

model	dim 1	dim 2	action	$\overline{\rho_a}$
Tucker-HOALS	200	200	morning	**23.65**
HOSVD	200	200	morning	24.58
imp Tucker-HOALS	200	200	morning	26.54
imp MI-ALS	50	-	morning	31.62
imp Tucker-HOALS	10	10	morning	30.40
CP-ALS-R	10	-	morning	31.76
imp CP-HOALS	10	-	morning	34.42
CP-HOALS	4	-	morning	36.98
MI-SVD	50	-	morning	34.72
MI-ALS	10	-	morning	40.53
Tucker-HOALS	200	200	afternoon	**23.62**
imp tucker-HOALS	200	200	afternoon	25.61
HOSVD	100	100	afternoon	27.26
imp Tucker-HOALS	10	10	afternoon	29.47
imp MI-ALS	50	-	afternoon	30.14
CP-ALS-R	10	-	afternoon	30.85
imp CP-HOALS	4	-	afternoon	32.57
CP-HOALS	5	-	afternoon	37.25
MI-SVD	50		afternoon	34.00
MI-ALS	10	-	afternoon	39.25
Tucker-HOALS	200	200	night	**24.81**
imp Tucker-HOALS	200	200	night	25.28
HOSVD	100	100	night	27.61
imp Tucker-HOALS	10	10	night	29.97
imp MI-ALS	50	-	night	31.49
CP-ALS-R	10	-	night	31.85
imp CP-HOALS	4	-	night	32.92
CP-HOALS	3	-	night	37.05
MI-SVD	10	-	night	35.17
MI-ALS	10	-	night	40.30

Table 2: $\overline{\rho_a}$ measure (rank-measure) on the Last.fm dataset ordered by decreasing performance. For MF and CP, *dim*1 refers to the common latent space. For TF, *dim*1 and *dim*2 refers to the number of latent vectors for the user and item space respectively. Due to memory limitations, we were not able to perform a CP-ALS-R in dimension greater than 10, we thus compare to HOALS in dimension 10 as well.

given the good performance of the HOSVD decomposition, we conjecture that treating missing values as 0 as done by the SVD composition models is not completely unreasonable in this case, since a user never listens to an artist if he/she never heard about or do not like her. The combination of the two precedents reasons and this observation explains the bad performance of CP-HOALS on this dataset that has difficulty to learn good user and item representation in a very low dimension with only the known entries. However, once again the good performance of the tensor factorization models prove the effectiveness of transfer learning in this context. Indeed the gain between best ALS and the best tucker HOALS is around 7 points.