Knowledge-Based Systems 209 (2020) 106478

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

DeepRank: Learning to rank with neural networks for recommendation

Ming Chen, Xiuze Zhou *

Hithink RoyalFlush Information Network Co., Ltd., Hangzhou, China

ARTICLE INFO

Article history:
Received 31 May 2020
Received in revised form 14 September 2020
Accepted 18 September 2020
Available online 23 September 2020

Keywords:
Recommender systems
Deep learning
Ranking learning
Neural networks
Collaborative filtering

ABSTRACT

Although, widely applied deep learning models show promising performance in recommender systems, little effort has been devoted to exploring ranking learning in recommender systems. It is important to generate a high quality ranking list for recommender systems, whose ultimate goal is to recommend a ranked list of items for users. Also, the latent features learned from Matrix Factorization (MF) based methods do not take into consideration any deep interactions between the latent features; therefore, they are insufficient to capture user–item latent structures. To address these problems, we propose a novel model, DeepRank, which uses neural networks to improve personalized ranking quality for Collaborative Filtering (CF). This is a general architecture that can not only be easily extended to further research and applications, but also be simplified for pair-wise learning to rank. Finally, we perform extensive experiments on three data sets. Results demonstrate that our proposed models significantly outperform the state-of-the-art approaches. Our projects are available at: https://github.com/XiuzeZhou/deeprank.

1. Introduction

Recommender systems have been successfully applied to many fields, such as e-commerce, music, and news. These systems model users' preferences and predict the best-suited services or products for users to help them discover useful information from a plethora of options [1–4]. In practice, recommender systems aim at recommending items that users may be interested in a ranking list. Therefore, the effects from ranking-oriented methods are more suitable than the accuracy performance from rating prediction methods for recommender systems [5,6].

To achieve a better quality of recommender systems and improve their ranking performance, various approaches have been proposed. For example, Rendle et al. [7] proposed a Bayesian ranking framework, which compares ordered pairs of items to decide which is preferred over another in the recommendation list. Park et al. [8] adopted some external information, such as user profiles and item contents, to solve the cold-start problem for pair-wise preference regression. Shi et al. [9] proposed a method, ListRank-MF, which combined a list-wise ranking method with Matrix Factorization (MF) for Collaborative Filtering (CF). However, one major limitation of those traditional solutions is that they are unable to fully capture complex structures and deeper information from user–item interactions.

Recently, all kinds of deep learning models have achieved remarkable success in various fields, such as Computer Vision (CV), speech recognition, and Natural Language Processing (NLP). These deep learning models are also widely applied to recommender systems to improve the quality of recommendation. For example, Kim et al. [10] applied Convolutional Neural Networks (CNN) to reviews to obtain contextual information and then combined it with MF for recommendation. Wang et al. [11] presented a collaborative deep learning method, which uses a Stacked Denoising Auto-encoder (SDA) to obtain information from reviews to alleviate the data sparse problem of CF. McAuley et al. [12] used the latent features learned from images by neural networks for the style and appearance of products to catch visual relationships between the products. However, these methods have some shortcomings. First, they cannot be applied to fields with little or no additional information, such as textual and image information [13]. Second, capturing and processing that auxiliary data requires extensive time and effort.

Although many methods based on MF have achieved good performance on recommendation, they cannot effectively learn user and item representations, which leads them to have poor ability to capture complicated and deeper information about the interaction between users and items. To solve this problem, and inspired by the great success of deep learning methods applied to ranking learning, we propose DeepRank, a list-wise ranking method with neural networks. Point-wise methods, rather than focusing on the personalized ranking of a set of items, focus only on predicting an accurate rating value of an item. In practice,

The code (and data) in this article has been certified as Reproducible by Code Ocean: https://help.codeocean.com/en/articles/1120151-code-ocean-s-verification-process-for-computational-reproducibility. More information on the Reproducibility Badge Initiative is available at www.elsevier.com/locate/knosys.

* Corresponding author.
E-mail addresses: chm@zju.edu.cn (M. Chen), zhouxiuze@foxmail.com (X. Zhou).

https://doi.org/10.1016/j.knosys.2020.106478
0950-7051/

users tend to pay more attention to the ranking order of an item than to its accurate ratings. For example, when wanting to see a movie online, a user often cares less about its rating and chooses the movie at the top of the recommended list. Rather than predicting a rating, the goal of DeepRank is to use predicted scores to rank the position of an item.

Then, we reduce our model from top-n list-wise to a simpler structure: top-one list-wise, and then to the simplest structure: a pair-wise learning method, which is one of the most popular ranking-oriented methods in recommender systems. Pair-wise methods, in which each user is represented as a set of pair-wise preferences over items, help users understand their preferences more than do point-wise methods [14,15].

Finally, the users' preference features and characteristic features of the items are not directly related. Therefore, to further improve the generation performance of our model, we set the latent features of users and items at different sized dimensions. In most recommendation approaches, the interaction between user latent features and item latent features is used to predict the rating or ranking score [16,17]. But, the difference in the number of latent features between them is rarely taken into account. To the best of our knowledge, this is the first work that aims at setting the number of latent features for users and items at different values.

We demonstrate that our proposed DeepRank has several attractive advantages:

(1) When using DeepRank to make predictions, it achieves better ranking performance. To the best of our knowledge, this is the first list-wise work based on neural network to rank learning;
(2) It has a simple and flexible structure, which can be simplified from top-n list-wise to top-one list-wise and pair-wise ranking learning for efficiency;
(3) This is the first time the effect of setting the number of latent features for users and items at different values has been investigated.

The rest of the paper is organized as follows: Section 2 briefly reviews the background and some related work. Section 3 presents our proposed models in detail. Section 4 describes experimental results for several data sets to show the performance of our models. Section 5 gives the conclusion and provides future directions.

2. Related work

In this section, we briefly introduce some background information and related works. First, we introduce some ranking-oriented approaches and deep learning models for recommendations. Then, we introduce ListRank-MF, which inspired us to propose our method.

2.1. Ranking-oriented methods

MF is one of the most effective methods to deal with various recommendations [18]. The key idea of MF is to learn a latent feature with low dimension to represent users and items. Many MF approaches focus on prediction accuracy, but a low prediction error cannot guarantee a high quality of recommendation [9,19]. Thus, many ranking-oriented MF methods have been proposed for top-n tasks. For example, Weimer et al. [20], instead of rating, developed MF by minimizing a convex upper bound of the Normalized Discounted Cumulative Gain (NDCG) loss for optimizing ranking. Wu et al. [21] designed a list-wise method, which maximizes the likelihood of a permutation model for building use-specific ranking.

Although those methods perform well, they encounter a critical problem: they lack nonlinear and complicated representations about users and items [22–24]. Traditional MF methods consider only the linear interaction between users and items, without exploring the nonlinear and more complicated interaction between them.

In recent years, deep learning models have been widely and successfully applied to CV and NLP [25]. Because deep learning models have a powerful ability to learn complicated and nonlinear representations by their hidden layers, many researchers resort to deep learning models to develop recommender systems [26]. For example, He et al. [16] transformed traditional MF to neural MF (NeuMF), which uses a neural network architecture to obtain latent features and achieves state-of-the-art performance. Wu et al. [5], by integrating user-specific bias into a Denoising Auto-Encoder (DAE) to obtain latent properties of items, proposed an improved method: Collaborative Denoising Auto-Encoder (CDAE). Zhang et al. [27] proposed a model, DeepCF, which uses deep neural networks to for implicit user–item coupling learning.

Although many deep learning models have been proposed for recommendation, little effort has been devoted to explore the ranking learning in recommender systems. Pair-wise and list-wise methods are the most important ranking methods in the field of machine learning. Pair-wise methods generate a personalized recommendation list for users and build the users' pair-wise preference between items [6,13,28]. And pair-wise methods capture users' preferences according to their pair-wise behaviors, where a set of preferences for each pair of items is used to represent each user [1]. For example, Burges et al. [29] proposed RankNet, a pair-wise method with neural networks which calculates the probability about user's pair-wise preferences. Rendle et al. [7] proposed Bayesian Personalized Ranking (BPR), the most popular method to deal with pair-wise ranking for implicit feedback.

However, methods based on pair-wise have two major problems: (1) They consider only the relative order of two items, not the position of the items in the recommended list. The items at the top of the recommended list are more important than those at the bottom. If the items at the top are misjudged, the ranking cost is significantly higher than for the items at the bottom. (2) The number of relevant items varies greatly among different users. After being converted into item pairs, some users have hundreds of corresponding item pairs; whereas, others have only dozens of corresponding item pairs, which makes it difficult to evaluate the performance of the models.

2.2. ListRank-MF

To alleviate the problems caused by pair-wise approaches, many list-wise approaches proposed. We introduce a traditional ranking-oriented method, the list-wise learning to rank with MF (ListRank-MF), which is the most relevant to our model. To obtain top-one probability, Shi et al. [9] proposed ListRank-MF, a list-wise probabilistic MF method that optimizes the cross entropy between the distribution of the observed and predicted ratings.

ListRank-MF seeks to maximize the top-one probability of items in a user's ranking list. The function about top-one probability proposed by Cao et al. in [30], and has showed a good performance for ranking. The top-one probability that an item, i, rated by user u in his list, l_u, is defined as follows:

$$P_{l_u}(r_{ui}) = \frac{g(r_{ui})}{\sum_{k=1}^{K} g(r_{uk})}, \tag{1}$$

where r_{ui} denotes the rating of item i given by user u; K denotes the number of items in the list l_u, i.e. the length of l_u; $g(\cdot)$ denotes a monotonically increasing and strictly positive function.

Then, ListRank-MF uses cross-entropy to calculate the top-one probability in observed ratings. The loss function of ListRank-MF, which measures the distance between the true list and the predicted list from the ranking model, is defined as follows:

$$L(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{K} P_{l_u}(r_{ui}) \, log P_{l_u}\left(f\left(\boldsymbol{p}_u^T \boldsymbol{q}_i\right)\right) + \frac{\lambda}{2}(\|\boldsymbol{p}\|_F^2 + \|\boldsymbol{q}\|_F^2), \tag{2}$$

where N and M denote the number of users and items in data, respectively; λ denotes a regularization parameter; $\|\cdot\|_F^2$ denotes the Fibonacci-norm; $\boldsymbol{p}$ and $\boldsymbol{q}$ denote the latent features, which represent users and items, respectively; and $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ represent the feature vector of user u and item i, respectively; $f(\cdot)$ is a logistic function, whose purpose is to bound the range of $\boldsymbol{p}_u^T \boldsymbol{q}_i$, defined as follows:

$$f(x) = \frac{1}{1 + exp(-x)}. \tag{3}$$

But ListRank-MF has two major problems: First, it is modeled on the traditional MF method, which uses the most common way, the inner product, to model the interaction between users and items. This way prevents ListRank-MF from obtaining the nonlinear representation. Second, it only maximizes the top-one rather than top-K probability of items in a user's ranking list, which loses a lot of ranking information.

3. Methods

In this section, we introduce our proposed model: DeepRank, which is a top-n list-wise model for implicit feedback. First, we discuss the problem definition and the notations used throughout the paper. Then, we introduce our model in detail. Finally, we show that our model can be simplified to a top-one list-wise, and further to a pair-wise method for ranking.

3.1. Problem definition and notations

Given a user–item rating matrix, $\boldsymbol{R}$, with N users and M items, an interaction matrix, $\boldsymbol{Y}$, is defined as follows:

$$y_{ui} = \begin{cases} 1, & \text{if } r_{ui} > 0 \\ 0, & \text{else} \end{cases}, \tag{4}$$

where $y_{ui} \in \boldsymbol{Y}$, and $r_{ui} \in \boldsymbol{R}$ denotes the rating of item i given by user u.

The main goal of DeepRank is to predict the unrated order of items based on their final scores. The objective function is defined as follows:

$$L = f(y, \hat{y}) + \lambda \Omega(\Theta), \tag{5}$$

where $f(\cdot)$ is the loss function of the model; y and $\hat{y}$ are the true and the predicted labels of instances, respectively; $\Omega(\Theta)$ is the regularization used to reduce over-fitting.

3.2. Neural networks for ranking

We focus on ranking learning for top-n recommendation performance, which is more meaningful for real recommender systems. To elaborate on the DeepRank model, we employ a deep learning framework for list-wise learning for ranking. The graphical representation of our proposed model is shown in Fig. 1. Our model consists of four layers: input, embedding, hidden, and predictive.

Input and Embedding Layers. A fully connected layer projects each sparse feature to a dense vector representation. The function of the embedding layer in a neural network is to convert users and items to low-dimensional space and use dense vectors to represent them. The preference features of users and attributes of items are not directly related; therefore, it makes more sense that their embedding should have dimensions of different sizes. As far as we know, this is the first time that the embedding of users and items has been set to different sized dimensions to improve the generalization performance of neural networks.

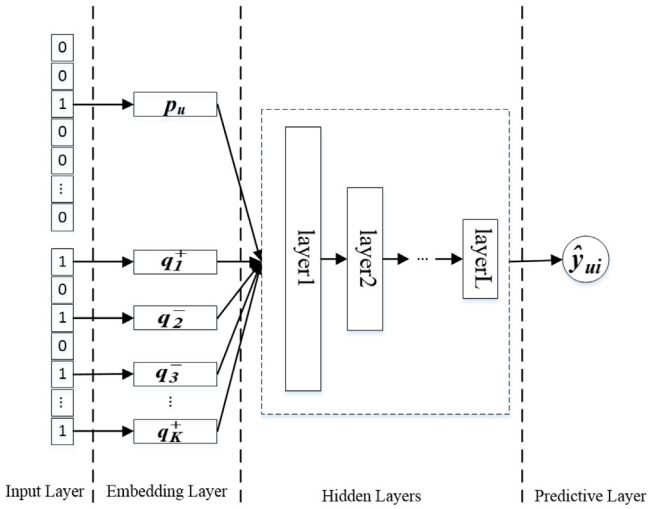


Fig. 1. Network Architecture for DeepRank.

Embedding layer maps the sparse features to dense features, the embedding vectors from lookup-tables, defined as follows:

$$\boldsymbol{p}_u = f_{lookup}(u), \tag{6}$$

$$\boldsymbol{q}_i = f_{lookup}(i), \tag{7}$$

where $\boldsymbol{p}_u \in \mathbb{R}^{1 \times d_u}$ and $\boldsymbol{q}_i \in \mathbb{R}^{1 \times d_i}$ denote the latent representations (embedding vectors) of user u and item i, respectively, and d_u and d_i denote their respective dimension sizes.

There are two reasons for setting different latent features for users and items:

On the one hand, users and items are independent of each other; therefore, their number of latent features is not necessarily identical. In fact, as the number of items interacted with users increases, so does interest, leading to an increase in the latent features of the users. However, the attributes of the items are relatively stable; thus, the number of their features does not change significantly.

On the other hand, as is well-known, neural network training is time-consuming. Therefore, a sound strategy is to use a pre-trained model to accelerate the training. Over time, users' preferences change and many new interactions are generated, resulting in a change in the number of the latent features of users. Then, to speed up the training, rather than re-training the model from random initialization, additional features are added to the user embeddings of the pre-trained model.

In the list of K items for training, there are K^+ positive instances, and $(K - K^+)$ negative instances sampled from user u. $\boldsymbol{q}_i^+$ and $\boldsymbol{q}_i^-$ denote the embeddings from positive and negative instances, respectively.

Hidden Layers. Hidden layers are stacks of several fully connected layers: the dense vectors of a user and K different items from their embedding layers. The function of hidden layers is to jointly encode user preferences and item attributes and capture the nonlinear interactions between them. The Rectified Linear Unit (ReLU), selected as the activation function for hidden layers, demonstrates good performance in neural networks [31]. We used only three hidden layers in DeepRank to not only simplify

the model and reduce the difficulty of tuning parameters, but also achieve better generalization. Also, our model can be extended easily to deeper networks. With $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ as inputs, the interaction between them is defined as follows:

$$\boldsymbol{h}_1 = f(\boldsymbol{p}_u, \boldsymbol{q}_i), \tag{8}$$

$$\boldsymbol{h}_2 = f_1(\boldsymbol{w}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2), \tag{9}$$

$$\ldots$$

$$\boldsymbol{h}_l = f_l(\boldsymbol{w}_l^T \boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \tag{10}$$

$$\boldsymbol{h}_L = f_L(\boldsymbol{w}_L^T \boldsymbol{h}_{L-1} + \boldsymbol{b}_L). \tag{11}$$

where $f(\cdot)$ denotes the interaction function between $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$, such as concatenation, and inner product; $\boldsymbol{h}_l$, $\boldsymbol{w}_l$, and $\boldsymbol{b}_l$ denote the output, weights, and bias of hidden layer l, respectively; $f_l(\cdot)$ denotes the activation function; and L denotes the number of hidden layers.

Predictive Layer. The function of the predictive layer is to map the results from the final hidden layer to the probability, $\hat{y}_{ui}$. Then the prediction score is formulated as follows:

$$\hat{y}_{ui} = softmax(x_{ui}), \tag{12}$$

where x_{ui} denotes the output from the final hidden layer. We chose the softmax function to map the results from the hidden layer to prediction. The probabilities $\hat{y}_{ui}$ that item i ranks at the top-one for user u are defined as follows:

$$\hat{y}_{ui} = \frac{e^{x_{ui}}}{\sum_{k=1}^{K} e^{x_{uk}}}, \tag{13}$$

where K denotes the number of items in the list l_u.

In this paper, we focus on top-n model, and the probability of items in the user list is defined as:

$$P_{l_u}(S(i_1, i_2, \ldots, i_K)) = \prod_{j \in l_u^+} \hat{y}_{uj} \prod_{k \in l_u^-} (1 - \hat{y}_{uk}), \tag{14}$$

where $S(i_1, i_2, \ldots, i_K)$ denotes the set of all items in list l_u; The sets of items rated and unrated by user u in the recommended list l_u, are denoted by l_u^+ and l_u^-, respectively.

Then, loss is evaluated by cross entropy, which used to measure the distribution between the true list and the predicted list from the ranking model, is defined as follows:

$$f(y, \hat{y}) = -\sum_{u=1}^{N} \left(\sum_{i \in l_u^+} log\hat{y}_{ui} + \sum_{j \in l_u^-} log(1 - \hat{y}_{uj}) \right). \tag{15}$$

where y_{ui} and $\hat{y}_{ui}$ denote the true and predicted probability that an item, i, rated by the user, u, is in all items from the list, respectively.

Finally, the regularization is defined as follows:

$$\Omega(\Theta) = \sum_{l=1}^{L} \|\boldsymbol{w}_l\|_F^2 + \sum_{u=1}^{N} \|\boldsymbol{p}_u\|_F^2 + \sum_{i=1}^{M} \|\boldsymbol{q}_i\|_F^2. \tag{16}$$

3.3. Pair-wise DeepRank

Pair-wise methods model the relative ordering from each pair of items to make predictions. Pair-wise methods assume that a user prefers observed rather than unobserved items. Thus, for training, pair-wise methods construct the relationship between positive and negative instances. So, its objective function is defined as follows:

$$f(y, \hat{y}) = -\sum_{u=1}^{N} \left(\sum_{i \in l_u^+} log\hat{y}_{ui} + \sum_{j \in l_u^-} log(1 - \hat{y}_{uj}) \right), \tag{17}$$

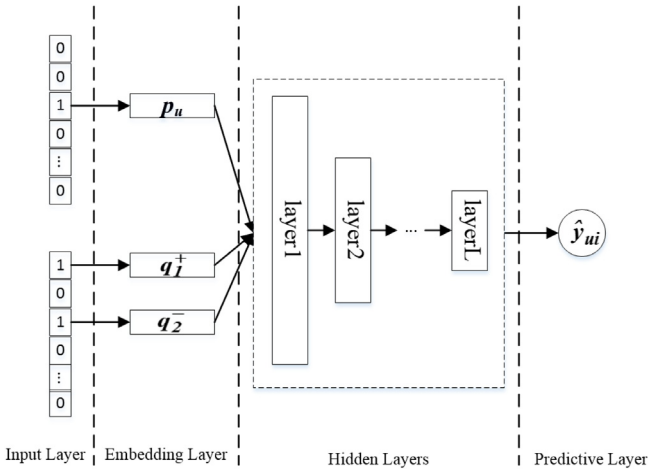


Fig. 2. Network Architecture for Pair-wise DeepRank.

where I_u^+ and I_u^- denote the sets of items rated and unrated by user u, respectively.

DeepRank of pair-wise optimizes ranking loss by modeling the relative order of two items to calculate the probability of the pair-wise preference of the users between their items. We set K=2 and then optimize the objective function of DeepRank. The graphical representation of pair-wise DeepRank is shown in Fig. 2.

3.4. Relations with other models

Relations with ListRank-MF. ListRank-MF is a well-known list-wise method for recommendation, and it is most related to the top-one list-wise DeepRank. After $g(x) = e^x$ is defined in Eq. (1), $P_{l_u}(r_{ui})$ is viewed as the softmax function of the deep learning models. To be consistent with our objective function, let $y'_{ui} = P_{l_u}(r_{ui})$, and $\hat{y}'_{ui} = P_{l_u}\left(f\left(\boldsymbol{p}_u^T \boldsymbol{q}_i\right)\right)$, the objective function, Eq. (2) is rewritten as:

$$L = -\sum_{u=1}^{N} \sum_{i \in l_u^+} y'_{ui} log\hat{y}'_{ui} + \lambda\Omega(\Theta). \tag{18}$$

From Eq. (18), we find that different from our model, it models only on observed instances. But our model is trained on both observed and unobserved instances. If we train top-one list-wise DeepRank with only observed instances, we treat ListRank-MF as a special network of top-one list-wise DeepRank without hidden layers and sigmoid function as the activation function.

Compared with the ListRank-MF, our model has three major advantages:

(1) DeepRank models on top-n, rather than top-one ListRank-MF adopted, which learns more ranking information from list;

(2) DeepRank uses neural networks and nonlinear function to learn latent features for users and items, rather than linear combination ListRank-MF adopted, which has a more powerful representation than ListRank-MF;

(3) DeepRank is able to set the embedding size of users and items to different sized dimensions, rather than fixing them at a same value in ListRank-MF, which has stronger robustness and generalization.

Relations with BPR. BPR is one of the most popular pair-wise methods for ranking, and it is most related to the pair-wise DeepRank model. Pair-wise methods predict the order of items for each user by calculating scores for pair-wise preferences,

Table 1
Basic information for data sets.

Data sets	# of users	# of items	# of ratings
MovieLens100K	943	1,682	100,000
MovieLens1M	6,040	3,952	1,000,000
Yahoo! Movie	7,642	11,915	211,231

rather than predicting ratings. The objective function of BPR is defined as following:

$$L = \sum_{u=1}^{N} \sum_{i \in I_u^+, j \in I_u^-} -log\sigma(\hat{x}_{uij}) + \lambda\Omega(\Theta), \tag{19}$$

where $\hat{x}_{uij} = \boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j$, and $\sigma(x) = 1/(1+exp(-x))$, the sigmoid function.

After $\hat{y}_{uij} = \sigma(\hat{x}_{uij}) = 1/(1+exp(-\hat{x}_{uij}))$ is defined in Eq. (19):

$$\begin{aligned} L &= \sum_{u=1}^{N} \sum_{i \in I_u^+, j \in I_u^-} -log\hat{y}_{uij} + \lambda\Omega(\Theta) \\ &= -\sum_{u=1}^{N} \left(\sum_{i \in I_u^+} log\hat{y}_{ui} + \sum_{j \in I_u^-} log(1-\hat{y}_{uj}) \right) + \lambda\Omega(\Theta). \end{aligned} \tag{20}$$

Then, we find that the loss function of BPR is same as the loss function of pair-wise DeepRank in Eq. (17). In DeepRank, $\hat{x}_{uij} = f_{MLP}(\boldsymbol{p}_u, \boldsymbol{q}_i)$, and

$$\hat{y}_{uij} = softmax(\hat{x}_{uij}) = \frac{e^{x_{ui}}}{e^{x_{ui}} + e^{x_{uj}}} = \frac{1}{1 + e^{-(x_{ui} - x_{uj})}} = \frac{1}{1 + e^{-x_{uij}}}$$

If the interaction function between $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ is the inner product, and no hidden layer in our model, we get $\hat{x}_{uij} = \boldsymbol{p}_u^T \boldsymbol{q}_i$, and $\hat{y}_{uij}$ is the sigmoid function, which are same in BPR. So, BPR can be viewed as a special case of pair-wise DeepRank. And DeepRank is a more flexible architecture for pair-wise ranking.

4. Experiments

First, we introduce the data sets used in our experiments. Then, we present the baselines we compared with our model and the metrics we adopted for evaluation. Finally, we conduct the experiments in detail and then answer the following research questions:

-RQ1: How does DeepRank perform compared with other methods?

-RQ2: How do the different dimension sizes between user and item embedding affect the performance of the model?

-RQ3: How does the depth of the model affect DeepRank?

4.1. Experimental setting

Data Sets. We conducted experiments on three public data sets: MoiveLens100K, MovieLens1M and Yahoo! Movie. MovieLens data sets (available from the MovieLens web site[1]) and the Yahoo! Movie data set (available from Yahoo! Labs[2]) are widely used in recommendation experiments. The basic information (data sets, number of ratings, number of users, number of items) is shown in Table 1. For all data sets, each user rated at least twenty movies, and each item is rated at least by one user.

Baseline Approaches. Some baseline approaches, required to quantitatively evaluate the performance of our proposed models, are introduced briefly as follows:

-**BPR**: BPR [7], which learns to rank by optimizing a pair-wise loss function to find the correct personalized ranking for all items, is one of the most famous ranking methods for CF.

-**ListRank-MF**: ListRank-MF [9], which uses MF to calculate the top-one probability of an unobserved item in the recommended list of each user, is one of the most popular list-wise learning methods in recommendation;

-**NeuMF**: NeuMF [16], which combines Multi-Layer Perception (MLP) and Generalized Matrix Factorization (GMF) to learn the interaction between users and items, is one of the state-of-the-art CF methods. (The code for NeuMF is available online at: https://github.com/hexiangnan/neuarl_collaborative_filtering);

-**DeepCF**: DeepCF [27], which adds a new interaction layer for users and items before being fed to MLP for collaborative ranking, is a point-wise method for ranking learning.

Parameter Settings and Experimental Setup. We set the learning rate and regularization coefficient (λ) in all methods at 0.001 and 10^{-6}, respectively; the number of latent features (k) in BPR and ListRank-MF, embedding size in NeuMF and DeepCF at 16; hidden layer size, epochs, and batch size in NeuMF, DeepCF, and our models at 3, 50, and 512, respectively.

All experiments were conducted on a computer with 32GB memory and 8 core Intel i7 3.0 GHz, and were implemented on Tensorflow.

Evaluation Metrics. We regard the recommend issue as a ranking issue. Then, we follow the *leave-one-out* evaluation, which is a popular way to measure the ranking quality for recommendation, used by Rendle et al. [6], He et al. [16], and Deng et al. [23]. For each user, we randomly sampled one rated item and 100 unrated items as testing data. Finally, to evaluate the relevant performance of the top-n results for all methods, we adopted *NDCG* and the Hit Ratio (*HR*). *NDCG* is sensitive to the relevance of higher ranked items and assigns higher scores to correct recommendations at higher ranks in the list. *NDCG@n* and *HR@n* is defined as follows:

$$DCG@n = \sum_{i=1}^{n} \frac{2^{r_i} - 1}{log(1+i)}$$

$$NDCG@n = \frac{DCG@n}{IDCG@n}$$

$$r_i = \begin{cases} 1, & \text{if the item at position } i \text{ is a hit item} \\ 0, & \text{other} \end{cases}$$

where *IDCG@n* refers to the maximum possible value of *DCG*, which is used to normalize the *NDCG@n* value. The higher the *NDCG@n* value, the better the performance.

The *HR@n* score is defined as:

$$HR@n = \frac{hits}{n}$$

where *hits* denotes the time relevant items are in the top-n list of each user; and n denotes the number of the top-n items generated from methods. The higher the *HR* value, the better the performance.

[1] https://grouplens.org/datasets/movielens/.
[2] https://webscope.sandbox.yahoo.com/.

Table 2
Impact of the length of the list on *HR@10*.

HR@10			
K	MovieLens100K	MovieLens1M	Yahoo! Movie
2	0.7402	0.7556	0.8672
5	0.7635	0.7568	0.8723
10	0.7667	0.7738	0.8765
15	**0.7699**	**0.7745**	**0.8804**

Table 3
Impact of the length of the list on *NDCG@10*.

NDCG@10			
K	MovieLens100K	MovieLens1M	Yahoo! Movie
2	0.4783	0.4898	0.7062
5	0.4902	0.5011	0.7104
10	0.4988	0.5148	0.7176
15	**0.5099**	**0.5159**	**0.7275**

4.2. Overall performance (RQ1)

To empirically evaluate DeepRank quantitatively, we performed experiments on all data sets and compared the results with baselines to show the performance of DeepRank. The performance of all methods on data sets MovieLens100K, MovieLens1M and Yahoo! Movie is shown in Fig. 3.

As seen in Fig. 3, in general, the results of *HR* and *NDCG* are consistent, and their performance is mimicked on different data sets. As the number (n) of the top-n items increases, all methods improve at ranking prediction. Our proposed method achieves superior ranking performance. Also, DeepRank consistently outperforms the state-of-the-art method, NeuMF, by a considerable margin.

On the sparsest data set, Yahoo! Movie, DeepRank significantly outperforms the baseline methods, indicating that the skills used in our model are very effective and ensure the high performance of the model. BPR and ListRank-MF methods achieve limited performance across all data sets. Because they do not have complex and deeper interaction in the data. NeuMF learns latent features only from user–item rating data without considering any ranking information about items. Therefore, some important information about ranking is missing in the point-wise method, NeuMF . Compared with NeuMF, ranking learning methods modeled on a ranked items set are better for generating a personalized ranking list of items for users.

The task for personalized recommendation is to provide users with a ranked list of items. Point-wise methods (NeuMF and DeepCF), trained only on user–item pairs to predict the probabilities between users and items, do not consider any ranking information about items, as pair-wise and list-wise methods do. In contrast, our models (both pair-wise DeepRank and list-wise DeepRank) rank learning methods and capture a user's features from his pair-wise or list-wise behavior on items, where a set of ranked items is used to represent each user. Therefore, the ability of our models to predict a personalized ranking performance is more powerful than that of NeuMF and DeepCF.

Impact of the Length of the List. We also conduct extensive experiments to investigate the impact of the length of the list (K) for list-wise DeepRank. When K is equal to 2, it is equivalent to pair-wise DeepRank, which is the most simple model of DeepRank. The results are reported in Tables 2 and 3.

From Tables 2 and 3, first, we observe that more positions information of items added to our model can further improve its performance. As the length of the list, K, increases, the values of *HR@10* and *NDCG@10* of the model increase. On all data sets, DeepRank with the larger value of K achieves better results than that with the smaller one. Second, when this value of K is less than 5, the experimental results improve significantly. And when the value of K is greater than 5, the improvement is slight. So, it is an appropriate value when K equates to 5.

Table 4
Time costs of all methods.

	l	MovieLens100K	MovieLens1M	Yahoo! Movie
BPR	2	3 m 52 s	20 m 18 s	30 m 17 s
ListRank-MF	5	6 m 29 s	2 h 31 m 15 s	8 h 28 m 22 s
NeuMF	1	3 m 27 s	18 m 44 s	34 m 43 s
DeepCF	1	3 m 10 s	16 m 58 s	31 m 38 s
pair-wise DeepRank	2	2 m 3 s	13 m 12 s	19 m 2 s
list-wise DeepRank	5	5 m 18 s	1 h 56 m 39 s	6 h 27 m 6 s

Table 5
Impact of different dimension sizes of embedding.

d_u	d_i	MovieLens100K		MovieLens1M	
		HR@10	*NDCG@10*	*HR@10*	*NDCG@10*
8	8	0.7169	0.4585	0.7393	0.4673
	16	**0.7699**	0.4894	0.7530	0.4928
16	8	0.7381	0.4748	**0.7593**	0.4973
	16	0.7635	**0.4902**	0.7568	**0.5011**
32	8	0.7423	0.4752	0.7548	0.4921
	16	0.7466	0.4765	0.7524	0.4911

Time Cost. We conducted some experiments to compare all baseline approaches and show their efficiency. The training time for all models is shown in Table 4, where l is the length of items for one-time training.

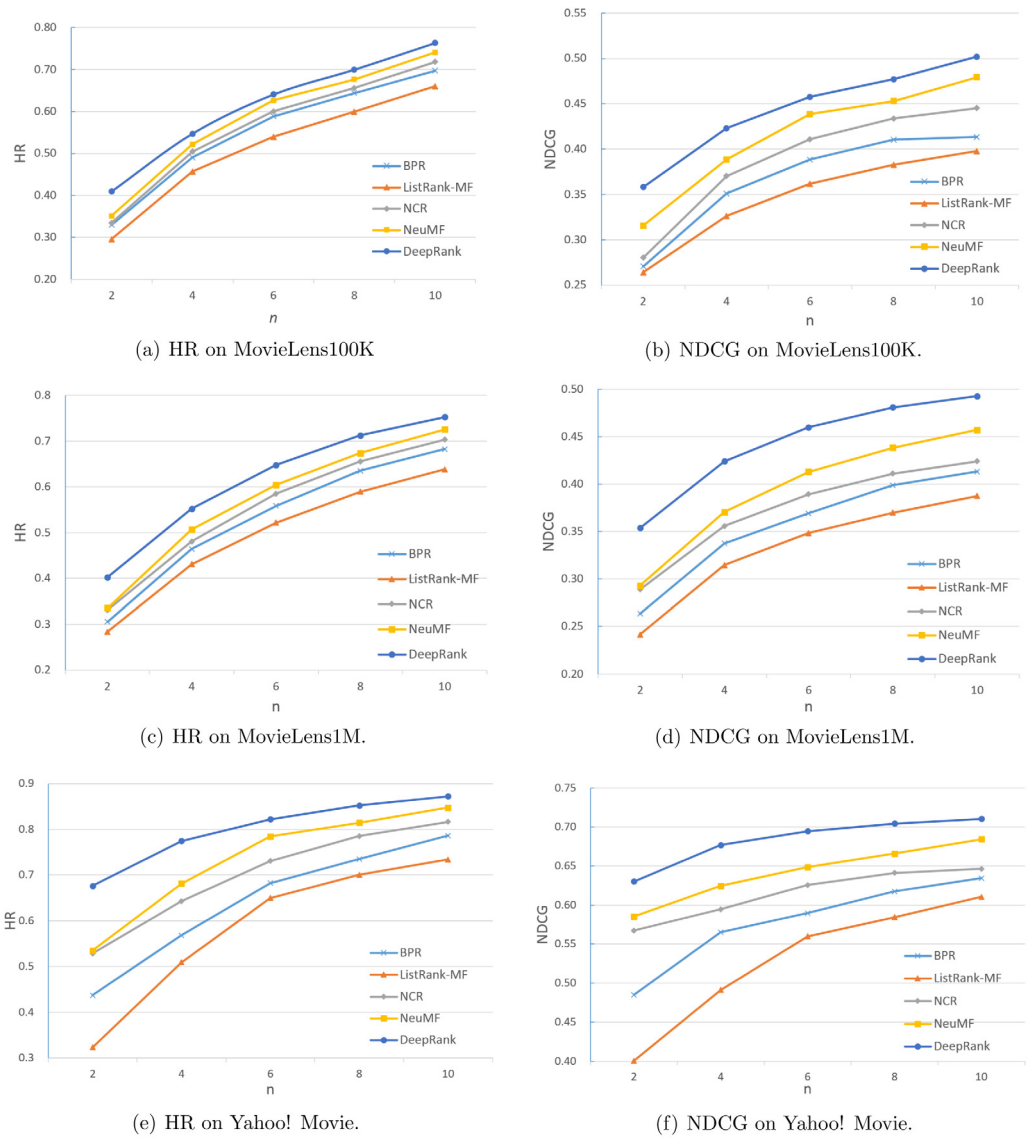
From the results shown in Table 2, Table 3, and Table 4, the following is observed: (1) Both of our models require a longer time to make a prediction than do NeuMF and DeepCF. The main reason is DeepRank requires more time to generate ranked lists of items from raw user–item interactions for training; but NeuMF and DeepRank (point-wise methods) directly train on user–item pairs; (2) For the same data set, pair-wise DeepRank spends less time than BPR. Pair-wise DeepRank and BPR are pair-wise methods that require generating pair-wise interactions from raw data. Although our model has more parameters, it achieves more stable convergence than the MF based method, BPR; (3) List-wise DeepRank and ListRank-MF are list-wise methods that require generating list instances from raw data. Therefore, for the same data set, they spend about the same amount of time. Our models achieve good performance, but they require more time to generate ranked lists of items from raw user–item interactions for training. To sum up, pair-wise DeepRank can be chosen for efficiency; for higher values of HR and NDCG, list-wise DeepRank is a better choice.

4.3. Effect of embedding size (RQ2)

Second, we performed some experiments to show the effect of different dimension sizes of embedding for generalization. We verify the effect of different dimension sizes of users and items embedding in our method.

High quality embedding is critical to the representation of items. We set the user embedding dimension size $d_u = \{8, 16, 32\}$, the item embedding dimension size $d_i = \{8, 16\}$, and the length of list, K=5. Other parameters were set as same in previous experiments. Results are shown in Table 5.

From Table 5, we observe the following: First, compared with the same dimension sizes of users and items, setting different dimension sizes for users and items further improves the performance of the model. Second, when the dimension size of user embedding is small, a better result is achieved by setting the



(a) HR on MovieLens100K (b) NDCG on MovieLens100K.

(c) HR on MovieLens1M. (d) NDCG on MovieLens1M.

(e) HR on Yahoo! Movie. (f) NDCG on Yahoo! Movie.

Fig. 3. Performance vs. the number (n) of the top-n items.

dimension size of item embedding to a larger value. When the dimension size of user embedding is large, it is better to set the dimension size of item embedding to a smaller value. Next, comparing the performance on both data sets, on the larger data set, MovieLens1M, the model must set higher dimension sizes of embedding to obtain the better results. We believe that when users have more rating information, it is better to have a larger dimension size for their embedding. Finally, we conclude that too many parameters cause the model to be over-fitting; too few parameters cause the model to be under-fitting. By adjusting the parameters, the risk of model over-fitting and under-fitting is reduced, thereby improving the generalization of the model.

4.4. Effect of depth (RQ3)

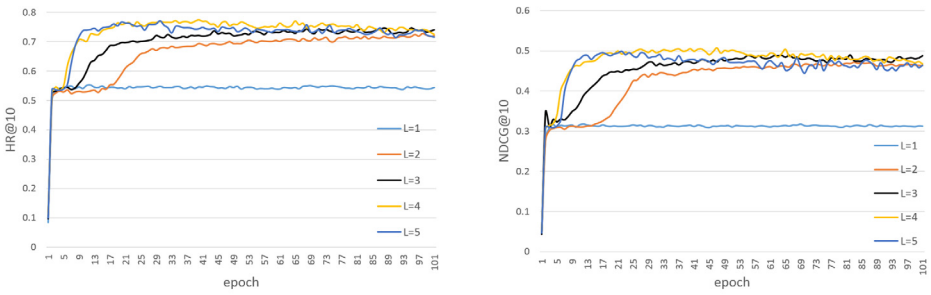
Finally, to make full use of the capacity of list-wise DeepRank, we explored the impact of the depth of hidden layers on ranking performance. In this experiment, we set the sizes of the hidden layers at [8], [16, 8], [32, 16, 8], [64, 32, 16, 8], and [128, 64, 32, 16, 8]. We set the dimensional sizes of users and items d_u, d_i, to the same value: the size of the first hidden layer divided by 2. Extensive experimentation was conducted to determine the effect of the depth of hidden layers for DeepRank. We set the following: number of hidden layers from 1 to 5, length of list, K=5. Experimental results are shown in Table 6.

Table 6
Effect of the depth of hidden layers.

L	MovieLens100K		MovieLens1M	
	HR@10	*NDCG@10*	*HR@10*	*NDCG@10*
1	0.5557	0.3153	0.5626	0.3316
2	0.7136	0.4605	0.7255	0.4506
3	0.7635	0.4892	0.7568	**0.5011**
4	**0.7738**	**0.5032**	**0.7579**	0.4946
5	0.7702	0.4710	0.7528	0.4977

As seen in Table 6, more hidden layers further improve model performance. As the number of hidden layers increases, the values of *HR@10* and *NDCG@10* of the model increase. On both data sets, DeepRank, achieves better results with a larger number of hidden layers than with a smaller number of hidden layers. Also, when the number of hidden layers is fewer than three, the performance of the model increases significantly.

To further compare the effect of the depth of hidden layers on our model, we do some experiments to present the *HR@10*



(a) *HR@10* on MovieLens100K. (b) *NDCG@10* on MovieLens100K.

Fig. 4. Performance of the depth of hidden layers.

and *NDCG@10* performance in training on MovieLens100K data set (see Fig. 4). From Fig. 4, as the epoch increases, the curve of highest number of hidden layers rises fastest in both *HR@10* and *NDCG@10* metrics. And list-wise DeepRank with larger value of the number of hidden layers converges faster than the smaller one. More hidden layers further improve model performance, but the values of *HR@10* and *NDCG@10* increase very slightly, even down, when L is larger than three. Deep neural networks have a strong representation power for modeling, however, too few parameters make the model under-fitting, and too many parameters can easily lead model over-fitting. Therefore, we conclude that a sensible number of hidden layers is indeed helpful for improving the model.

5. Conclusion

We proposed a novel method, DeepRank, for ranking recommender systems. DeepRank, a promising tool for recommendation, provides new insight into CF models for ranking learning. Compared with existing ranking-oriented methods, our method achieves better performance and presents higher quality recommendations. In addition, our model has several outstanding advantages: (1) It captures user and item latent features in a complicated and nonlinear architecture; (2) Because it has a simple and flexible structure, our model is extended or simplified easily to other scenarios.

In future work, we will explore several possible directions. First, to enrich our model for better quality, we will add some additional sources of information, such as textual information and image description information. Second, because of our interest in online learning, we plan to apply our model to develop the performance of online CF. Finally, for further improvement, we will incorporate our model with other deep learning models.

CRediT authorship contribution statement

Ming Chen: Supervision, Writing - review & editing. **Xiuze Zhou:** Conceptualization, Methodology, Writing - original draft, Investigation, Software, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors thank Michael McAllister for proofreading this paper.

References

[1] Shuaiqiang Wang, Shanshan Huang, Tie-Yan Liu, Jun Ma, Zhumin Chen, Jari Veijalainen, Ranking-oriented collaborative filtering: A listwise approach, ACM Transactions on Information Systems (TOIS) 35 (2) (2016) 1–28, http://dx.doi.org/10.1145/2960408.

[2] Qiannan Zhu, Xiaofei Zhou, Zeliang Song, Jianlong Tan, Li Guo, DAN: Deep attention neural network for news recommendation, in: Proceedings of the AAAI Conference on Artificial Intelligence - AAAI '19, Honolulu, Hawaii, USA, vol. 33, 2019, pp. 5973–5980, http://dx.doi.org/10.1609/aaai.v33i01.33015973.

[3] Qian Zhang, Jie Lu, Dianshuang Wu, Guangquan Zhang, A cross-domain recommender system with kernel-induced knowledge transfer for overlapping entities, IEEE Transactions on Neural Networks and Learning Systems 30 (7) (2019) 1998–2012, http://dx.doi.org/10.1109/TNNLS.2018.2875144.

[4] Antonio Jesús Fernández-García, Luis Iribarne, Antonio Corral, Javier Criado, James Z Wang, A recommender system for component-based applications using machine learning techniques, Knowledge-Based Systems 164 (2019) 68–84, http://dx.doi.org/10.1016/j.knosys.2018.10.019.

[5] Yao Wu, Christopher DuBois, Alice X Zheng, Martin Ester, Collaborative denoising auto-encoders for top-n recommender systems, in: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining - WSDM '16, San Francisco, California, USA, 2016, pp. 153–162, http://dx.doi.org/10.1145/2835776.2835837.

[6] Rasaq Otunba, Raimi A Rufai, Jessica Lin, MPR: Multi-objective pairwise ranking, in: Proceedings of the Eleventh ACM Conference on Recommender Systems - RecSys '17, Como, Italy, 2017, pp. 170–178, http://dx.doi.org/10.1145/3109859.3109903.

[7] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, Lars Schmidt-Thieme, BPR: Bayesian personalized ranking from implicit feedback, in: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence - UAI '09, Montreal, Quebec, Canada, 2009, pp. 452–461, http://dx.doi.org/10.5555/1795114.1795167.

[8] Seung-Taek Park, Wei Chu, Pairwise preference regression for cold-start recommendation, in: Proceedings of the Third ACM Conference on Recommender Systems - RecSys '09, New York, New York, USA, 2009, pp. 21–28, http://dx.doi.org/10.1145/1639714.1639720.

[9] Yue Shi, Martha Larson, Alan Hanjalic, List-wise learning to rank with matrix factorization for collaborative filtering, in: Proceedings of the Fourth ACM Conference on Recommender Systems - RecSys '10, Barcelona, Spain, 2010, pp. 269—272, http://dx.doi.org/10.1145/1864708.1864764.

[10] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, Hwanjo Yu, Convolutional matrix factorization for document context-aware recommendation, in: Proceedings of the Tenth ACM Conference on Recommender Systems - RecSys '16, Boston, Massachusetts, USA, 2016, pp. 233–240, http://dx.doi.org/10.1145/2959100.2959165.

[11] Hao Wang, Naiyan Wang, Dit-Yan Yeung, Collaborative deep learning for recommender systems, in: Proceedings of the Twenty-First ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15, Sydney, NSW, Australia, 2015, pp. 1235–1244, http://dx.doi.org/10.1145/2783258.2783273.

[12] Julian McAuley, Christopher Targett, Qinfeng Shi, Anton Van Den Hengel, Image-based recommendations on styles and substitutes, in: Proceedings of the Thirty-Eighth International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '15, Santiago, Chile, 2015, pp. 43–52, http://dx.doi.org/10.1145/2766462.2767755.

[13] Wenhui Yu, Zheng Qin, Spectrum-enhanced pairwise learning to rank, in: Proceedings of the Twenty-Eighth International Conference on World Wide Web - WWW '19, New York, NY, USA, 2019, pp. 2247–2257, http://dx.doi.org/10.1145/3308558.3313478.

[14] Laura Blédaité, Francesco Ricci, Proceedings of the Twenty-Sixth ACM Conference on Hypertext & Social Media - HT '15, Guzelyurt, Northern Cyprus, 2015, pp. 231—236, http://dx.doi.org/10.1145/2700171.2791049,

[15] Shuang Qiu, Jian Cheng, Ting Yuan, Cong Leng, Hanqing Lu, Item group based pairwise preference learning for personalized ranking, in: Proceedings of the Thirty-Seventh International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '14, Gold Coast, Queensland, Australia, 2014, pp. 1219–1222, http://dx.doi.org/10.1145/2600428.2609549.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua, Neural collaborative filtering, in: Proceedings of the Twenty-Sixth International Conference on World Wide Web - WWW '17, Perth, Australia, 2017, pp. 173–182, http://dx.doi.org/10.1145/3038912.3052569.

[17] Kangkang Li, Xiuze Zhou, Fan Lin, Wenhua Zeng, Gil Alterovitz, Deep probabilistic matrix factorization framework for online collaborative filtering, IEEE Access 7 (2019) 56117–56128, http://dx.doi.org/10.1109/ACCESS.2019.2900698.

[18] Yehuda Koren, Robert Bell, Chris Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.

[19] Xiuze Zhou, Shunxiang Wu, Rating LDA model for collaborative filtering, Knowledge-Based Systems 110 (2016) 135–143, http://dx.doi.org/10.1016/j.knosys.2016.07.020.

[20] Markus Weimer, Alexandros Karatzoglou, Quoc V Le, Alex J Smola, Cofi rank-maximum margin matrix factorization for collaborative ranking, 2008, pp. 1593–1600,

[21] Liwei Wu, Cho-Jui Hsieh, James Sharpnack, Sql-rank: a listwise approach to collaborative ranking, in: Proceedings of the Twenty-Fifth International Conference on Machine Learning - ICML'08, Stockholmsmässan, Stockholm, Sweden, 2018, pp. 5315–5324.

[22] Seongjun Yun, Raehyun Kim, Miyoung Ko, Jaewoo Kang, Sain: Self-attentive integration network for recommendation, in: Proceedings of the Forty-Second International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR'19, Paris, France, 2019, pp. 1205–1208, http://dx.doi.org/10.1145/3331184.3331342.

[23] Zhi-Hong Deng, Ling Huang, Chang-Dong Wang, Jian-Huang Lai, S Yu Philip, Deepcf: a unified framework of representation learning and matching function learning in recommender system, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence - AAAI '19, Honolulu, Hawaii, USA, vol. 33, 2019, pp. 61–68, http://dx.doi.org/10.1609/aaai.v33i01.330161.

[24] Shang Liu, Zhenzhong Chen, Hongyi Liu, Xinghai Hu, User-video co-attention network for personalized micro-video recommendation, in: Proceedings of the Twenty-Eighth International Conference on World Wide Web - WWW '19, San Francisco, CA, USA, 2019, pp. 3020–3026, http://dx.doi.org/10.1145/3308558.3313513.

[25] Xiangnan He, Tat-Seng Chua, Neural factorization machines for sparse predictive analytics, in: Proceedings of the Fortieth International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '17, New York, New York, USA, 2017, pp. 355–364, http://dx.doi.org/10.1145/3077136.3080777.

[26] Shuai Zhang, Lina Yao, Xiwei Xu, Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders, in: Proceedings of the Fortieth International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '17, Shinjuku, Tokyo, Japan, 2017, pp. 957–960, http://dx.doi.org/10.1145/3077136.3080689.

[27] Quangui Zhang, Longbing Cao, Chengzhang Zhu, Zhiqiang Li, Jinguang Sun, Coupledcf: Learning explicit and implicit user-item couplings in recommendation for deep collaborative filtering, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence - IJCAI '18, Stockholm, Sweden, 2018, pp. 3662–3668, http://dx.doi.org/10.24963/ijcai.2018/509.

[28] Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, Eric Horvitz, Pairwise ranking aggregation in a crowdsourced setting, in: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining - WSDM '13, Rome, Italy, 2013, pp. 193–202, http://dx.doi.org/10.1145/2433396.2433420.

[29] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, Greg Hullender, Learning to rank using gradient descent, in: Proceedings of the Twenty-Second International Conference on Machine Learning - ICML '05, Bonn, Germany, 2005, pp. 89–96, http://dx.doi.org/10.1145/1102351.1102363.

[30] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, Hang Li, Learning to rank: from pairwise approach to listwise approach, in: Proceedings of the Twenty-Fourth International Conference on Machine Learning - ICML '07, Corvalis, Oregon, 2007, pp. 129–136, http://dx.doi.org/10.1145/1273496.1273513.

[31] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, Tat-Seng Chua, NAIS: Neural attentive item similarity model for recommendation, IEEE Transactions on Knowledge and Data Engineering 30 (12) (2018) 2354—2366, http://dx.doi.org/10.1109/TKDE.2018.2831682.