# Recurrent Collaborative Filtering for Unifying General and Sequential Recommender

**Disheng Dong[1], Xiaolin Zheng[1]* , Ruixun Zhang[2], Yan Wang[3]**
[1] Zhejiang University
[2] MIT Laboratory for Financial Engineering
[3] Macquarie University
eson@zju.edu.cn, xlzheng@zju.edu.cn, zhangruixun@gmail.com, yan.wang@mq.edu.au

## Abstract

General recommender and sequential recommender are two applied modeling paradigms for recommendation tasks. General recommender focuses on modeling the general user preferences, ignoring the sequential patterns in user behaviors, whereas sequential recommender focuses on exploring the item-to-item sequential relations, failing to model the global user preferences. In addition, better recommendation performance has recently been achieved by adopting an approach to combining them. However, the existing approaches are unable to solve both tasks in a unified way and cannot capture the whole historical sequential information. In this paper, we propose a recommendation model named Recurrent Collaborative Filtering (RCF), which unifies both paradigms within a single model. Specifically, we combine recurrent neural network (the sequential recommender part) and matrix factorization model (the general recommender part) in a multi-task learning framework, where we perform joint optimization with shared model parameters enforcing the two parts to regularize each other. Furthermore, we empirically demonstrate on MovieLens and Netflix datasets that our model outperforms the state-of-the-art methods across the tasks of both sequential and general recommender.

## 1 Introduction

Recommendation systems play a crucial role in helping users identify interesting items in online services. Based on modeling different aspects of user behavioral data, two types of paradigms have been applied to recommendation tasks: general recommender and sequential recommender [Rendle et al., 2010; Wang et al., 2015].

general recommender has been widely studied in the last decade; its goal is to elicit the general preferences of users given their historical interactions with items, such as ratings and clicks. One of the most successful techniques in this setting is collaborative filtering based upon matrix factorization (MF), which learns user and item latent vectors to model the underlying user preferences over items [Koren et al., 2009].

On the other hand, a less explored yet more practical setting is sequential recommender, which views the interactions of a user as a time-ordered sequence and aims to predict which item the user will interact with next based on his/her previous interactions. A typical approach for sequential recommender is to model the sequential patterns into an item-to-item relational matrix, whereby items that are most related to the user's last interaction are recommended [Shani et al., 2005; Rendle et al., 2010; Grbovic et al., 2015]. Recently, recurrent neural network has shown to be a promising approach to modeling the item sequences and performing sequential predictions [Hidasi et al., 2016; Devooght and Bersini, 2017].

Both the aforementioned paradigms have strengths and weaknesses [Wang et al., 2015]. While general recommender discovers the general preferences of users, it discards the sequential information within user behaviors. While sequential recommender captures the item-to-item relations by exploring sequential patterns, it lacks the modeling of the user-item interactions, thus failing to capture the global user preferences. Therefore, general recommender is effective in performing long-term prediction which predicts items that user may interact with eventually, whereas sequential recommender excels at predicting users' short-term preferences.

A better way to build a recommendation model, therefore, is to unify the ideas of these two paradigms. For example, [Wang et al., 2015] improves the performance of sequential recommender by using global user representations in prediction. [Liang et al., 2016] facilitates matrix factorization based general recommender by additionally decomposing an item-to-item co-occurrence matrix which approximately encodes the sequential information. However, these approaches focus on using one recommendation paradigm to aid the other and hence are unable to deal with both tasks in a unified way. Moreover, they only capture local sequential features of the interaction sequences, thus losing the whole historical information contained in user behaviors.

To tackle the above problems, we propose a model named

*corresponding author

Recurrent Collaborative Filtering (RCF) that brings together the advantages of both general recommender and sequential recommender. Specifically, we employ a multi-task learning approach to jointly modeling the different aspects of the user behavioral data: (1) the general recommender part performs ordinary matrix factorization based collaborative filtering to capture the general tastes of users, and (2) the sequential recommender part utilizes recurrent neural network (RNN) to leverage the sequential item-to-item relations. We further optimize a joint loss with shared user and item vectors (embeddings) between the MF and RNN. In this way, the global user information is propagated to the sequential recommender method; in the meantime the sequential information is injected into the general recommender method.

To sum up, our main contributions are as follows:

- To exploit the sequential dependencies among items, we adapt recurrent neural network (RNN) to play the part of sequential recommender. Furthermore, we incorporate user vectors in sequential prediction to allow the RNN to attend to the global user preferences.
- We build a unified recommender by subsuming matrix factorization and recurrent neural network within a multi-task learning framework, which ties model parameters across the two methods to enforce information transfer, thus capturing both the sequential item relations and global user-item relations simultaneously.
- By conducting extensive experiments on Netflix and MovieLens datasets, we demonstrate that our model can achieve superior performance both as a general and a sequential recommender when compared to the state-of-the-art approaches.

# 2 Related Work

## 2.1 General Recommender

General Recommender recommends items through modeling the users' general tastes from their historical interactions. These items are not expected to be interacted with by a user in his/her next move, but may be interacted with eventually. One of the most effective approaches to this task is collaborative filtering based upon matrix factorization (MF), where user and item latent vectors are learned to discover the underlying user preferences [Koren et al., 2009]. MF-based methods can be further categorized according to two types of data that they deal with: explicit feedback and implicit feedback. Explicit feedback oriented methods formulate recommendations as a rating prediction problem, where users' ratings directly reflect their preferences [Koren et al., 2009]. However, explicit ratings are not always available and users more often interact with items through implicit feedback, e.g., clicks. Many of the recent MF-based methods tend to deal with implicit feedback by borrowing the idea of the Learning-to-Rank technique, which hinges on the design of an effective objective loss function to optimize [Karatzoglou et al., 2013].

General recommender is good at capturing the general features of user behaviors. Nevertheless, without considering the sequential patterns, general recommender can hardly adapt its recommendations directly to users' recent interactions.

## 2.2 Sequential Recommender

Sequential Recommender views the interactions of a user as a sequence and aims to predict which item the user will interact with next. A typical solution to this setting is to compute an item-to-item relational matrix, whereby items, which are most similar (nearest) to the last interacted one, are recommended to users. For example, Markov Chains based methods estimate an item-to-item transition probability matrix that predicts the probability of the next item given the last interaction of the user [Shani et al., 2005; Rendle et al., 2010]. Prod2Vec, inspired by word embedding technique [Mikolov et al., 2013], learns distributed item representations from the interaction sequences and uses them to compute a cosine similarity matrix [Grbovic et al., 2015].

Recently, Recurrent Neural Network (RNN), which is a state-of-the-art deep learning method for sequence modeling, has shown to be effective in capturing the sequential user behavioral patterns [Zhang et al., 2014; Hidasi et al., 2016; Devooght and Bersini, 2017]. Different from the previous methods, applying RNN to sequential recommender introduces the capability of modeling the whole historical interactions.

## 2.3 Unified Recommender

There are some recent attempts to build a unified recommender by considering both the sequential relations among items and general preferences of users. Factorized Personalized Markov Chains (FPMC) combines Markov Chains with matrix factorization to construct user-specific transition matrices, which are jointly decomposed to achieve better next-basket recommendation [Rendle et al., 2010]. Similarly, Hierarchical Representation Model (HRM) nonlinearly aggregates item vectors with global user vectors to form the context vectors that are predictive for the next-basket items [Wang et al., 2015; Yu et al., 2016]. Both FPMC and HRM focus on improving sequential recommender methods by involving the general user preferences. On the other hand, [Liang et al., 2016] considers developing a better matrix factorization based general recommender by exploiting a co-occurrence item-to-item matrix to capture the sequential patterns.

Our model belongs to this category, and the advantages of our method over the previous work are two-folds. First, unlike previous methods which use one recommendation paradigm to aid the other, our proposed model considers both paradigms at the same time, thus can make recommendations both as a general recommender and a sequential recommender. Second, we adapt recurrent neural network to model the sequential relations among items, and our model thereby is more effective in exploiting the whole historical sequential information.

# 3 Preliminaries

## 3.1 MF based Collaborative Filtering

This paper focuses on binary implicit feedback data and deals with a sparse $N \times M$ user-item interaction matrix $R$, where each entry $_{i,j} \in \{0, 1\}$ records whether user has interacted with item . The idea of matrix factorization is to learn effective user and item latent vectors (embeddings) from the

matrix $R$ to model user preferences. Let $^{|}{}_{i}$ and $_{j}$ represent the vector of user and item , respectively. We predict the probability that user interacts with item as:

$$\hat{}_{i,j} = \sigma(^{|}{}_{i} \cdot {}_{j} + {}_{i} + {}_{j}), \quad (1)$$

where $\sigma(\ )$ is the logistic sigmoid function. We include user and item bias terms $_{i}$ and $_{j}$ in the equation; we find that these terms are important to improve model performance.

However, training this model faces the challenge of lacking negative feedback, a.k.a., one-class problem [Pan , 2008]. While $_{i,j} = 1$ indicates positive preference, $_{i,j} = 0$ does not necessarily mean negative preference. To address this problem, we adopt the sampling method, which samples negative examples from the missing values [Pan , 2008]. Let $S^{+}$ denote the set of observed interactions in $R$ and $S^{-}$ denote the set of sampled negative interactions. We maximize the corresponding log-likelihood by equivalently minimizing the binary cross-entropy loss:

$$L_g = -\sum_{(i,j)\in S^{+}\cup S^{-}} {}_{i,j}\log\hat{}_{i,j} + (1 - {}_{i,j})\log(1 - \hat{}_{i,j}). \quad (2)$$

We call this method Basic-MF, which will then be used as the building block for our proposed model.

### 3.2 Sequence Modeling via RNN

Recently, Recurrent Neural Network (RNN) has become the state-of-the-art approach to modeling sequential data. Given a variable-length sequence $(\mathrm{x}^{(0)}, \mathrm{x}^{(1)}, \cdots, \mathrm{x}^{(T)})$, RNN maintains a hidden state vector $\mathrm{h}^{(t)}$ over time step , which can be considered as the cumulative summary of the sequence information till the time step , and is computed by:

$$\mathrm{h}^{(t)} = {}_{,}(\mathrm{h}^{(t-1)}, \mathrm{x}^{(t)}), \quad (3)$$

where $_{,}$ is a recurrent function that defines the central architecture of RNN.

Popular choices for $_{,}$ are the Long Short-Term Memory unit (LSTM) [Hochreiter and Schmidhuber, 1997] and Gate Recurrent Unit (GRU) [Cho , 2014], both adopting a gating mechanism to allow modeling long-term sequential dependencies. We use the GRU in our model, as it has been shown to outperform the LSTM in modeling the sequential item relations [Hidasi , 2016].

The GRU computes $\mathrm{h}^{(t)}$ as a linear interpolation between the previous state vector $\mathrm{h}^{(t-1)}$ and the candidate vector $\mathrm{c}^{(t)}$:

$$\mathrm{h}^{(t)} = (1 - \mathrm{o}^{(t)}) \odot \mathrm{h}^{(t-1)} + \mathrm{o}^{(t)} \odot \mathrm{c}^{(t)}, \quad (4)$$

where $\mathrm{o}^{(t)}$ acts as the update gate, and is given by:

$$\mathrm{o}^{(t)} = \sigma(W_o\mathrm{h}^{(t-1)} + U_o\mathrm{x}^{(t)} + {}_{o}). \quad (5)$$

Here, let $\odot$ denote the elementwise product, $W_{\cdot}$, $U_{\cdot}$ and $_{\cdot}$ denote the GRU parameters to be estimated, and denote the elementwise hyperbolic tangent function. The candidate vector $\mathrm{c}^{(t)}$ is computed as follows:

$$\mathrm{c}^{(t)} = (W_c(\mathrm{h}^{(t-1)} \odot \mathrm{r}^{(t)}) + U_c\mathrm{x}^{(t)} + {}_{c}), \quad (6)$$

where $\mathrm{r}^{(t)}$ acts as a reset gate that controls which parts of the previous hidden state to consider at the current time step, and is calculated by:

$$\mathrm{r}^{(t)} = \sigma(W_r\mathrm{h}^{(t-1)} + U_r\mathrm{x}^{(t)} + {}_{r}). \quad (7)$$

## 4 Recurrent Collaborative Filtering

### 4.1 RNN for Modeling User Interaction Sequences

With a slight abuse of notation, we use $\mathrm{x}_i^{(t)}$ to denote the item that user interacts with at the -th interaction. Note that $\mathrm{x}_i^{(t)}$ takes on value in $\{1, \cdots, M\}$, which correspond to the column indexes of the user-item interaction matrix $R$, and the items are embedded into vectors before we input them into the RNN defined in Eq.(3). Here, we consider the relative time steps, i.e., the first, second, etc., interaction with respect to a user, and let $(\mathrm{x}_i^{(0)}, \mathrm{x}_i^{(1)}, \cdots, \mathrm{x}_i^{(T_i)})$ denote the interaction sequence of user , where $T_i$ is the maximum time index of the sequence.

The goal of the sequential recommender is to predict which item a user will interact with next given his/her previous interactions. We achieve this by using the GRU-based RNN to model a conditional probability $(\mathrm{x}_i^{(t)}|\mathrm{x}_i^{(<t)})$, which defines a softmax distribution over the $M$ target items:

$$(\mathrm{x}_i^{(t)} = |\mathrm{x}_i^{(<t)}) = \frac{\exp({}_{j} \cdot \mathrm{h}_i^{(t)})}{\sum_{k=1}^{M}\exp({}_{k} \cdot \mathrm{h}_i^{(t)})}, \quad (8)$$

for all possible items $= 1, \cdots, M$. Here, $_{j}$ denotes the embedding vector of item and $\mathrm{h}_i^{(t)}$ is the hidden state vector of user at time step . We tie the output weights with the input item embeddings, which proves to be effective in learning the embedding vectors [Press and Wolf, 2016].

Directly maximizing the (log) probability defined in Eq.(8) over all users and time steps is impractical, because the cost of computing the full softmax is proportional to the total number of items, which is often extremely large. Therefore, we adopt the negative sampling technique [Mikolov , 2013] for efficient optimization, and minimize the following objective, which aims to distinguish the target items from the sampled items using logistic regression:

$$\mathcal{L}_s = -\sum_{i=1}^{N}\sum_{t=0}^{T_i}\{\log\sigma({}_{\mathrm{x}_i^{(t)}} \cdot \mathrm{h}_i^{(t)}) + \cdot E_{j'\sim P_m}[\log\sigma(-{}_{j'} \cdot \mathrm{h}_i^{(t)})]\}, \quad (9)$$

where is the number of negative samples, and $'$ is the sampled item drawn from a noise distribution $P_m$.

### 4.2 Joint Modeling

In order to simultaneously model the general user preferences over items and the sequential item relations, we propose a novel method named Recurrent Collaborative Filtering (RCF), where a multi-task learning approach is employed to combine the aforementioned Matrix Factorization (MF) with RNN. The architecture of RCF is illustrated in Figure 1. In particular, we jointly model the problem of sequential and general prediction, with the following strategies to allow information transfer between the two tasks:

(1) **Parameter sharing.** We constrain the item embeddings used in RNN to be the same as the item latent vectors used in matrix factorization. Formally, let $_{j} = {}_{j}$, for $= 1, \cdots, M$. In this way, we can capture the sequential patterns as well as the general user preferences into the item representations.

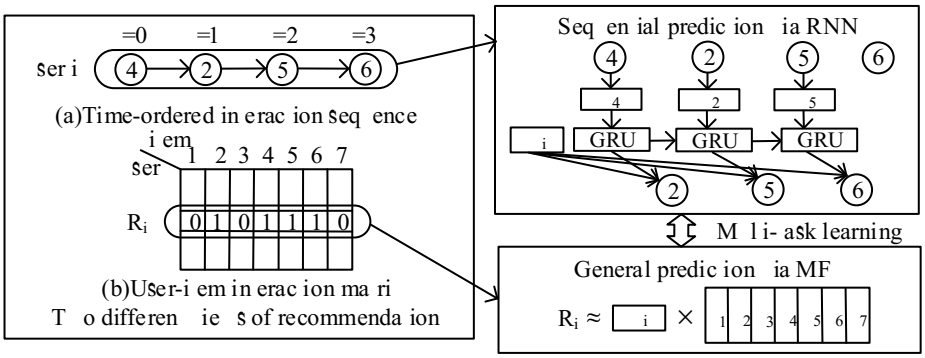



Figure 1: Illustration of RCF. The left-hand side shows two different views of recommendation task: sequential prediction and general prediction, with their corresponding methods in the right-hand side. The two methods are unified in a multi-task learning framework. The sequential part (RNN) incorporates global user vectors to attend to the general user preferences, while in the meantime the general part (MF) shares item latent vectors with the item embeddings used in RNN to capture the sequential patterns.

(2) **Additive attention.** We involve the user latent vectors to form new hidden state vectors that are predictive for the next item, allowing the RNN to attend to the global information of user preferences. That is, let $\tilde{\mathrm{h}}_i^{(t)} = \mathrm{h}_i^{(t)} + \mathrm{u}_i$, for $= 1, \cdots, N$. This additive approach is inspired by the popular Seq2Seq model, where the target sequence decoder uses attention mechanism to incorporate the context vectors that are obtained from the source sequence encoder [Bahdanau et al., 2014].

(3) **Multi-task loss.** To perform joint optimization, we adopt a convex combination of the losses defined in Eq.(2) and Eq.(9). Since both methods require sampling the negative items, we use a shared negative sampling process in the combined loss. This is reasonable in that such a popularity-based sampling strategy makes sense for both methods.

Therefore, considering all the aforementioned strategies, we obtain the following objective to minimize:

$$\begin{aligned}\mathcal{L} = -\sum_{i=1}^{N}\sum_{t=0}^{T_i}\{&(1-\alpha)\log\sigma(\ _{\mathrm{x}_i^{(t)}} \cdot \tilde{\mathrm{h}}_i^{(t)}) + \alpha\log\sigma(\hat{}_{i,\mathrm{x}_i^{(t)}}) \\ &+ \ \cdot E_{j'\sim P_m}[(1-\alpha)\log\sigma(-\ _{j'} \cdot \tilde{\mathrm{h}}_i^{(t)}) \\ &+ \frac{\alpha}{}\log(1-\sigma(\hat{}_{i,j'}))]\} + \gamma\Omega(\theta).\end{aligned} \tag{10}$$

Here, can be seen as the sampling ratio with respect to the number of positive observations. The trade-off parameter $\alpha \in [0, 1]$ balances the relative importance of the two tasks to the overall objective. $\theta$ represents the collection of model parameters including the user and item vectors used in MF and the weight matrices used in RNN. $\Omega(\theta)$ is a $\ell_2$-regularization term with its effect controlled by the hyper-parameter $\gamma$.

### 4.3 Training and Prediction

Given the interaction sequence $(\mathrm{x}_i^{(0)}, \mathrm{x}_i^{(1)} \cdots, \mathrm{x}_i^{(T_i)})$ of user , training sequences are generated as: $(\mathrm{x}_i^{(0)}, \mathrm{x}_i^{(1)})$, $(\mathrm{x}_i^{(0)}, \mathrm{x}_i^{(1)}, \mathrm{x}_i^{(2)}), \cdots, (\mathrm{x}_i^{(0)}, \mathrm{x}_i^{(1)} \cdots, \mathrm{x}_i^{(T_i)})$, where the last element of each sequence is considered as the label to be predicted from the previous elements. This approach can avoid leaking future information and has shown promising results in live testing [Covington et al., 2016]. We adopt embedding dropout [Tan et al., 2016] to make the model more robust against noisy interactions. We use the Back Propagation Through Time (BPTT) algorithm to train our proposed RCF and experiment with adagrad [Duchi et al., 2011].

After training, the model performs Top-$K$ recommendation for each user by ranking the user's preference scores over the test (unobserved) items. RCF is capable of handling the two recommendation paradigms.

As a sequential recommender, RCF can take $\tilde{\mathrm{h}}_i^{(t)}$ as the "profile" vector of user at time step . We use $\hat{}_{i,j}^{(t)}$ to represent the preference score of user over item at time step ; and compute $\hat{}_{i,j}^{(t)}$ by:

$$\hat{}_{i,j}^{(t)} = \tilde{\mathrm{h}}_i^{(t)} \cdot \ _j, \tag{11}$$

It is worth noting that $\tilde{\mathrm{h}}_i^{(t)}$ is updated with the interaction $_i^t$, which provides RCF with the potential ability for dynamic recommendation.

As a general recommender, RCF can represent user as the vector $\mathrm{u}_i$ and predict the preference score over item by:

$$\hat{}_{i,j} = \mathrm{u}_i \cdot \ _j + \ _i + \ _j. \tag{12}$$

Here, we omit the logistic sigmoid transformation for clarity, since it doesn't influence the relative rank of users' preferences over items.

## 5 Experiments

### 5.1 Setup

**Datasets**

We conduct experiments on the two widely used datasets: MovieLens and Netflix. The datasets are preprocessed to ensure that each user has a minimum of 20 ratings. Following [Rendle et al., 2009], we transform the ratings into binary feedback data, where each entry is marked as 0/1 indicating whether the user has interacted with (rated) the item. Note that both datasets are time stamped, with which we can produce a time-ordered interaction sequence for each user. The statistics of the final datasets are summarized in Table 1.

| Dataset | #Interactions | #Users | #Items | Sparsity |
|---|---|---|---|---|
| MovieLens | 1,000,209 | 6,040 | 3,706 | 95.53% |
| Netflix | 99,884,940 | 429,584 | 17,770 | 98.69% |

Table 1: Statistics of the evaluation datasets.

**Evaluation Scheme**

Since our proposed model can perform the tasks of both sequential and general recommender, for a fair comparison with the previous methods, we consider the following two popular protocols for evaluating the corresponding recommenders:

**Leave-one-out Prediction**. This protocol is used for evaluating the performance of sequential recommenders [Wang et al., 2015]. In this protocol, we hold out the penultimate element of the time-ordered interaction sequence for each user as validation data; we leave the last element out as test data and train the final model on the remaining data with the validation data included.

**General Prediction**. This protocol evaluates the performance of general recommenders. Following [Liang et al., 2016], we sort all user-item interactions in chronological order, training on the first 80% of the interactions and holding out the latest 20% for testing. We randomly select 10% of the interactions from the training set as validation data.

We assess against the test data a ranking list of top-$K$ items that are recommended for each user. Two popular evaluation metrics are adopted: Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) [Wang et al., 2015]. Due to page limitation, we report the results when $K$ is set to 10; we found out that different values of $K$ had a slight impact on the model comparison results.

**Parameter Settings**

In the experiments, we used the validation data to find the optimal hyper-parameters. The dimension of latent vectors (and hidden vectors in the GRU unit) was fixed as 50. To form a training batch, we randomly sampled a number of user-item interactions along with their sequential prefixes; for each interaction, we drawn negative items from the unobserved interactions from the noise distribution $P_m$, for which we adopted the unigram distribution over items. We truncated the BPTT algorithm using a fixed window of 20 time steps; sequences that are shorter than 20 items were padded with zeros for simplicity. The initial hidden state vectors for the training sequences were initialized with zeros. We used a single recurrent (GRU) layer in our model; we found that additional layers caused overfitting, thus resulting in worse performance. The regularization parameter $\gamma$ was set to $10^{-4}$ for all datasets. We report other best performing hyper-parameters in Table 2.

## 5.2 Leave-one-out Prediction

We compare the performance of RCF (as a sequential recommender) with the following sequential recommender methods using the leave-one-out prediction protocol:

- **Prod2Vec**: Prod2Vec has demonstrated to be a promising approach for next-item prediction [Grbovic et al., 2015].
- **GRU-TOP1**: This is a state-of-the-art method for session-based recommendation and can be easily adapted to our task [Hidasi et al., 2016].
- **HRM**: Hierarchical Representation Model (HRM) [Wang et al., 2015] is a state-of-the-art method for next-item recommendation.
- **RCF-vec**: We use the item embeddings learned by RCF and perform nearest neighbor based recommendation.

By design, RNN-based models can account for arbitrarily distant historical interactions. However, since a user's intent is most related to the recent behaviors, we set the number of look-back interactions to 10 for GRU-TOP1 and RCF. The performance results of all the above-mentioned methods are shown in Table 3. We summarize key observations below.

Firstly, RCF-vec outperforms Prod2Vec, implying RCF's ability in learning more effective item embeddings than Prod2Vec. On the other hand, the performance improvement of RCF over RCF-vec indicates the necessity of keeping track of more historical information through the recurrent function.

Secondly, RCF delivers better performance than GRU-TOP1 across both datasets. The reason is that RCF involves modeling the general user preferences rather than purely relying on the item-to-item relations.

Thirdly, RCF shows superior performance over HRM. Since both methods utilize the general user preferences and the sequential information, we attribute the performance difference to the consideration of different numbers of historical interactions (HRM only considers the last interaction).

To further understand the impact of the historical information, we show the performance changes of RCF with respect to the number of look-back items in Figure 2. The number of look-back items is set to 10 in the validation phase. However, we also tried using different numbers of look-back items for hyper-parameter optimization based on the validation data and found that the results were broadly the same when the number of look-back items was set between 6 and 12. Note that considering zero look-back items reduces the model to a plain general recommender, that is, Basic-MF.

From the figure, we can see that the performance of RCF improves gradually initially as the number of look-back items increases, which, again, shows the necessity of accounting for more than just the last interaction in sequential recommendation. However, further increase in the look-back items hurts the model performance. This suggests that considering long historical interactions introduces the noise that influences the prediction accuracy; notwithstanding RCF is relatively robust to noisy interactions, which can be inferred from the stable performance of RCF when the number of look-back items is in the range of $6 \sim 12$ on both datasets.

## 5.3 General Prediction

We compare the performance of RCF (as a general recommender) to the following baselines using the general prediction protocol:

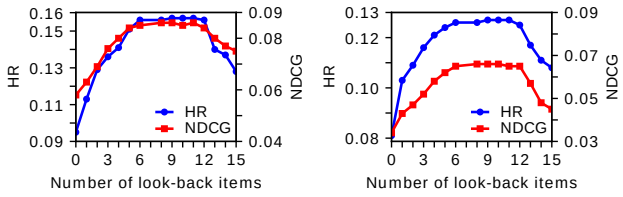
- **BPR**: This is a highly competitive baseline for binary implicit feedback [Rendle et al., 2009].
- **Basic-MF**: As discussed previously, this method is the building block of our proposed model.

| Dataset | Protocol | $\alpha$ | Sampling ratio | Batch | Dropout | Learning rate | Momentum |
|---|---|---|---|---|---|---|---|
| MovieLens | Leave-one-out | 0.2 | 4 | 128 | 0.1 | 0.1 | 0.1 |
| MovieLens | General | 0.8 | 4 | 128 | 0.1 | 0.1 | 0.1 |
| Netflix | Leave-one-out | 0.2 | 6 | 256 | 0.3 | 0.05 | 0.3 |
| Netflix | General | 0.8 | 6 | 256 | 0.3 | 0.05 | 0.3 |

Table 2: Best parametrization for different datasets/protocols.

| | MovieLens | | Netflix | |
|---|---|---|---|---|
| Method | HR | NDCG | HR | NDCG |
| Prod2Vec | 0.102 | 0.060 | 0.085 | 0.036 |
| RCF-vec (ours) | 0.116 | 0.066 | 0.092 | 0.041 |
| GRU-TOP1 | 0.138 | 0.074 | 0.113 | 0.048 |
| HRM | 0.142 | 0.076 | 0.114 | 0.051 |
| RCF (ours) | **0.157** | **0.086** | **0.126** | **0.065** |

Table 3: Performance comparison using leave-one-out prediction protocol.




(a) MovieLens (b) Netflix

Figure 2: Performance of RCF (as a sequential recommender) w.r.t. the number of look-back items using leave-one-out prediction protocol.

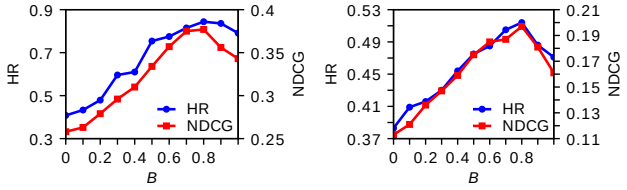
- **CoFactor**: This is a state-of-the-art general recommender, which considers the sequential relations among items [Liang et al., 2016].

Table 4 shows the performance of the compared methods using the general prediction protocol. We observe that CoFactor and RCF consistently outperform BPR and Basic-MF across the two metrics on both datasets. This is not surprising because both CoFactor and RCF make use of the additional information inherent in the item-to-item sequential relations. Nonetheless, CoFactor simply utilizes an item co-occurrence matrix to approximately model the sequential patterns, thus failing to fully capture the sequential information. We can see that RCF achieves better performance than CoFactor (the relative improvement on MovieLens and Netflix is 3.2% and 5.9%, respectively), which demonstrates that RCF learns more effective user and item representations for modeling the general user preferences than CoFactor.

Figure 3 shows the performance changes of RCF with respect to the value of trade-off parameter $\alpha$. Note that we fix other hyper-parameters as in Table 2. From the figure, we can see that the performance of RCF first increases then decreases as the value of $\alpha$ increases from 0 to 1, which implies that RCF controls its performance as a general recommender through balancing the value of $\alpha$. When $\alpha = 1$, RCF reduces to plain Basic-MF and completely discards the sequential information, which, however, does not give the best performance. The results further demonstrate that it is important for general recommender to incorporate the information of the sequential item-to-item relations.

| | MovieLens | | Netflix | |
|---|---|---|---|---|
| Method | HR | NDCG | HR | NDCG |
| BPR | 0.787 | 0.308 | 0.462 | 0.152 |
| Basic-MF | 0.791 | 0.343 | 0.471 | 0.161 |
| CoFactor | 0.818 | 0.361 | 0.485 | 0.182 |
| RCF (ours) | **0.844** | **0.377** | **0.514** | **0.197** |

Table 4: Performance comparison using general prediction protocol.




(a) MovieLens (b) Netflix

Figure 3: Performance of RCF (as a general recommender) w.r.t. the value of the trade-off parameter $\alpha$ using general prediction protocol.

## 6 Conclusion and Future Work

In this paper, we have proposed a recommendation method named Recurrent Collaborative Filtering (RCF) that unifies the general and sequential recommender. RCF subsumes matrix factorization (MF) and recurrent neural network (RNN) in a multi-task learning framework. In particular, we adapt RNN to model the sequential user behaviors with the aim of capturing the whole historical information. Then we optimize a joint loss for MF and RNN, both of which share model parameters in each other. By design, RCF unites the merits of general recommender (MF) and sequential recommender (RNN). Empirically, we show on two real world datasets that RCF can deliver superior performance both as a general and a sequential recommender.

For the future work, we plan to encode other types of side information into RCF to make it more applicable to real scenarios. Moreover, it will be computationally appealing to apply convolutional neural network (CNN), which is more parallelizable than RNN, in our model for sequence modeling.

## Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (No.U1509221), the National Key Technology R&D Program (2015BAH07F01), the Zhejiang Province key R&D program (No.2017C03044).

## References

[Bahdanau et al., 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. , abs/1409.0473, 2014.

[Cho et al., 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. , 2014.

[Covington et al., 2016] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In , pages 191–198, 2016.

[Devooght and Bersini, 2017] Robin Devooght and Hugues Bersini. Long and short-term recommendations with recurrent neural networks. In , pages 13–21, 2017.

[Duchi et al., 2011] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. , pages 2121–2159, 2011.

[Grbovic et al., 2015] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In , pages 1809–1818, 2015.

[Hidasi et al., 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In , 2016.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. , 9:1735–1780, 1997.

[Karatzoglou et al., 2013] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. Learning to rank for recommender systems. In , pages 493–494, 2013.

[Koren et al., 2009] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. , 42:30–37, 2009.

[Liang et al., 2016] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M. Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In , pages 59–66, 2016.

[Mikolov et al., 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In , pages 3111–3119. 2013.

[Pan et al., 2008] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In , pages 502–511, 2008.

[Press and Wolf, 2016] Ofir Press and Lior Wolf. Using the output embedding to improve language models. , 2016.

[Rendle et al., 2009] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In , pages 452–461, 2009.

[Rendle et al., 2010] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In , pages 811–820, 2010.

[Shani et al., 2005] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. , pages 1265–1295, 2005.

[Tan et al., 2016] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In , pages 17–22, 2016.

[Wang et al., 2015] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for next-basket recommendation. In , pages 403–412, 2015.

[Yu et al., 2016] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. A dynamic recurrent model for next basket recommendation. In , pages 729–732, 2016.

[Zhang et al., 2014] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. Sequential click prediction for sponsored search with recurrent neural networks. In , pages 1369–1375, 2014.