Joint Neural Collaborative Filtering for Recommender Systems

WANYU CHEN, National University of Defense Technology, China and University of Amsterdam, The Netherlands
FEI CAI and HONGHUI CHEN, National University of Defense Technology, China
MAARTEN DE RIJKE, University of Amsterdam, The Netherlands

We propose a Joint Neural Collaborative Filtering (J-NCF) method for recommender systems. The J-NCF model applies a joint neural network that couples deep feature learning and deep interaction modeling with a rating matrix. Deep feature learning extracts feature representations of users and items with a deep learning architecture based on a user-item rating matrix. Deep interaction modeling captures non-linear user-item interactions with a deep neural network using the feature representations generated by the deep feature learning process as input. J-NCF enables the deep feature learning and deep interaction modeling processes to optimize each other through joint training, which leads to improved recommendation performance. In addition, we design a new loss function for optimization that takes both implicit and explicit feedback, point-wise and pair-wise loss into account.

Experiments on several real-world datasets show significant improvements of J-NCF over state-of-the-art methods, with improvements of up to 8.24% on the MovieLens 100K dataset, 10.81% on the MovieLens 1M dataset, and 10.21% on the Amazon Movies dataset in terms of HR@10. NDCG@10 improvements are 12.42%, 14.24%, and 15.06%, respectively. We also conduct experiments to evaluate the scalability and sensitivity of J-NCF. Our experiments show that the J-NCF model has a competitive recommendation performance with inactive users and different degrees of data sparsity when compared to state-of-the-art baselines.

CCS Concepts: • **Information systems → Collaborative filtering**; **Recommender systems**;

Additional Key Words and Phrases: Neural recommendation, collaborative filtering

ACM Reference format:
Wanyu Chen, Fei Cai, Honghui Chen, and Maarten de Rijke. 2019. Joint Neural Collaborative Filtering for Recommender Systems. *ACM Trans. Inf. Syst.* 37, 4, Article 39 (August 2019), 30 pages.
https://doi.org/10.1145/3343117

This research was partially supported by the National Natural Science Foundation of China under No. 61702526, the Defense Industrial Technology Development Program under No. JCKY2017204B064, Ahold Delhaize, the China Scholarship Council, and the Association of Universities in the Netherlands (VSNU). All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.
Authors' addresses: W. Chen, Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, 410073, China, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands; email: wanyuchen@nudt.edu.cn; F. Cai (corresponding author) and H. Chen, Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, 410073, Changsha, China; emails: {caifei, chenhonghui}@nudt.edu.cn; M. de Rijke, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands; email: derijke@uva.nl.

1046-8188/2019/08-ART39 $15.00
https://doi.org/10.1145/3343117

1 INTRODUCTION

Recommender systems are an effective solution to help people cope with an increasingly complex information landscape. Collaborative Filtering (CF) approaches have been widely investigated and used for personalized recommendation [2, 54]. Many traditional CF techniques are based on Matrix Factorization (MF) [54]. They characterize users and items by latent factors that are extracted from the user-item rating matrix. In the latent space, traditional CF methods, such as the Latent Factor Model (LFM) [28], often predict a user's preference for an item with a linear kernel, i.e., a dot product of their latent factors, which may not be able to capture the complex structure of user-item interactions well.

Recently introduced Deep Learning (DL)-based approaches to recommender systems overcome shortcomings of conventional approaches to recommender systems, such as dynamic user preferences and intricate relationships within the data itself, and are able to achieve high recommendation quality. Today's DL-based approaches to recommender systems mostly use DL to explore auxiliary information, e.g., textual descriptions of items or audio features of music, which is then used to model item features [25, 48, 49]. For the user-item rating matrix, recent work mostly continues to use traditional MF-based approaches. Restricted Boltzmann Machines (RBMs) [39] seem to have been the first model to use neural networks to model the user-item rating matrix and obtain competitive results over traditional methods; it is a two-layer network rather than a deep learning structure. Another recent approach, Collaborative Denoising Auto-Encoder (CDAE) [52], is mainly designed for rating prediction with a one-hidden-layer neural network. Neural Collaborative Filtering (NCF) [16] uses deep neural networks for learning the interaction function from data with multi-layer perceptrons, yet it does not explore users' and items' features that are known to be helpful in improving CF recommendation performance. CDAE and NCF only exploit implicit feedback for recommendations instead of explicit rating feedback. Deep Matrix Factorization (DMF) [22] models the user-item rating matrix with a neural network that maps the users' and items' features into a low-dimensional space with non-linear projections; it uses an inner product to compute interactions between users and items and applies the same linear kernel (i.e., dot product) as LFM [28].

We hypothesize that DL should be able to effectively capture both non-linear and non-trivial user-item relationships as well as users' (items') characteristics with multi-layer projections [54]. We propose a Joint Neural Collaborative Filtering (J-NCF) model that enables two processes—feature extraction and user-item interaction modeling—to be trained jointly in a unified DL structure. The J-NCF model contains two main networks for recommendation. The first network uses the rating information of a user (an item) as the network input and outputs a vector representation for the user (the item). Then, using the connection of a user's and an item's vectors as input, the second neural network models the user-item interactions and outputs the prediction of the corresponding rating of the user and item. Thus, these two networks can be coupled tightly and trained jointly in a unified structure. Interaction modeling can optimize the feature learning process and more accurate feature representations can, in turn, improve the user-item interaction prediction. We take both implicit and explicit feedback, point-wise and pair-wise loss into account to enhance the prediction performance. In contrast, previous neural approaches such as CDAE, NCF, and DMF are all optimized only with point-wise loss functions and leave dealing with pair-wise loss as future work.

To the best of our knowledge, in the area of recommender systems ours is the first attempt to use a joint neural network to tightly couple feature learning and interaction modeling with the rating matrix. J-NCF allows these two processes to optimize each other through joint training and thereby improve the recommendation performance.

Our experiments on real-world datasets, including the MovieLens dataset and the Amazon Movies dataset, show that J-NCF outperforms the state-of-the-art baselines in prediction accuracy, with improvements of up to 8.24% on the MovieLens 100K dataset, 10.81% on the MovieLens 1M dataset, and 10.21% on the Amazon Movies dataset in terms of HR@10. NDCG@10 improvements are 12.42% on the MovieLens 100K dataset, 14.24% on the MovieLens 1M dataset, and 15.06% on the Amazon Movies dataset, respectively, over the best baseline model. In addition, we investigate the scalability and sensitivity of J-NCF with different degrees of sparsity and different numbers of users' ratings. Our experimental results indicate that J-NCF achieves competitive recommendation performance when compared to the best state-of-the-art model.

Our contributions in this article are:

(1) We design a Joint Neural Collaborative Filtering model (J-NCF) for recommendation, which enables deep feature learning and deep user-item interaction modeling to be coupled tightly and jointly optimized in a single neural network.
(2) We design a new loss function that explores the information contained in both point-wise and pair-wise loss as well as implicit and explicit feedback.
(3) We analyze the recommendation performance of J-NCF as well as baseline models and find that J-NCF consistently yields the best performance. J-NCF also shows competitive improvements over the best baseline model when applied with inactive users and different degrees of data sparsity.

We summarize related work in Section 2. Our approach, J-NCF, is described in Section 3. Section 4 presents our experimental setup. In Section 5, we report our results to demonstrate the recommendation performance of J-NCF. We also investigate the scalability and sensitivity of our model as well as other baselines in Section 6. Finally, we conclude our work in Section 7, where we also suggest future research directions.

2 RELATED WORK

In Section 2.1, we first look back to traditional approaches to recommender systems that focus on modeling the similarity between users (items) for recommendation. Then, as applying deep learning techniques into recommender systems is gaining momentum due to its state-of-the-art performance and high-quality recommendations, in Section 2.2, we summarize recent work on deep learning–based recommender systems that can provide a better understanding of users' demands, items' characteristics, as well as historical interactions between them by extracting the features of items with auxiliary information, e.g., the content of movies.

2.1 Traditional Recommender Systems

In many commercial systems, "best bet" recommendations are shown, but the predicted rating values are not. This is usually referred to as a top-N recommendation task, where the goal of the recommender system is to find a few specific items that are supposed to be most appealing to the user. A similar prediction schema, denoted as Top Popular (Item-pop), recommends the top-N items with the highest popularity (largest number of ratings).

Most top-N recommender systems are based on collaborative filtering [2], where recommendations rely on past behavior (ratings) from users, regardless of domain knowledge [44]. We group these CF approaches into two categories, i.e., neighborhood-based methods [31, 40] and latent factor-based models [24, 28]. Neighborhood-based models share the typical merits of CF, which concentrate on exploring the similarity among either users or items. For instance, two users are similar because they have rated similarly the same set of items. A dual concept of similarity can be defined among items. Latent factor-based approaches generally model users and items as vectors in the same "latent factor" space by means of a reduced number of hidden factors. In such a space, users and items are directly comparable: the rating of a user u on an item i is predicted by the proximity (e.g., inner-product) between the related latent factor vectors.

For neighborhood-based models, algorithms that are centered around user-user similarity typically predict the rating by a user based on the ratings expressed by other users similar to her about such item. However, algorithms centered around item-item similarity compute the user preference to an item based on their own ratings to similar items. The similarity between item i and item j is measured as the tendency of users to rate items i and j similarly. It is typically based either on the cosine, the adjusted cosine, or (most commonly) the Pearson correlation coefficient [40]. The kNN (k-nearest-neighborhood) approach is a representative enhanced neighborhood model [1] that considers only the k items rated by user u that are the most similar to the item i when predicting the rating r_{ui}. kNN-based approaches discard items that are poorly correlated to the target item, thus decreasing noise for improving the quality of recommendations. Neighborhood-based approaches are similar to the item-item model for user personalization, which is different from our approach based on the user-item model [40]. Thus, we focus on the latent factor modeling approach.

Most research on latent factor modeling is based on factoring the user-item rating matrix, which is known as Singular Value Decomposition (SVD) [28]. SVD factorizes the user-item rating matrix to a product of two lower rank matrices, one containing the "user factors," the other containing the "item-factors." Then, with an inner product and biases (b_{ui}), the user's preference towards an item can be generated, i.e.,

$$\hat{y}_{ui} = b_{ui} + \mathbf{z_u}\mathbf{z_i}^{\mathrm{T}}, \tag{1}$$

where $\mathbf{z_u}$ and $\mathbf{z_i}$ denote the "user factors" and "item-factors," respectively.

Since the conventional SVD is undefined in the presence of unknown values, i.e., missing ratings, several solutions have been proposed. Earlier work addresses this issue by filling the missing ratings with a baseline estimation [41]. However, this leads to a very large, dense user rating matrix, where the factorization process becomes computationally infeasible. Recent work learns factor vectors directly on known ratings through a suitable objective function that minimizes a prediction error. The proposed objective functions are usually regularized to avoid overfitting [35]. Typically, gradient descent is applied to minimize the objective function. An advantage of SVD-based approaches is that they can provide recommendations for new users after giving their ratings towards some items without reconstructing the parameters of the models. Thus, for a new user, SVD-based approaches can provide recommendations immediately according to their current ratings.

Another model based on SVD, SVD++ [27], incorporates both explicit and implicit feedback and shows improved performance over many MF models. This is consistent with our motivation of combining explicit and implicit feedback in J-NCF. However, applying traditional MF methods to sparse ratings matrices can be a non-trivial challenge with high computational costs for decomposing the rating matrix.

Many traditional recommender systems apply a linear kernel with an inner product of user and item vectors to model user-item interactions. Linear functions may not be able to give an accurate description of the characteristics of users (items) and user-item interactions: previous work has pointed out that non-linearities have potential advantages for improving the performance of recommender systems with extensive experiments [29, 42, 52].

2.2 Deep Learning-based Recommender System

DL-based recommender systems can be divided into two categories, i.e., single neural network models and deep integration models, depending on whether they rely solely on deep learning techniques or integrate traditional recommendation models with deep learning [3, 15, 23, 32, 34, 44, 51, 54, 56].

For the first category, RBM [33, 39, 46] is an early neural recommender system. It uses a two-layer undirected graph to model tabular data, such as users' explicit ratings of movies. RBM targets rating prediction, not top-N recommendation, and its loss function considers only the observed ratings. It is technically challenging to incorporate negative sampling into the training of RBMs [52], which would be required for top-N recommendation. AutoRec [42] uses an Auto-Encoder for rating prediction. It only considers the observed ratings in the loss function, which does not guarantee good performance for top-N recommendation. To prevent the Auto-Encoder from learning an identity function and failing to generalize to unseen data, Denoising Auto-Encoders (DAEs) [29] have been applied to learn from intentionally corrupted inputs. Most of the publications listed so far focus on explicit feedback and, hence, fail to learn users' preferences from implicit feedback. CDAE [52] extends DAEs: its input is a user's partially observed implicit feedback. Unlike our work, both DAEs and CDAE use an item-item model for personalization that represents a user with their rated items [40], and the outputs are the item scores decoded from the learned user's representation. Our work is a kind of user-item model that learns users' as well as items' representations first and then calculates the relevance between them. The proposed J-NCF model is a user-item model that personalizes by modeling user-item interactions. Also, CDAE applies a linear kernel to model the relationship between users and items, whereas J-NCF applies a non-linear kernel.

Several Convolutional Neural Network (CNN)-based recommendation models have been proposed [25, 47, 48]. They primarily use CNNs to extract item features with auxiliary information, e.g., review text or contextual information, which we will incorporate in our future work. As for Recurrent Neural Networks, they are used in recommender systems that address the temporal dynamics of ratings and sequential features [20, 45].

Most closely related to our model is Neural Collaborative Filtering (NCF) [16]. It uses multi-layer perceptrons to model the two-way interaction between users and items, which is meant to capture the non-linear relationship between users and items. Let v_u^{user} and v_u^{item} denote the side information (e.g., the feature information), then, the prediction rule of NCF is formulated as follows:

$$\hat{y}_{ui} = f\left(U^{\mathrm{T}} \cdot v_u^{user}, V^{\mathrm{T}} \cdot v_u^{item} \mid U, V, \theta\right), \tag{2}$$

where the function $f(\cdot)$ defines the multilayer perceptron, and θ are the parameters of the network. However, NCF randomly initializes the representation of users and items with just a one-hot identifier of user u and item i, respectively, which only explores the users' and items' features in a limited manner. J-NCF adopts a joint neural network structure to capture both user and item features and user-item relationships, as we hypothesize that the two parts can be optimized through tight coupling and joint training. In addition, NCF only exploits implicit feedback for item recommendations and ignores explicit feedback.

An extension based on NCF is CCCFNet (Cross-domain Content-boosted Collaborative Filtering neural Network) [30]. The basic building block of CCCFNet is also a dual network (for users and items, respectively). It models the user-item interactions in the last layer with the dot product. Unlike our work, it applies content information with a neural network to capture the user's preferences and item features. In addition, DeepFM (Deep Factorization Machine) [14] is an end-to-end model that seamlessly integrates factorization machine and MLP. However, it also applies content information and thus models higher-order feature interactions via a deep neural network and low-order interactions via a factorization machine. In contrast, J-NCF adopts the rating information to explore both user and item features, which are easier to collect.

As to deep integration models, Collaborative Deep Learning (CDL) [49] is a hierarchical Bayesian model that integrates stacked DAEs into traditional probabilistic MF. It differs from our work in two ways: (1) it extracts deep feature representations of items from the content information that we do not explore, and (2) it uses a linear kernel to model relations between users and items with the dot product of user and item vectors.

A well-known integration model is DeepCoNN (Deep Cooperative Neural Network) [55], which adopts two parallel convolutional neural networks to model user behavior and item properties from review texts. In the final layer, a factorization machine is applied to capture their interactions from rating predictions. It alleviates the sparsity problem and enhances model interpretability by exploiting a rich semantic representation of the reviews, which could be investigated in J-NCF as future work.

Wide & Deep learning [12] and DeepFM [14] are two state-of-the-art recommendation works with deep learning techniques. While they focus on incorporating various features of users and items, we aim at exploring deep learning methods for pure collaborative filtering systems. Another integration model that is directly relevant to our work is Deep Matrix Factorization (DMF) [22]. It uses a deep MF model with a neural network that maps users and items into a common low-dimensional space. It follows the LFM, which uses the inner product to compute interactions between users and items. This may partially explain why using deep layers does not help to improve the performance of DMF (see Reference [22, Section 4.4]). Unlike DMF, we apply multi-layer perceptrons to model user-item interactions using a combination of user and item feature vectors as input. This does not only help our model to be more expressive in modeling user-item interactions than linear products, but it also helps to improve the accuracy of user and item feature extraction.

On top of the previous work discussed above, our proposed model J-NCF combines feature learning and interaction modeling into an end-to-end trainable neural network, which enables the two processes to be optimized jointly. Besides this, we design a new loss function that combines point-wise and pair-wise losses to explore the integration of different types of information, i.e., both implicit and explicit feedback.

3 APPROACH

The proposed model, J-NCF, has a joint structure with a layer used for modeling users' and items' features (the DF network) and a higher layer used for modeling user-item interactions (the DI network). These two layers can be trained in a joint manner to give a predicted score of a user's interactions with an item with minimum prediction error. We first describe the notation used and then detail J-NCF. We also describe the loss function that we use for optimization.

3.1 Problem Formulation and Notation

First, we describe the task of top-N recommendation that we study in this article. Suppose that there are M users and N items, denoted as $U = \{user_1, \dots, user_M\}$ and $I = \{item_1, \dots, item_N\}$. $R \in \mathbb{R}^{M \times N}$ denotes the rating information, where R_{ui} is the rating given by user $user_u$ to item $item_i$. The task for top-N recommendation is to return a list containing a set of items for an individual user to maximize the user's satisfaction.

The main notation we use in this article is listed in Table 1.

3.2 Joint Neural Collaborative Filtering

The joint architecture of the proposed J-NCF model is shown in Figure 1. The model contains two main networks: a DF network for modeling features and a DI network for modeling interactions between items and users, where the output of the first network serves as the input of the second.

The DF network is used for modeling users' and items' features. It contains two parallel neural networks coupled in the last layer, one network for users (Net_{user}), and another for items (Net_{item}). We give the ratings of a user and an item as inputs to Net_{user} and Net_{item}, respectively, which are

Table 1. Main Notation Used in the Article

Notation	Description
U	the set of users
I	the set of items
R_{ui}	an explicit rating of user u to item i
$\mathbf{v_u}$	a vector containing a user's ratings; serves as input to Net$_{user}$
$\mathbf{v_i}$	a vector containing an item's ratings; serves as input to Net$_{item}$
M	the number of unique users
N	the number of unique items
$\mathbf{W_u^x}$	the weight matrix for the xth layer in Net$_{user}$
$\mathbf{b_u^x}$	the bias for the xth layer in Net$_{user}$
f_u^x	the activation function for the xth layer in Net$_{user}$
X	the number of layers in DF network
$\mathbf{W_{ui}^y}$	the weight matrix for the yth layer in the DI network
$\mathbf{a_{ui}}$	a combination of user and item vectors; serves as input to the DI network
$\mathbf{b_{ui}^y}$	the bias for the yth layer in the DI network
f_{ui}^y	the activation function for the yth layer in the DI network
Y	the number of layers in the DI network
$\hat{y}_{ui}$	the predicted score of the interaction between user u and item i
V^+	the set of items that a user rates
V^-	the set of items that are not rated by a user
α	a tradeoff parameter controlling the contributions of the point-wise loss and pair-wise loss

defined as $\mathbf{v_u} = \langle y_{u1}, \ldots, y_{uN} \rangle$ and $\mathbf{v_i} = \langle y_{1i}, \ldots, y_{Mi} \rangle$, where

$$y_{ui} = \begin{cases} 0, & \text{for unknown ratings,} \\ R_{ui}, & \text{when explicit feedback is available.} \end{cases} \tag{3}$$

We think of ratings as non-trivial explicit feedback from users, as different ratings indicate different levels of users' preferences towards items. Obviously, there are many unknown ratings between users and items indicating non-preference of a user towards an item. Following References [16, 22], we regard these unknown ratings as a kind of implicit feedback and mark them as zeros. When pursuing a top-N recommendation task, we are interested only in a correct item ranking and care less about the exact rating scores. This grants us some flexibility, like considering all missing values in the user rating matrix as zeros [13]. Thus, we can take both explicit and implicit feedback into consideration with Equation (3).

Then, with multi-layer perceptrons (MLP), the initial high-dimensional rating vectors of users and items are mapped to lower-dimensional vectors. Since Net$_{user}$ and Net$_{item}$ only differ in their inputs, we focus on illustrating the process for Net$_{user}$; the same process is applied for Net$_{item}$ with similar layers. The MLP model in the DF network is defined as:

$$\begin{aligned} \mathbf{z_u^1} &= f_u^1\left(\mathbf{W_u^1 v_u} + \mathbf{b_u^1}\right) \\ \mathbf{z_u^2} &= f_u^2\left(\mathbf{W_u^2 z_u^1} + \mathbf{b_u^2}\right) \\ &\vdots \\ \mathbf{z_u} &= f_u^X\left(\mathbf{W_u^X z_u^{X-1}} + \mathbf{b_u^X}\right), \end{aligned} \tag{4}$$

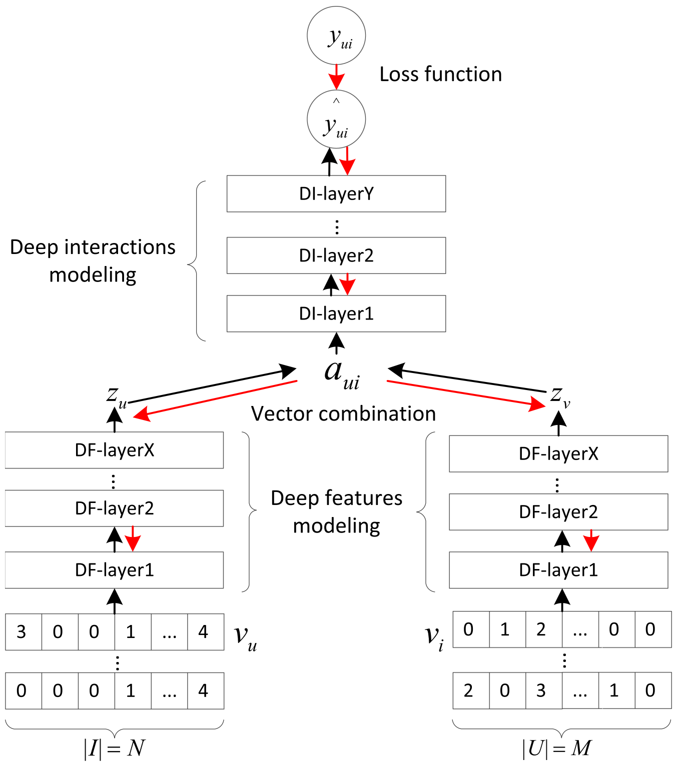


Fig. 1. Structure of the J-NCF model. Black arrows indicate the forward propagation for calculating the predictions. Red arrows indicate the back-propagation for optimizing the parameters.

where $\mathbf{W}_\mathbf{u}^\mathbf{x}$, $\mathbf{b}_\mathbf{u}^\mathbf{x}$, and f_u^x denote the weight matrix, the bias vector, and the activation function for the xth layer. Here, we use a ReLU as the activation function, as it has been shown to be more expressive than others and can effectively deal with the vanishing gradient problem [16, 22]. X indicates the number of layers used in the DF network. The output of the final layer z_u is a deep representation of the user features; likewise, z_i is the deep representation for the item features.

As to modeling user-item interactions, traditional LFM methods have been widely used. Such methods are based on the dot product of user and item vectors, which models a user's preference with a linear kernel. To investigate the differences between non-linear and linear functions in modeling user-item interactions, we propose two ways to obtain fused users' and items' feature vectors $\mathbf{a}_\mathbf{ui}$ as the input of the DI network:

$$\mathbf{a}_\mathbf{ui} = \begin{cases} \begin{bmatrix} \mathbf{z}_\mathbf{u} \\ \mathbf{z}_\mathbf{i} \end{bmatrix}, & \text{concatenation, or} \\ \mathbf{z}_\mathbf{u} \odot \mathbf{z}_\mathbf{i}, & \text{multiplication.} \end{cases} \tag{5}$$

The first way is to concatenate the two input vectors $\mathbf{z}_\mathbf{u}$ and $\mathbf{z}_\mathbf{i}$, which we regard as a non-linear fusion. The second way is to use the element-wise product of vectors, which uses a linear kernel to generate user-item interactions. Based on these two ways of fusing the input vectors $\mathbf{z}_\mathbf{u}$ and $\mathbf{z}_\mathbf{i}$, we propose two versions of J-NCF, which we discuss in detail in our experiments.

Generating $\mathbf{a}_\mathbf{ui}$ is the first step for modeling user-item interactions. However, it is insufficient for modeling the complex relationship between users and items. Thus, we adopt intermediate hidden layers to which $\mathbf{a_{ui}}$ is fed to obtain a multi-layer non-linear projection of user-item interactions:

$$\begin{aligned} \mathbf{z^1_{ui}} &= f^1_{ui}\left(\mathbf{W^1_{ui}a_{ui}} + \mathbf{b^1_{ui}}\right) \\ \mathbf{z^2_{ui}} &= f^2_{ui}\left(\mathbf{W^2_{ui}z^1_{ui}} + \mathbf{b^2_{ui}}\right) \\ &\vdots \\ \mathbf{z_{ui}} &= f^Y_{ui}\left(\mathbf{W^Y_{ui}z^{Y-1}_{ui}} + \mathbf{b^Y_{ui}}\right), \end{aligned} \tag{6}$$

where $\mathbf{W^y_{ui}}$, $\mathbf{b^y_{ui}}$, and f^y_{ui} denote the weight matrix, the bias vector and the activation function for the yth layer in the DI network. A ReLU is applied again as the activation function. Y indicates the number of layers used in the network. The output of the network is the predicted score of the interaction between user u and item i:

$$\hat{y}_{ui} = \sigma\left(\mathbf{h}^{\mathrm{T}}\mathbf{z_{ui}}\right), \tag{7}$$

where the sigmoid function σ can restrict the output in (0,1). $\mathbf{h}$ can be learned through the training process with back-propagation to control the weight of each dimension in $\mathbf{z_{ui}}$.

3.3 Loss Function

Objective functions for training recommender systems can be divided into three groups: point-wise, pair-wise, and list-wise. Point-wise objectives aim at obtaining accurate ratings, which is more applicable in rating prediction tasks [24]. Pair-wise objectives are usually focused on users' preferences towards pairs of items and are usually considered more suitable for top-N recommendation [16, 17, 24, 37]. List-wise objectives are focused on users' interests towards a list of items, which are also used in some deep learning algorithms. We briefly summarize the three groups of loss functions.

We use $\ell(\cdot)$ to denote a loss function and $\Omega(\theta)$ to represent a regularization term that controls the model complexity and encodes prior information such as sparsity, non-negativity, or graph regularization.

For a *point-wise* loss function, the general calculation is:

$$L = \sum_{u \in U} \sum_{i \in I} \ell_{\textit{point-wise}}(y_{ui}, \hat{y}_{ui}) + \lambda\Omega(\theta). \tag{8}$$

There are several types of point-wise loss function. E.g., squared loss is more suitable for explicit feedback than implicit feedback, as it is calculated with:

$$\ell_{squ} = \sum_{u \in U} \sum_{i \in I} w_{ui}(y_{ui} - \hat{y}_{ui})^2, \tag{9}$$

where w_{ui} is a hyper-parameter denoting the weight of training instance (u, i). The use of squared loss is based on the assumption that observations are generated from a Gaussian distribution; however, it may not tally well with implicit data [38]. For implicit feedback, there is a point-wise loss function mainly used for classification tasks [16, 22] named log loss [24] that can perform better with implicit feedback than squared loss:

$$\ell_{\log} = -\sum_{u \in U} \sum_{i \in I} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}). \tag{10}$$

Pair-wise loss considers the relative order of the prediction for pairs of items, which is a more reliable kind of information for top-N recommendation. Hidasi and Karatzoglou [19] investigate several popular pair-wise loss functions, i.e., TOP1, BPR-max, and TOP1-max. We give a brief introduction of them. TOP1 is the regularized approximation of the relative rank of the relevant item, which can be calculated as:

$$\ell_{\text{TOP1}} = \frac{1}{|N_S|} \sum_{j \in N_S} \sigma(\hat{y}_{uj} - \hat{y}_{ui}) + \sigma\left(\hat{y}_{uj}^2\right), \tag{11}$$

where $\hat{y}_{uj}$ and $\hat{y}_{ui}$ denote the prediction scores for a negative item j and a positive item i, respectively; N_S is the set of negative samples. The first part of TOP1 aims to ensure that the target score is higher than the score of the negative samples, while the second part pushes the score of the negative samples down. As for BPR-max and TOP1-max, they have been proposed by Hidasi and Karatzoglou [19] to overcome the vanishing gradients as the number of negative samples increases. The idea is to have the target score compared with the most relevant sample score, which is the maximum score among the samples. As the maximum operation is non-differentiable, softmax scores are used to preserve differentiability. By summing over the individual losses weighted by the corresponding softmax scores s_j, TOP1-max can be calculated as:

$$\ell_{\text{TOP1-max}} = \sum_{j \in N_S} s_j \left(\sigma(\hat{y}_{uj} - \hat{y}_{ui}) + \sigma\left(\hat{y}_{uj}^2\right)\right). \tag{12}$$

And the BPR-max loss function can be calculated as:

$$\ell_{\text{BPR-max}} = -\log \sum_{j \in N_S} s_j \sigma(\hat{y}_{ui} - \hat{y}_{uj}). \tag{13}$$

For *list-wise* loss, many deep learning–based methods combine cross-entropy loss with softmax, which introduces list-wise properties into the loss. We refer to it as softmax+cross-entropy (XE) loss, which can be calculated with the following function:

$$\ell_{\text{XE}} = -\log s_i = -\log \frac{e^{\hat{y}_{ui}}}{\sum_{j \in N_S} e^{\hat{y}_{uj}}}. \tag{14}$$

Most deep learning–based models only use the point-wise loss function for optimization and leave the pair-wise loss function for future work [16, 22]. Point-wise loss only uses the rating information and ignores the information contained in the relative order of pairs of items. Pair-wise loss, in contrast, ignores the information of a user's individual preference for a certain item. Thus, unlike previous work, NCF and DMF, our proposed J-NCF model considers both point-wise and pair-wise loss for the top-N recommendation task and combines them into a new loss function:

$$L = \alpha L_{\textit{pair-wise}} + (1 - \alpha) L_{\textit{point-wise}}, \tag{15}$$

where α is used to control the weights of the two parts.

For point-wise loss, we adopt the log loss (Equation (10)), which can integrate both implicit and explicit feedback. As to pair-wise loss, combining with different pair-wise losses yields different new loss functions, i.e., point-wise+TOP1, point-wise+BPR-max, and point-wise+TOP1-max. We analyze the performance of these different combined loss functions with experiments in Section 5.

Acknowledging that explicit and implicit feedback both contain information about a user's preference towards items, we combine both kinds of feedback in our loss function for optimization and rewrite Equation (15) in detail as

$$L = \alpha L_{\textit{pair-wise}} + (1 - \alpha)(-Y_{ui} \log \hat{y}_{ui} - (1 - Y_{ui}) \log(1 - \hat{y}_{ui})), \tag{16}$$

where $Y_{ui} = \frac{y_{ui}}{Max(R_u)}$ and $Max(R_u)$ denotes the largest rating score of user u given to items, so different values of y_{ui} have a different influence on the loss. For example, if the largest rating score of a user u given to items is 4, when he rates an item i with 2, we can generate $Y_{ui} = \frac{y_{ui}}{Max(R_u)} = \frac{2}{4}$. We refer to our loss function Equation (16) as a "*hybrid*" loss function.

ALGORITHM 1: Joint Neural Collaborative Filtering.

```
Input:  Epochs: training iterations;
        $R$: the original rating matrix;
        $U$: user set;
        $I$: item set;
Output: $\mathbf{W_u^x}$ ($x = 1, \ldots, X$): Weight matrix of Net$_{user}$;
        $\mathbf{b_u^x}$ ($x = 1, \ldots, X$): Bias of Net$_{user}$;
        $\mathbf{W_i^x}$ ($x = 1, \ldots, X$): Weight matrix of Net$_{item}$;
        $\mathbf{b_i^x}$ ($x = 1, \ldots, X$): Bias of Net$_{item}$;
        $\mathbf{W_{ui}^y}$ ($y = 1, \ldots, Y$): Weight matrix of DI network;
        $\mathbf{b_{ui}^y}$ ($y = 1, \ldots, Y$): Bias of DI network.
 1: randomly initialize $\mathbf{W_u}$, $\mathbf{W_i}$,$\mathbf{W_{ui}}$, $\mathbf{b_u}$, $\mathbf{b_i}$, and $\mathbf{b_{ui}}$;
 2: $y_{ui} \leftarrow$ use Equation (3) with $R$;
 3: $V^+ \leftarrow$ all none-zero interaction pairs;
 4: for epoch in range(Epochs) do
 5:     random shuffle of $V^+$
 6:     for $\langle u, i \rangle \in V^+$ do
 7:         sample the set of negative samples $N_S$
 8:         for $j \in N_S$ do
 9:             $\mathbf{v_u}$, $\mathbf{v_i}$, $\mathbf{v_j} \leftarrow y_{ui}$ with Equation (3);
10:             $\mathbf{z_u}$, $\mathbf{z_i}$, $\mathbf{z_j} \leftarrow$ use Equation (4) with $\mathbf{v_u}$, $\mathbf{v_i}$, $\mathbf{v_j}$ as inputs;
11:             $\mathbf{a_{ui}}$, $\mathbf{a_{uj}} \leftarrow$ use Equation (5) with $\mathbf{z_u}$, $\mathbf{z_i}$, $\mathbf{z_j}$;
12:             $\hat{y}_{ui}$, $\hat{y}_{uj} \leftarrow$ use Equation (6) and Equation (7);
13:             $L \leftarrow$ use Equation (16) with $y_{ui}$, $\hat{y}_{ui}$ and $\hat{y}_{uj}$ as inputs;
14:             use back-propagation to optimize the parameters;
15:         end for
16:     end for
17: end for
18: return $\mathbf{W_u}$, $\mathbf{W_i}$, $\mathbf{W_{ui}}$, $\mathbf{b_u}$, $\mathbf{b_i}$ and $\mathbf{b_{ui}}$.
```

We have developed the joint neural network structure of the J-NCF model. The training process of J-NCF is shown in Algorithm 1. We first initialize the parameters in the network and modify the rating matrix from step 1 to 3. Then, in steps 9 and 10, we generate deep feature representations for both users and items with the DF network. In steps 11 and 12, we calculate the predicted scores for the user-item interactions with the DI network. Finally, we use the hybrid loss function in Equation (16) and back-propagation to optimize the network parameters with steps 13 and 14.

4 EXPERIMENTAL SETUP

We design experiments on a variety of datasets to examine the effectiveness of J-NCF. We first explain the research questions and the models we use for comparison in Section 4.1. The datasets and experiments are described in Section 4.2.

4.1 Model Summary and Research Questions

We conduct experiments with the aim of answering the following research questions:

- **RQ1** Does our proposed J-NCF method outperform state-of-art collaborative filtering baselines for recommender systems?
- **RQ2** How is the performance of J-NCF impacted by different choices for the pair-wise loss in Equation (16)?

RQ3 Does the hybrid loss function Equation (13), which combines point-wise and pair-wise loss, help to improve the performance of J-NCF?

RQ4 Are deeper layers of hidden units in the DF network and DI network helpful for the recommendation performance of J-NCF?

RQ5 Does the combination of explicit and implicit feedback help to improve the performance of J-NCF?

RQ6 How does the performance of J-NCF vary across users with different numbers of interactions?

RQ7 Is J-NCF sensitive to different degrees of data sparsity?

RQ8 How does J-NCF perform on a large and sparse dataset?

RQ9 How do the training and inference times of J-NCF compare against those of other neural models?

We compare J-NCF against a number of traditional collaborative filtering baselines and against state-of-the-art deep learning–based models:

Item-pop This method ranks items based on the number of interactions, which is a non-personalized approach to determine recommendation scores [2].

BPR This method uses a pair-wise loss function to optimize an MF model based on implicit feedback. We use it as a strong baseline for traditional collaborative filtering method [37].

NCF This is a state-of-the-art neural network–based method for recommender systems. It aims to capture the non-linear relationship between users and items. Unlike J-NCF, it simply uses one-hot vectors representing users and items as the input for modeling user-item interactions. And it only uses implicit feedback and a point-wise loss function [16].

DMF This method uses multi-layer perceptrons for rating matrix factorization. Unlike our work, after projecting users and items into low dimensional vectors, it applies an inner product to calculate interactions between users and items, which is a linear kernel. It uses a point-wise loss function for optimization [22].

In addition, following the choices that we identified in Equation (5), we consider two versions of J-NCF:

J-NCF$_m$ This is J-NCF using element-wise multiplication for combining a user and an item feature vector as the input for the DI layer, which has a linear kernel inside.

J-NCF$_c$ This is J-NCF using concatenation for combining a user and an item feature vector as the input for the DI layer, which is a non-linear way.

We list all the models to be discussed in Table 2.

4.2 Datasets and Experimental Setup

4.2.1 Datasets. We use three publicly available datasets to evaluate our models and the baselines:

(1) **MovieLens**, which contains several rating datasets from the MovieLens web site. The datasets are collected over various periods of time, depending on the size of the set [16, 22]. We use two sets for our experiments, i.e., MovieLens 100K (ML100K) containing 100,000 ratings from 943 users on 1,682 movies, and MovieLens 1M (ML1M) containing more than 1M ratings from 6,040 users on 3,706 movies.[1]

[1]https://grouplens.org/datasets/movielens/.

Table 2. An Overview of the Models Discussed in the Article

Model	Description	Source
Item-pop	A typical recommendation approach, which ranks items based on the number of interactions.	[2]
BPR	A recommendation method using a pair-wise loss function to optimize an MF model based on implicit feedback.	[37]
NCF	A state-of-the-art neural-based method for recommender systems.	[16]
DMF	A method using multi-layer perceptrons for rating matrix factorization.	[22]
J-NCF$_m$	A J-NCF model using element-wise multiplication for combining a user and an item feature vector as the input for the DI layer.	This article
J-NCF$_c$	A J-NCF model using concatenation for combining a user and an item feature vector as the input for the DI layer.	This article
J-NCF$_{point}$	A J-NCF model with only point-wise loss based on Equation (10).	This article
J-NCF$_{pair}$	A J-NCF model with only pair-wise loss based in Equation (11).	This article
J-NCF$_{hybrid}$	A J-NCF model with our designed loss function in Equation (13).	This article
J-NCF$_{ex}$	A J-NCF model with both explicit and implicit feedback in the input and the loss function.	This article
J-NCF$_{im}$	A J-NCF model with only implicit feedback in the input and the loss function.	This article

(2) **Amazon Movies (AMovies)**, which contains 4,607,047 ratings for movies from Amazon, which is bigger and sparser than the MovieLens datasets and used widely in the recommender systems literature for evaluation [22, 54].[2]

(3) **Amazon Electronics (AEle)**, which is a larger and sparser dataset than the other datasets used in our article. It contains 7,824,482 ratings of users on different electronics. We use it to test the performance of our model when applied on a large and sparse dataset.[3]

For the two MovieLens datasets, we do not process them, because they are already filtered. For the AMovies dataset, following References [16, 22], we filter the dataset so that, similar to the MovieLens data, only users with at least 20 interactions and items with at least 5 interactions are retained. For the larger dataset AEle, we only do minor filtering on the data, i.e., filtering the users with less than 2 interactions and items with less than 5 interactions. To answer **RQ1** to **RQ7**, we use the ML100K, ML1M, and AMovies datasets to evaluate our models and baselines. As for **RQ8** to **RQ9**, we test the models on all of the datasets. The characteristics of the datasets after preprocessing are summarized in Table 3.

To answer **RQ5**, we plot distributions of users with different numbers of interactions in the ML100K, ML1M, and AMovies datasets in Figure 2. The x-axis denotes the number of ratings while the y-axis indicates the number of users corresponding to the ratings. We see that the majority of users in the three datasets only have a few ratings, which we regard as "inactive users," and few "active users" have far more ratings. E.g., in the ML100K dataset, 61.72% of the users have fewer than 100 ratings, 32.66% have between 100 and 300 ratings, and only 5.6% of the users have more than 300 ratings.

As we will see below, the models being considered in this article achieve different scores when used on datasets with different characteristics, i.e., number of users and number of items (see

[2]http://jmcauley.ucsd.edu/data/amazon/.
[3]http://jmcauley.ucsd.edu/data/amazon/.

Table 3. Dataset Statistics

Dataset	#Users	#Items	#Ratings	#Density(%)
ML100K	943	1,682	100,000	6.3047
ML1M	6,040	3,706	1,000,209	4.4685
AMovies	15,067	69,629	877,736	0.0837
AEle	1,221,341	157,003	4,486,501	0.00234

"Density" is the density of each dataset (i.e., *#Density* = *#Ratings*/ (*#Uses* × *#Items*)).

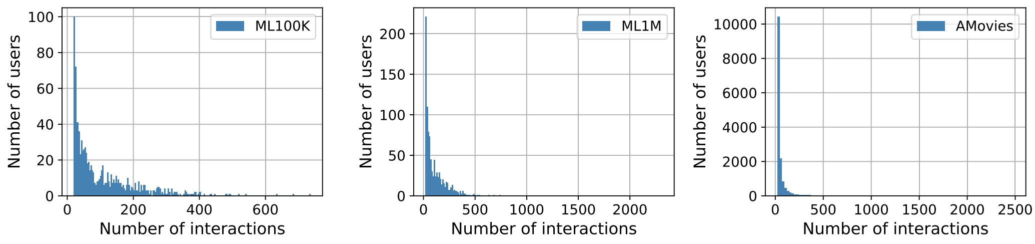


Fig. 2. Distribution of users with varying numbers of interactions in the ML100K, ML1M, and AMovies datasets, respectively.

Section 5). Thus, for **RQ6**, to evaluate the performance of our model on datasets with different degrees of sparsity, we keep the number of users and items the same. Namely, following Reference [24], for each of the three datasets, i.e., ML100K, ML1M, and AMovies, we create three versions at different sparsity levels with the following steps:

- Step 1 We start by randomly choosing a subset of users and items from the original dataset. This dataset is represented with a '−1' suffix.
- Step 2 We randomly choose a rating record and make a judgment if the numbers of users as well as items are unchanged of the sub-dataset after removing this record. If unchanged, we remove this record; otherwise, repeat Step 2.
- Step 3 After several repetitions of Step 2, the first sparser version of the dataset with the '−2' suffix is created.
- Step 4 Repeat Step 2 and Step 3 based on the dataset with a '−2' suffix, the second sparser version of the dataset with the '−3' suffix is created in the same way.

The characteristics of the datasets are summarized in Table 4.

4.2.2 Experimental Setup. For evaluation, we use a *leave-one-out* strategy that has been used widely in DL-based recommender systems [16, 17, 22]. The training set consists of all but the last interaction of every user; the test set contains the latest interaction of every user. When testing, it is time-consuming to give ranking predictions to all items for every user. Thus, following He et al. [16], Hong-Jian et al. [22], we randomly sample 100 items with which the user has not interacted and then give the test item ranking predictions among the 100 samples. Although using this sampling strategy during evaluation may overestimate the performance of all algorithms, Bellogin et al. [4] and Hidasi and Karatzoglou [19] have pointed out that the comparison among algorithms still remains fair.

The majority of the recommender system literature applies error metrics for evaluation, i.e., *Root Mean Squared Error* (RMSE) and *Mean Absolute Error* (MAE). Such classical error criteria do not really measure the top-N recommendation performance [13]. An extensive evaluation of

Table 4. Dataset Statistics with Different Degrees of Sparsity

Dataset	#Users	#Items	#Ratings	#Density(%)
ML100K-1	943	1,682	69,999	4.4132
ML100K-2	943	1,682	39,999	2.2522
ML100K-3	943	1,682	9,999	0.6304
ML1M-1	3,706	6,040	850,208	3.7982
ML1M-2	3,706	6,040	350,207	1.5645
ML1M-3	3,706	6,040	167,870	0.7499
AMovies-1	7,402	12,080	87,807	0.0982
AMovies-2	7,402	12,080	37,823	0.0423
AMovies-3	7,402	12,080	18,867	0.0211

several state-of-the-art recommender algorithms suggests that algorithms optimized for minimizing RMSE do not necessarily perform as expected in terms of the top-N recommendation task [13, 18]. Experimental results also show that improvements in terms of RMSE often do not translate into accuracy improvements [18]. Thus, here we choose to use accuracy metrics to examine the recommendation performance [16]. Specifically, we use HR and NDCG to evaluate the performance of our models. *Hit Ratio* (HR) is used to evaluate the precision of the recommender system, i.e., whether the test item is contained in the top-N list. The *Normalized Discount Cumulative Gain* (NDCG) measures the ranking accuracy of the recommender system, i.e., whether the test item is ranked at the top of the list.

As for parameters, we optimize the hyperparameters by running 100 experiments at randomly selected points of the parameter space. Optimization is done on a validation set, which is partitioned from the training set with the same procedure as the test set [11]. As for the loss function, we test the parameter α from 0 to 1 with a step size of 0.1 in our experiment. For the neural networks, we randomly initialize model parameters with a Gaussian distribution (mean of 0 and standard deviation of 0.01), optimizing the model with mini-batch Adam [26]. The batch size and learning rate are set to 256 and 0.0001. For the baselines, we set the parameters of DMF as well as NCF following References [16, 22], respectively. For DMF and NCF, we set the batch size to 256 and the learning rate to 0.0001 and 0.001. For the DF network in the DMF model, we apply two layers and the sizes of them are [128, 64]. For the DI network in the NCF model, we employ three hidden layers with sizes [128, 64, 8]. For the DF and DI networks in J-NCF, without special mention, we employ three layers in the DF network with the sizes of [256, 128, 64] and two layers in DI network with sizes of [128, 8]. Thus, the embedding sizes of users as well as items are same in all baseline models as well as J-NCF. We also keep the size of the last hidden layer of the DI network in J-NCF the same as NCF, which may determine the model capability. We also test our model as well as the baseline models with different numbers of layers to see if deep layers are beneficial to the overall performance of these models. Unless specified, for all the results presented in this article, the number of recommendations (N) is equal to 10 [16, 22].

5 RESULTS AND DISCUSSION

5.1 Overall Performance

To answer **RQ1**, we examine the recommendation performance of the baselines and the J-NCF$_m$ and J-NCF$_c$ models (see Table 5).

Let us first consider the baselines. From Table 5, we see that DMF achieves a better performance than the other baselines in terms of HR@10 and NDCG@10. Hence, we only use DMF as the best

Table 5. Performance of Recommendation Models

	ML100K		ML1M		AMovies	
Model	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
Item-pop	.3832	.2018	.4513	.2315	.5925	.3493
BPR	.5762	.3021	.6097	.3711	.6288	.3903
NCF	.6066	.3488	.6498	.3951	.6782	.4135
DMF	<u>.6309</u>	<u>.3616</u>	<u>.6748</u>	<u>.4221</u>	<u>.7151</u>	<u>.4616</u>
J-NCF$_m$	.6627$^{\triangle}$	.3877$^{\triangle}$	.7127$^{\blacktriangle}$	.4485$^{\blacktriangle}$	.7666$^{\blacktriangle}$	.5098$^{\blacktriangle}$
J-NCF$_c$	**.6829**$^{\blacktriangle}$	**.4065**$^{\blacktriangle}$	**.7377**$^{\blacktriangle}$	**.4822**$^{\blacktriangle}$	**.7881**$^{\blacktriangle}$	**.5311**$^{\blacktriangle}$

The results produced by the best baseline and the best performer in each column are underlined and boldfaced, respectively. Statistical significance of pairwise differences of J-NCF$_m$ and J-NCF$_c$ vs. the best baseline) is determined by a t-test ($^{\blacktriangle}/^{\blacktriangledown}$ for $\alpha = .01$, or $^{\triangle}/^{\triangledown}$ for $\alpha = .05$).

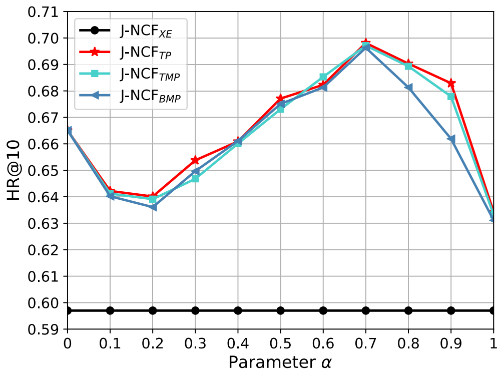
baseline for comparisons in later experiments. Bayesian Personalized Ranking (BPR) clearly shows higher improvements over the Item-pop baseline in terms of NDCG@10 than in terms of HR@10, which shows that pair-wise loss has a strong performance for ranking prediction. The NCF and DMF models both show better performance than the two traditional CF models, which indicates the utility of DL techniques in improving recommendation performance.

Next, we compare the baselines against the J-NCF models. NCF and DMF both lose against the J-NCF models in terms of HR@10 and NDCG@10. This shows that a joint neural network structure that tightly couples deep feature learning and deep interaction modeling helps to improve the recommendation performance. Regarding the J-NCF models, independent of the choice of combining the users' and items' vectors, J-NCF achieves a better performance than the DMF baseline, resulting in HR@10 improvements ranging from 5.04% to 8.24% on the ML100K dataset, 5.62% to 10.81% on the ML1M dataset, and 7.21% to 10.21% on the AMovies dataset. NDCG@10 improvements range from 7.22% to 12.42% on the ML100K dataset, 6.25% to 14.24% on the ML1M dataset, and 10.44% to 15.06% on the AMovies dataset. Significant improvements against the baseline in terms of HR@10 and NDCG@10 are observed for both J-NCF$_c$ and J-NCF$_m$ at the $\alpha = .01$ level, except for J-NCF$_m$ on the ML100K dataset, for which we observe significant improvements at the $\alpha = .05$ level in terms of HR@10 and NDCG@10. The higher improvements in NDCG@10 over HR@10 may be due to the fact that we incorporate pair-wise loss in our loss function, which motivates us to conduct a further investigation to answer **RQ3**.

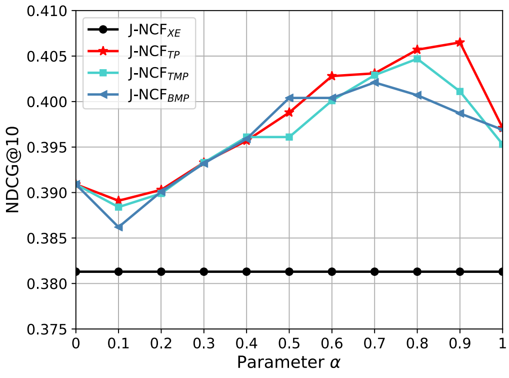
Comparing J-NCF$_c$ and J-NCF$_m$, we see that J-NCF$_c$ achieves the best performance, with improvements of 3.05%, 3.51%, and 2.81% in terms of HR@10, and 4.85%, 7.51%, and 4.18% in terms of NDCG@10 over J-NCF$_m$ on the three datasets, respectively. The complex relationship between users and items can be described better with a non-linear kernel than linear kernel, which is consistent with the findings in References [16, 33].

5.2 Impact of Different Loss Functions

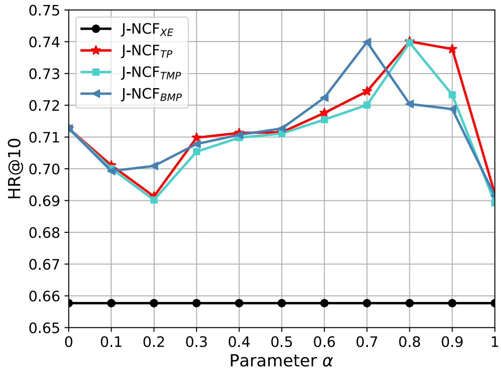
As we have mentioned in Section 3.3, there are several kinds of pair-wise loss functions that can be incorporated in Equation (15). When J-NCF combines the point-wise loss, i.e., log loss, with TOP1, TOP1-max, and BPR-max pair-wise losses, it gives rise to the J-NCF$_{TP}$, J-NCF$_{TMP}$, and J-NCF$_{BMP}$ models, respectively. Additionally, list-wise loss, i.e., softmax+cross-entropy (XE), can also be applied with J-NCF, which gives rise to the J-NCF$_{XE}$ model. To investigate the impact of various loss functions on J-NCF, we examine the recommendation performance of J-NCF$_{TP}$, J-NCF$_{TMP}$, J-NCF$_{BMP}$, as well as J-NCF$_{XE}$ models where the parameter α in Equation (15) ranges from 0 to 1 with a step size of 0.1. Figure 3 shows the results.



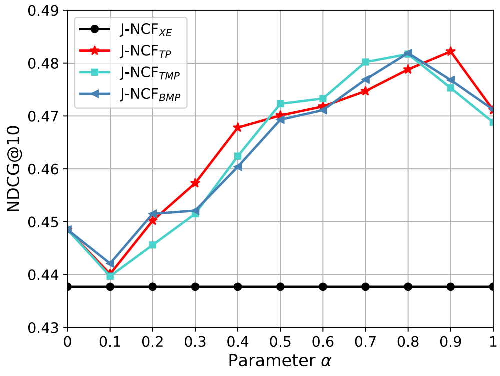
(a) Performance in terms of HR@10 on the ML100K dataset.



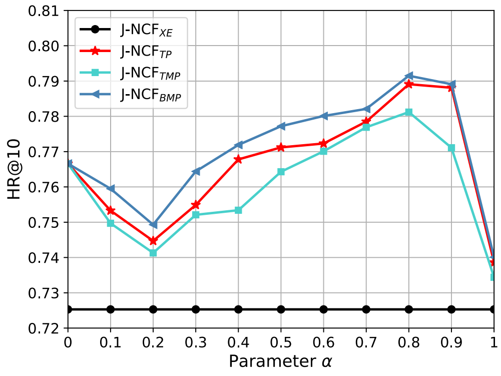
(b) Performance in terms of NDCG@10 on the ML100K dataset.



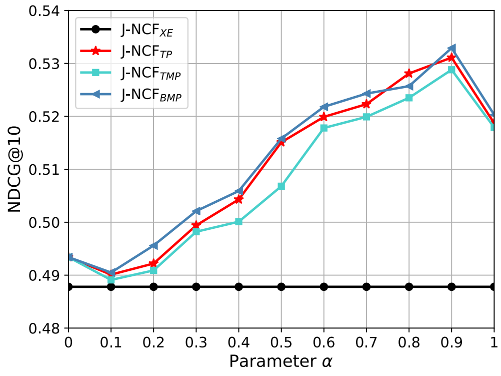
(c) Performance in terms of HR@10 on the ML1M dataset.



(d) Performance in terms of NDCG@10 on the ML1M dataset.



(e) Performance in terms of HR@10 on the AMovies dataset.



(f) Performance in terms of NDCG@10 on the AMovies dataset.

Fig. 3. Performance of the J-NCF models applied with different loss functions where the parameter α in Equation (15) ranges from 0 to 1 with a step size of 0.1.

As for the overall performance, we can see that when applied with a list-wise loss function, J-NCF$_{XE}$ has the worst performance among the four models. The other three models, which combine pair-wise and point-wise losses, show relatively similar results in terms of HR@10 and NDCG@10. When $\alpha = 0$, it results in J-NCF$_{point}$. When $\alpha = 1$, it leads to J-NCF, a model with only corresponding pair-wise loss functions. It is obvious that solely based on point-wise loss, J-NCF has better performance in terms of HR@10 while worse performance regarding NDCG@10 than J-NCF with only pair-wise loss. This can be explained by the fact that pair-wise loss can help J-NCF learn to rank items in right positions.

In Figure 3(a), the performance of all models increases from $\alpha = 0.2$ to $\alpha = 0.7$ before a short-term decrease and then a dramatic drop after reaching the peak at $\alpha = 0.7$. The performance of J-NCF$_{TP}$, J-NCF$_{TMP}$, and J-NCF$_{BMP}$ is comparable in terms of HR@10. As for NDCG@10, shown in Figure 3(b), J-NCF$_{TP}$ shows better performance than the other two models and achieves the highest point at $\alpha = 0.9$.

Regarding the performance on the ML1M dataset, similar trends can be found in Figure 3(c) and Figure 3(d) as in Figure 3(a) and Figure 3(b), respectively. For the AMovies dataset shown in Figure 3(e) and Figure 3(f), J-NCF$_{BMP}$ shows slightly better performance than both J-NCF$_{TP}$ and J-NCF$_{TMP}$ in terms of HR@10, while the performance of J-NCF$_{BMP}$ and J-NCF$_{TP}$ is similar in terms of NDCG@10, which is a little better than that of J-NCF$_{TMP}$.

As discussed in Reference [19], the BPR-max and TOP1-max loss functions have been proposed to overcome vanishing gradients as the number of negative samples increases. Since we use a small number of negative samples in our article, the performance is relatively similar between the three models J-NCF$_{TP}$, J-NCF$_{TMP}$, and J-NCF$_{BMP}$. As BPR-max and TOP1-max losses need additional softmax calculations for all negative samples, we apply the TOP1 pair-wise loss in Equation (15) for J-NCF in the experiments on which we report below.

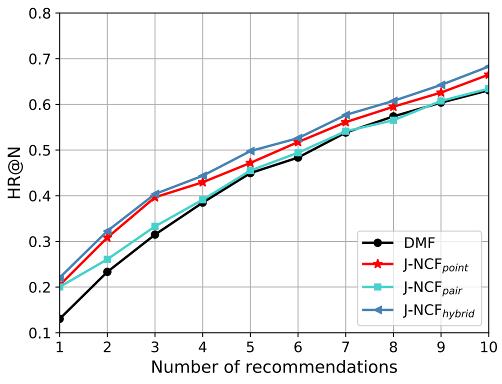
5.3 Utility of Hybrid Loss Function

For **RQ3**, to further investigate the utility of the hybrid loss function (Equation (15)), we examine the recommendation performance of the J-NCF$_c$ models under different settings, i.e., J-NCF$_{point}$ with only point-wise loss based on Equation (10) (we incorporate explicit feedback in the same way as Equation (16)), J-NCF$_{pair}$ with only pair-wise loss based on Equation (11), and J-NCF$_{hybrid}$ with our designed loss function from Equation (16). Figure 4 shows the results.

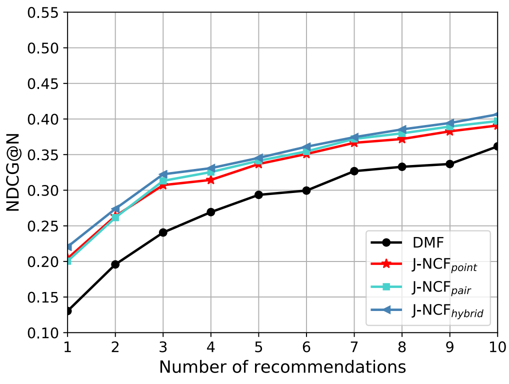
The overall performance in terms of HR and NDCG increases when the size of the top-N recommended list ranges from 1 to 10, as a large value of N increases the probability of including a user's preferred item in the recommendation list. J-NCF$_{hybrid}$ consistently achieves improvements over DMF as well as the two models with a single loss function across positions, which demonstrates the utility of our newly designed loss function. Based on the ML100K dataset, J-NCF$_{hybrid}$ improves by 2.68% and 7.61%, respectively, over J-NCF$_{point}$ and J-NCF$_{pair}$ in terms of HR@10; improvements of NDCG@10 over J-NCF$_{point}$ and J-NCF$_{pair}$ are 3.99% and 2.36%, respectively.

Comparing J-NCF$_{point}$ and J-NCF$_{pair}$, we find that J-NCF$_{point}$ beats J-NCF$_{pair}$ in terms of HR, while J-NCF$_{pair}$ shows more competitive performance in terms of NDCG than J-NCF$_{point}$. This confirms the findings in References [17, 37] that a pair-wise ranking-aware learner has a strong performance for ranking prediction. This finding motivates us to incorporate both point-wise loss and pair-wise loss into the hybrid loss function. Clearly, J-NCF$_c$-based models, i.e., J-NCF$_{point}$, J-NCF$_{pair}$, and J-NCF$_{hybrid}$, show a better performance than DMF, which also proves that the joint neural structure is effective, i.e., deep interaction modeling can optimize neural matrix factorization and thus improve the recommendation performance.

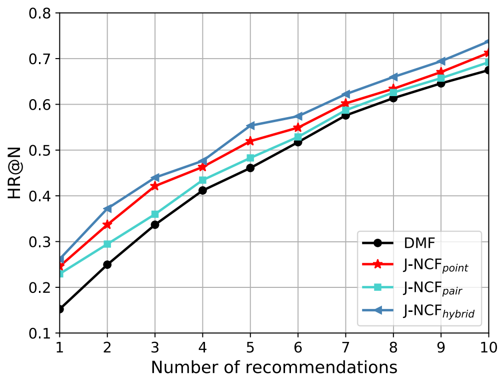
Comparing the left- and right-hand sides of Figure 4, we see that the improvements of J-NCF$_{hybrid}$ in terms of NDCG are more significant than those in terms of HR, as indicated by the



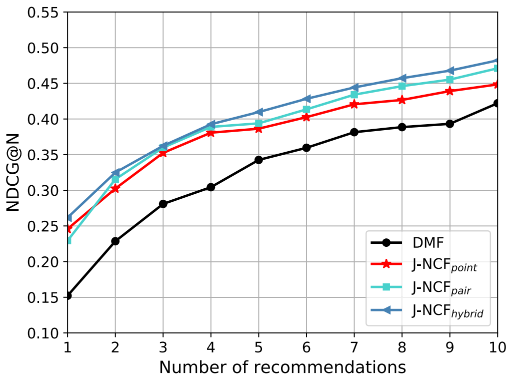
(a) Performance in terms of HR@N on the ML100K dataset.



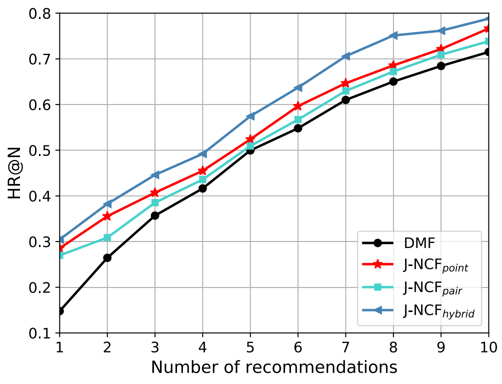
(b) Performance in terms of NDCG@N on the ML100K dataset.



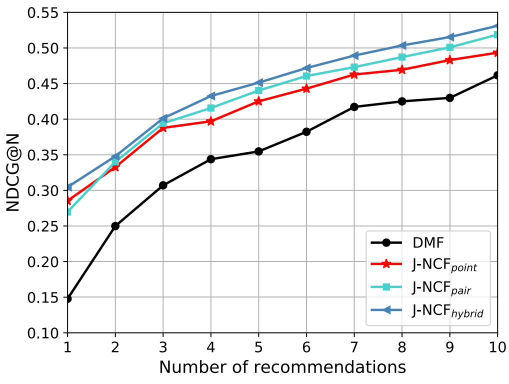
(c) Performance in terms of HR@N on the ML1M dataset.



(d) Performance in terms of NDCG@N on the ML1M dataset.



(e) Performance in terms of HR@N on the AMovies dataset.



(f) Performance in terms of NDCG@N on the AMovies dataset.

Fig. 4. Performance of Top-N item recommendation where N ranges from 1 to 10. The left and right plots show the performance in terms of HR@N and NDCG@N, respectively.

Table 6. Performance of J-NCF$_c$ and DMF with Different Numbers of Layers in Terms of HR@10 and NDCG@10

		HR@10					NDCG@10				
		DF-1	DF-2	DF-3	DF-4	DF-5	DF-1	DF-2	DF-3	DF-4	DF-5
ML100K	DI-1	.6242	.6511	.6713	.6955	.7213	.3581	.3721	.3971	.4123	.4313
	DI-2	.6351	.6642	.6829	.7183	.7388	.3694	.3899	.4067	.4277	.4426
	DI-3	.6493	.6712	.7144	.7309	.7479	.3811	.4001	.4197	.4388	.4535
	DI-4	.6571	.6832	.7277	.7411	**.7523**	.3945	.4183	.4311	.4481	**.4618**
	DI-5	.6501	.6799	.7254	.7408	.7501	.3903	.4111	.4287	.4433	.4587
	DMF	.6285	.6309	.6301	.6297	.6298	.3598	.3616	.3614	.3607	.3598
ML1M	DI-1	.6451	.6671	.7121	.7389	.7619	.3622	.3911	.4399	.4893	.5301
	DI-2	.6531	.6999	.7377	.7531	.7814	.3889	.4233	.4822	.5211	.5525
	DI-3	.6766	.7198	.7589	.7728	.7929	.4195	.4601	.5177	.5437	.5777
	DI-4	.7134	.7472	.7683	.7834	**.8088**	.4581	.5101	.5389	.5663	**.5906**
	DI-5	.7099	.7411	.7653	.7821	.8021	.4517	.5078	.5333	.5644	.5878
	DMF	.6673	.6748	.6738	.6722	.6725	.3955	.4221	.4201	.4197	.4199
AMovies	DI-1	.6611	.6922	.7481	.7911	.8188	.4041	.4533	.5004	.5413	.5622
	DI-2	.6872	.7378	.7881	.8101	.8411	.4327	.4911	.5311	.5597	.5803
	DI-3	.6989	.7633	.8078	.8378	.8787	.4632	.5204	.5501	.5714	.6102
	DI-4	.7414	.7999	.8293	.8612	**.8893**	.5137	.5461	.5644	.5966	**.6198**
	DI-5	.7379	.7922	.8201	.8589	.8821	.5111	.5402	.5599	.5934	.6145
	DMF	.7478	.7515	.7491	.7483	.7479	.4551	.4616	.4612	.4603	.4591

The results produced by the bestperforming setting on each dataset are boldfaced.

relative improvements over DMF with different sizes of the recommendation list. In Figure 4(a), J-NCF$_{hybrid}$ shows a 8.78% improvement over DMF in terms of HR at cutoff $N = 6$, a 5.91% improvement at $N = 8$ and an 8.24% improvement at $N = 10$ on the ML100K dataset. In Figure 4(b), the improvements in terms of NDCG at cutoff $N = 6$, $N = 8$, and $N = 10$ are 19.01%, 15.72%, and 12.42%, respectively. J-NCF$_c$ with the hybrid loss function cannot only recommend the correct item to a user, but is also competitive in terms of ranking it at the top of the list.

5.4 Number of Layers in the Networks

In J-NCF$_c$, we not only learn features of users and items through the DF neural network with multiple hidden layers, but also model user-item interactions with multi-layer perceptrons in the DI network. Thus, it is crucial to see whether DL is helpful in our model. We conduct experiments to examine the performance of J-NCF$_c$ with various numbers of layers in the DF and DI networks, respectively. In addition, we also test the performance of the best baseline model, i.e., DMF, with different DF networks. The results are shown in Table 6. The i in DF-i and DI-i in Table 6 denotes the number of layers in the DF network and DI network of J-NCF$_c$, respectively.

As shown in Table 6, in terms of HR@10, we can see that with the number of layers increasing, the recommendation performance of J-NCF is improved, which verifies the effectiveness of DL techniques for recommender systems.

Comparing the number of layers in the DI and DF networks, we can find that stacking more layers in the DF network of J-NCF$_c$ seems more helpful than in the DI network in enhancing the recommendation performance. For example, based on the ML100K dataset, the improvements of the configuration (DF-3, DI-2) over (DF-2, DI-2) are 2.82% and 4.31% in terms of HR@10 and NDCG@10, while the improvements are 1.05% and 2.62% for (DF-2, DI-3) over (DF-2, DI-2). When we stack more than 4 layers in the DI network (e.g., DI-5), the performance of J-NCF$_c$ no longer increases. However, stacking more layers in the DF network (e.g., DF-5) still seems helpful, and the best results produced for each dataset are all based on J-NCF$_c$ with the (DF-5, DI-4) configuration. This may be because deep layers are more helpful in extracting users' as well as items' features and thus enhance the user-item interaction predictions. It motivates us to incorporate more auxiliary information for exploring users' and items' features with deep learning techniques in future work.

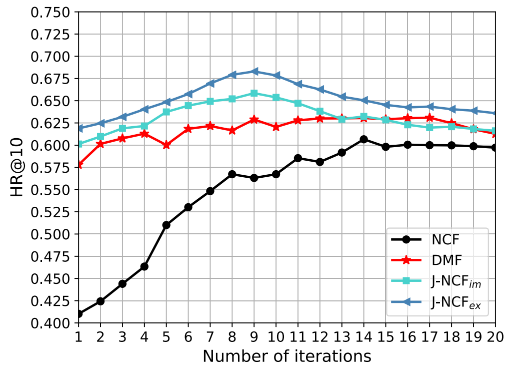
As for NDCG@10, a similar phenomenon can be found. However, when comparing the scores of HR@10 and NDCG@10 under the same configurations, we can find that deeper layers can lead to more obvious improvements in terms of NDCG@10 than HR@10 on all of the three datasets. The best performance of J-NCF with (DF-5, DI-4) outperforms the worst performance of J-NCF with (DF-1, DI-1) by 20.52%, 25.37%, and 34.52% in terms of HR@10 on the three datasets, respectively. However, the improvements are 28.96%, 63.05%, and 53.37% in terms of NDCG@10 on the three datasets.

As for the baseline model DMF shown in the bottom rows in Table 6, when applied with DF-1, J-NCF$_c$ with DI-1 loses to DMF on all datasets. Similar results can be found with DF-2, except on the ML100K dataset. This can be explained by the fact that the simple concatenation of users' and items' embeddings with only one MLP layer in J-NCF$_c$ is not efficient for modeling user-item interactions. When applied with more DI layers, J-NCF$_c$ has better performance than DMF with the same number of DF layers. Additionally, we can find that DMF achieves the best performance with DF-2, and deeper layers do not seem useful for the DMF model, which corresponds to the results in Reference [22]. However, J-NCF$_c$ achieves further improvements when stacking more layers in either the DI or DF network or both.

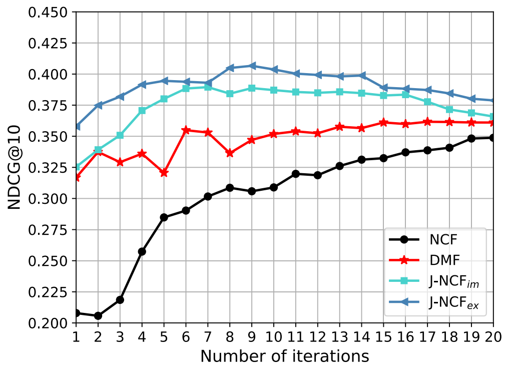
5.5 Impact of Feedback

In J-NCF, we consider different kinds of user feedback. On the one hand, we use the interaction matrix as the input of the network with Equation (3), which contains not only implicit feedback but also explicit feedback. On the other hand, our loss function in Equation (16) employs a normalized strategy in the form of $Y_{ui} = \frac{y_{ui}}{Max(R_u)}$, where $Max(R_u)$ denotes the largest rating score of user u given to items, to incorporate the explicit feedback. To answer **RQ5**, we conduct experiments to investigate whether the combination of explicit and implicit feedback works for J-NCF with different settings, i.e., J-NCF$_{ex}$ with both kinds of feedback in the input and the loss function as well as J-NCF$_{im}$ with only implicit feedback by labeling 1 for the interactions and 0 for unknown ratings in the input and the loss function. Figure 5 shows the recommendation performance of J-NCF$_{ex}$, J-NCF$_{im}$, DMF, and NCF across different numbers of training iterations, respectively.

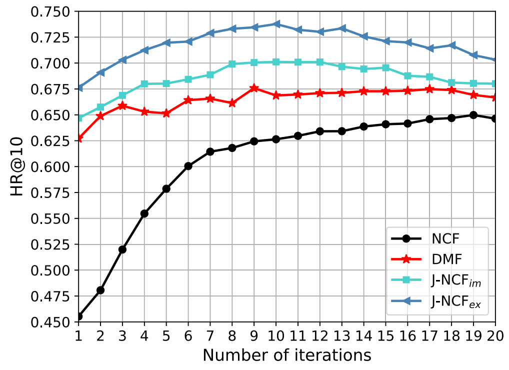
First, from Figure 5, we can see that J-NCF$_{ex}$ with both kinds of feedback achieves a competitive performance across all iterations in terms of HR@10 and NDCG@10 on the three datasets. It indicates that the combination of explicit and implicit feedback in the input and the specially designed loss function of J-NCF does help to improve the recommendation performance. Second, as the number of training iterations increases, the recommendation performance of all models is improved and then degraded after reaching a peak. More iterations may lead to overfitting, which hurts the recommendation performance. However, comparing J-NCF model with the baselines, i.e., DMF and NCF, we find that J-NCF converges to the best performance faster than other models. For example, on the ML100K dataset, the best result of J-NCF is generated after the first 9 effective iterations, while DMF and NCF need more training iterations to obtain the best results, i.e., 16 and 14 iterations, respectively. The same phenomenon can be observed on the other two datasets. The optimal number of updates needed for J-NCF, DMF, and NCF are around 10, 17, and 19 on the ML1M dataset, and 14, 18, and 19 on the AMovies dataset, respectively. Third, comparing the performance in terms of HR@10 and NDCG@10, we find that J-NCF$_{ex}$ shows larger improvements over J-NCF$_{im}$



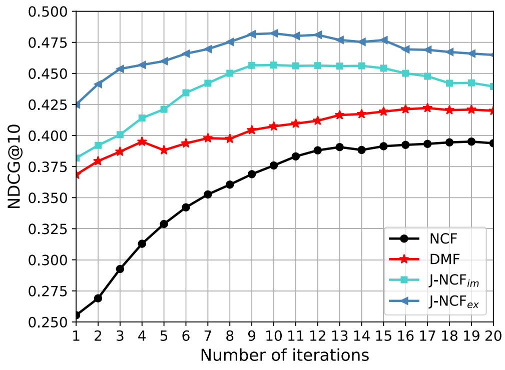
(a) Performance in terms of HR@10 on the ML100K datasets.



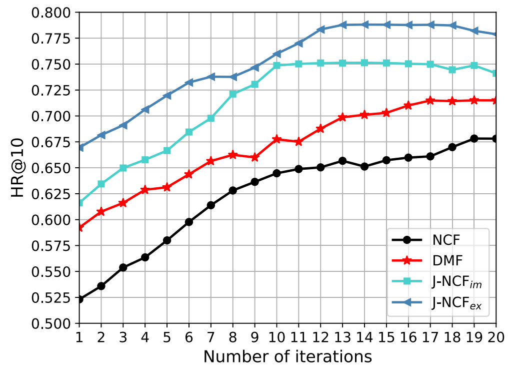
(b) Performance in terms of NDCG@10 on the ML100K datasets.



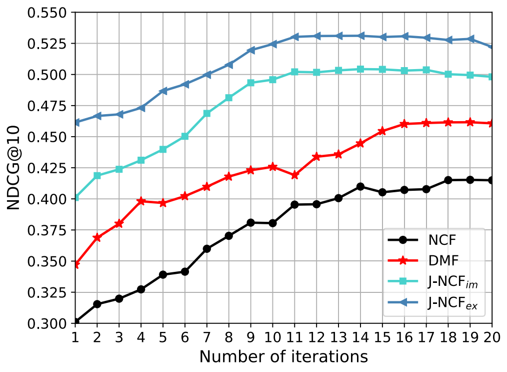
(c) Performance in terms of HR@10 on the ML1M datasets.



(d) Performance in terms of NDCG@10 on the ML1M datasets.



(e) Performance in terms of HR@10 on the AMovies datasets.



(f) Performance in terms of NDCG@10 on the AMovies datasets.

Fig. 5. Recommendation performance across different numbers of iterations. The left and right plots show the performance in terms of HR@10 and NDCG@10, respectively.

Table 7. Recommendation Performance across Users Who Are Ranked by the Number of Activities

		HR@10			NDCG@10		
		DMF	J-NCF$_m$	J-NCF$_c$	DMF	J-NCF$_m$	J-NCF$_c$
ML100K	10%	.7001	.7400$^{\blacktriangle}$	**.8015**$^{\blacktriangle}$	.4358	.4786$^{\blacktriangle}$	**.5001**$^{\blacktriangle}$
	50%	.6813	.7349$^{\vartriangle}$	**.7568**$^{\blacktriangle}$	.4200	.4379$^{\vartriangle}$	**.4602**$^{\blacktriangle}$
	90%	.6279	.6585$^{\vartriangle}$	**.6772**$^{\blacktriangle}$	.3813	.3897$^{\vartriangle}$	**.4092**$^{\blacktriangle}$
ML1M	10%	.7548	.7927$^{\blacktriangle}$	**.8511**$^{\blacktriangle}$	.5111	.5417$^{\blacktriangle}$	**.5952**$^{\blacktriangle}$
	50%	.7211	.7532$^{\blacktriangle}$	**.7982**$^{\blacktriangle}$	.4855	.5266$^{\blacktriangle}$	**.5587**$^{\blacktriangle}$
	90%	.6601	.6981$^{\blacktriangle}$	**.7277**$^{\blacktriangle}$	.4217	.4432$^{\blacktriangle}$	**.4751**$^{\blacktriangle}$
AMovies	10%	.7851	.8611$^{\blacktriangle}$	**.9191**$^{\blacktriangle}$	.5349	.5998$^{\blacktriangle}$	**.6611**$^{\blacktriangle}$
	50%	.7519	.7855$^{\blacktriangle}$	**.8411**$^{\blacktriangle}$	.5033	.5466$^{\blacktriangle}$	**.5821**$^{\blacktriangle}$
	90%	.7013	.7411$^{\blacktriangle}$	**.7732**$^{\blacktriangle}$	.4597	.5038$^{\blacktriangle}$	**.5301**$^{\blacktriangle}$

The results produced by the best performing recommender system in each row are boldfaced. Statistical significance of pair-wise differences of J-NCF$_m$ and J-NCF$_c$ vs. DMF is determined by a t-test ($^{\blacktriangle}/^{\blacktriangledown}$ for $\alpha = .01$, or $^{\vartriangle}/^{\triangledown}$ for $\alpha = .05$).

in terms of NDCG@10 than HR@10. For example, the improvements are 3.72%, 5.22%, and 4.89% in terms of HR@10, on the ML100K, ML1M, and AMovies datasets, respectively, vs. improvements of 4.61%, 5.58%, and 5.31% in terms of NDCG@10. This confirms our hypothesis that incorporating both explicit and implicit feedback can improve the ranking precision for recommendation.

6 SCALABILITY AND SENSITIVITY

To answer **RQ6** to **RQ9**, we study the scalability and sensitivity of J-NCF as well as the best baseline DMF when applied in different settings, i.e., with users with various numbers of ratings in Section 6.1, and with datasets with different levels of sparsity in Section 6.2. In addition, we also investigate the performance of the deep learning–based approaches, i.e., J-NCF, DMF, and NCF, when applied with a large and sparse dataset in Section 6.3. Moreover, the training and inference time needed for these models on all datasets is discussed in Section 6.4.

6.1 Model Scalability with User Ratings

In Figure 2, we have shown that in every dataset most users only have a few ratings, thus it is meaningful to investigate how the performance of J-NCF and DMF varies with different numbers of user ratings. Following Reference [36], we look at the performance for users of varying degrees of activity, measured by percentile. For example, in Table 7, we first rank the users according to their numbers of their activities. 10% shows the mean performance across the bottom 10% of users, who are least active; the 90% mark shows the mean performance for all but the top 10% most active users.

As shown in Table 7, J-NCF$_c$ outperforms the best baseline model DMF for users across all activity levels, i.e., both the "inactive" users who constitute the majority and the relatively few "very active" users who give more ratings. In addition, J-NCF$_c$ always achieves the best performance in terms of HR@10 and NDCG@10. To test the robustness of J-NCF under different settings, i.e., J-NCF$_c$ and J-NCF$_m$, we conduct t-tests between the two versions of J-NCF with DMF, respectively. Significant improvements against the baseline DMF in terms of HR@10 and NDCG@10 are observed for both J-NCF$_m$ and J-NCF$_c$ at the $\alpha = .01$ level across all activity levels, except for J-NCF$_m$ on the ML100K dataset with 50% and 90% users, for which we observe significant improvements at the $\alpha = .05$ level in terms of HR@10 and NDCG@10.

Specifically, J-NCF shows larger improvements over the DMF model for "inactive" users than for "very active" users. For example, when incorporating users with more interactions, i.e., from 50% to 90%, the improvements change from 11.08% to 7.85% in terms of HR@10, and 9.57% to 7.32% in terms of NDCG@10 on the ML100K dataset. This may be because the "very active" users have many interactions with the items that have few ratings and collaborative filtering lacks information for recommending items based only on the rating matrix. This naturally suggests a line of future work in which one would extend J-NCF with more auxiliary information, such as content information, to explore more accurate relationships between items.

To conclude and answer **RQ6**, the J-NCF models can beat the best baseline model for users across all activity levels. J-NCF$_c$ shows the best performance in all datasets. In addition, for "inactive" users, J-NCF shows larger improvements over DMF than for "very active" users.

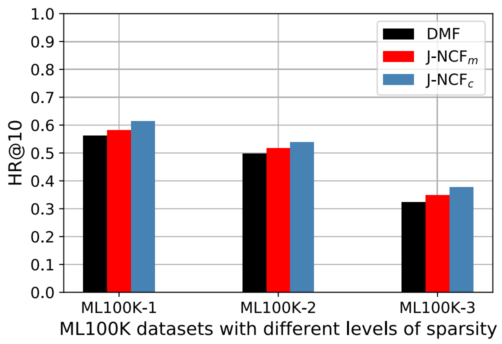
6.2 Sensitivity to Data Sparsity

To investigate the sensitivity of J-NCF to different levels of data sparsity, we examine the recommendation performance on datasets with different levels of sparsity, as presented in Table 4. Figure 6 shows the results. The overall performance of all models on the AMovies dataset is better than that on the other two datasets. That is to say, the recommendation performance may be influenced by the size of a dataset. Thus, to investigate the model sensitivity across datasets with different degrees of sparsity, it is essential to keep the number of users and items in the same scale for the datasets.

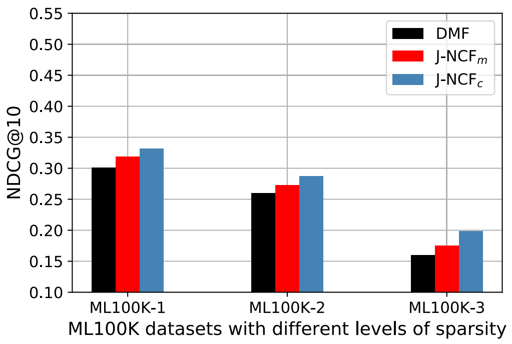
From Figure 6, in particular, for the ML100K dataset, the ML1M dataset, and the AMovies dataset, we see that the J-NCF models outperform the baseline model DMF across all sub datasets with different degrees of sparsity in terms of HR@10 and NDCG@10. In addition, we find that when the density of those datasets goes down, the performance of all models decreases. Thus, it is interesting to investigate the robustness of J-NCF when it is applied to sparse datasets. We find that when applied on small datasets, e.g., subsets of ML100K, our best model, i.e., J-NCF$_c$, shows higher improvements against DMF on sparser datasets. For example, J-NCF$_c$ achieves 4.91% and 9.12% improvements over DMF in terms of HR@10 and NDCG@10 on the ML100K-1 subset (Density = 4.413%), while the improvements on the ML100K-3 subset (Density = 0.630%) are 7.77% and 12.02% in terms of HR@10 and NDCG@10, respectively. However, when applied on larger datasets with more users and items, i.e., subsets of ML1M and AMovies, J-NCF$_c$ shows higher improvements against DMF on denser datasets. For instance, J-NCF$_c$ achieves 11.13% improvements over DMF in terms of HR@10 on the ML1M-1 subset (Density = 3.7982%), while the improvements on the ML1M-3 subset (Density = 0.7499%) are 6.53% in terms of HR@10. These results may indicate that when the dataset becomes larger and sparser, it will be more difficult for models to improve their recommendation performances, which motivates us to conduct a further investigation to answer **RQ8** (see Section 6.3 below).

In addition, comparing the left- and right-hand side plots in Figure 6, we find that J-NCF$_c$ shows a better performance in terms of NDCG@10 than HR@10. For example, the improvements of J-NCF$_c$ over DMF are 9.19%, 8.28%, and 15.11% in terms of HR@10 on ML100K-1, ML100K-2, and ML100K-3 datasets, respectively, while the improvements are 10.11%, 10.65%, and 20.55% in terms of NDCG@10. This result is consistent with our findings in Section 5.3.

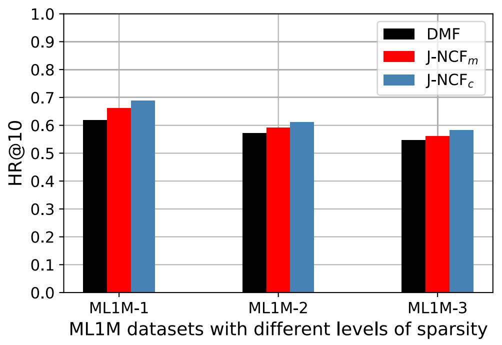
Thus, in answer to **RQ7**, the J-NCF models outperform the best baseline model DMF across all datasets with different degrees of sparsity in terms of both metrics. Specifically, when applied on large datasets, i.e., ML1M and AMovies, J-NCF$_c$ shows higher improvements against DMF on denser datasets. In addition, the improvements of J-NCF$_c$ over DMF in terms of NDCG@10 are larger than in terms of HR@10.



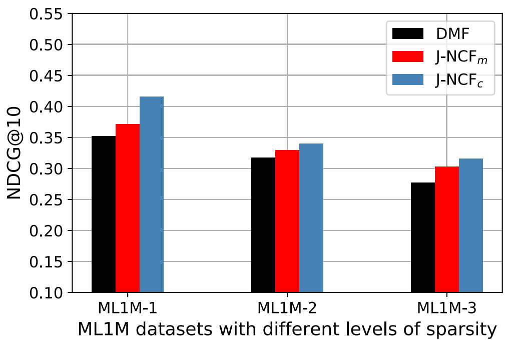
(a) Performance in terms of HR@10 on the ML100K datasets.



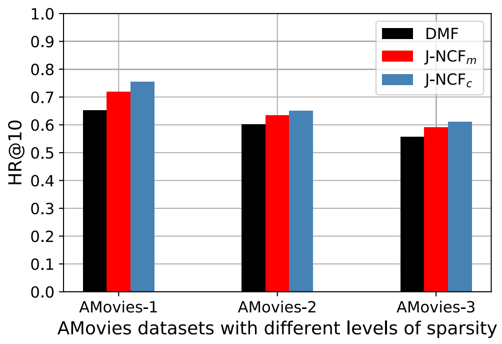
(b) Performance in terms of NDCG@10 on the ML100K datasets.



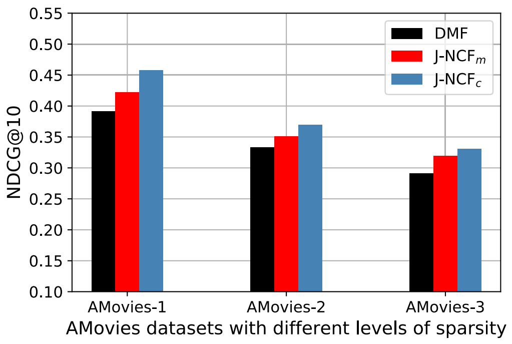
(c) Performance in terms of HR@10 on the ML1M datasets.



(d) Performance in terms of NDCG@10 on the ML1M datasets.



(e) Performance in terms of HR@10 on the AMovies datasets.

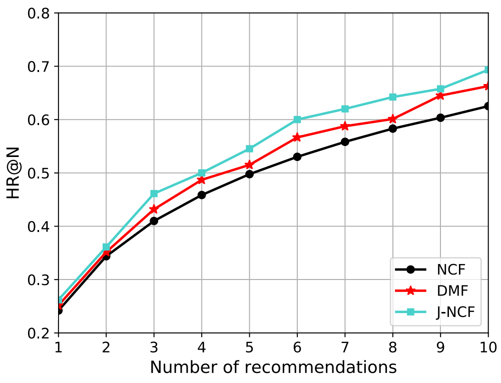


(f) Performance in terms of NDCG@10 on the AMovies datasets.

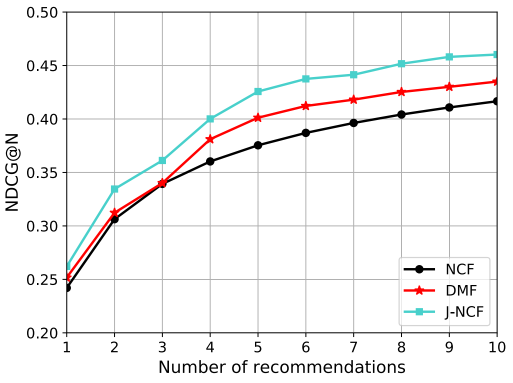
Fig. 6. Recommendation performance across datasets with different levels of sparsity. The left and right plots show the performance in terms of HR@10 and NDCG@10, respectively.

6.3 Performance with a Large and Sparse Dataset

For **RQ8**, to see if our model is able to work well on a large and sparse dataset, we examine our model as well as two baseline models, i.e., NCF and DMF, on the Amazon Electronic (AEle) dataset, which is larger and sparser than the MovieLens and Amazon Movies datasets. Figure 7 shows the performance of the three models with different sizes of top-N recommended lists.



(a) Performance in terms of HR@N on AEle dataset.



(b) Performance in terms of NDCG@N on AEle dataset.

Fig. 7. Performance of Top-N item recommendation where N ranges from 1 to 10, tested on AEle dataset.

Table 8. Training and Prediction Time Needed for Baseline Models as Well as J-NCF on All Datasets

		Training		Prediction	
		Total time(s)	Average epoch(s)	Total time(s)	Average ranking(s)
ML100K	NCF	46.344	1.943	1.389	0.00147
	DMF	180.017	9.587	1.558	0.00165
	J-NCF	116.023	10.925	1.607	0.00170
ML1M	NCF	494.038	17.751	8.251	0.00137
	DMF	5,451.671	320.687	12.376	0.00205
	J-NCF	3,539.059	340.048	13.858	0.00229
AMovies	NCF	977.265	25.836	25.599	0.00170
	DMF	39,249.657	2,180.537	34.955	0.00232
	J-NCF	31,414.628	2,206.084	37.818	0.00251
AEle	NCF	61,812.187	326.828	2,919.005	0.00239
	DMF	788,138.604	43,785.478	4,360.187	0.00357
	J-NCF	723,586.192	45,224.137	4,775.443	0.00391

It is clear that J-NCF outperforms DMF as well as NCF in terms of HR and NDCG across different numbers of recommendations. With the size of top-N recommended lists ranging from 1 to 10, the overall performances of all models increase, which is consistent with the conclusion in Section 5.3. Comparing the results shown in Figure 7(a) and Figure 7(b), the improvements of J-NCF over DMF in terms of NDCG are more significant than those in terms of HR. For example, when $N = 5$ and $N = 10$, the improvements of J-NCF over DMF in terms of HR are 5.88% and 4.62%, while the improvements are 6.12% and 5.82% in terms of NDCG, respectively. To conclude and answer **RQ8**, J-NCF can also work well with large and sparse datasets, especially in ranking items correctly.

6.4 Training and Inference Time

To answer **RQ9**, we investigate the scalability of J-NCF regarding training and inference time in Table 8. As shown in Table 8, in the "Training" part, "Total time" denotes the time needed for training the model to the best performance. And the "Average epoch" means the average training time for a single epoch in the training process. In the "Prediction" part, "Total time" denotes the prediction time needed for the whole test set. Since the test set contains the latest interaction of every user, the "Average ranking" indicates the time needed for providing a ranked list containing top-10 recommendations for a single user.

As we can see in Table 8, when the size of the dataset becomes larger, the time needed for both training and prediction gets increased significantly for all models. NCF consistently costs the least time among the three models for both training and prediction processes on all datasets. For the training process, the average training time for one epoch of J-NCF is slightly higher than DMF. However, the total training time for J-NCF is less than for DMF. It can be explained by the fact that J-NCF needs fewer iterations to obtain the best results than DMF, as indicated in Section 5.5. Thus, J-NCF costs less time for training to the best performance than DMF. For the prediction process, although the total time needed for J-NCF and DMF is more than NCF, the three models cost roughly similar amounts of time for providing a top-10 ranked list for a single user, which is around a few milliseconds.

7 CONCLUSIONS AND FUTURE WORK

We have proposed a joint neural collaborative filtering model, J-NCF, for recommender systems. J-NCF uses a unified deep neural network to tightly couple two important parts in a recommender system, i.e., deep feature learning of users and items, and deep modeling of user-item interactions. For the user and item feature extraction, we use a deep neural network with matrix factorization and a combination of explicit and implicit feedback as input. Then, we adopt another neural network for modeling user-item interactions using the feature vectors as inputs. Thus, J-NCF enables the two parts to be optimized with each other through a joint training process. To make J-NCF fit the top-N recommendation task, we design a new loss function that incorporates information from both pair-wise and point-wise loss.

The experimental results confirm the effectiveness of J-NCF. In addition, we have also experimentally investigated the performance of J-NCF under various settings, e.g., with different loss functions, with varying numbers of layers in the networks, and with using different feedback as inputs. The results confirm the effectiveness of our hybrid loss function and demonstrate that J-NCF performs better with more layers in the networks and using the combination of implicit and explicit feedback as input.

In addition, we have investigated the robustness of J-NCF with different degrees of data sparsity and different numbers of user ratings. J-NCF outperforms the best baseline model DMF for users across all activity levels, especially for "inactive users" who constitute the majority of users in the datasets. As for datasets with different levels of sparsity, in general, J-NCF shows more competitive recommendation performance on all datasets than the state-of-the-art baseline model DMF. Moreover, we have also tested the J-NCF model with a large and sparse dataset, i.e., AEle, and the results show that J-NCF also outperforms state-of-the-art baseline models on the dataset.

As to future work, first, we plan to extend J-NCF with more auxiliary information [5, 6, 50, 55], such as the content information of items as well as reviews, to get a more informed expression of users as well as items. As collaborative filtering usually suffers from limited performance due to the sparsity of user-item interactions [43], auxiliary information could be used to boost the performance. It would also be interesting to explore heterogeneous information in a knowledge base to improve the quality of recommender systems with deep learning [53]. Second, we plan to explore the context information of a user in a session with recurrent neural networks to deal with dynamic aspects recommender systems [7–9, 21]. In addition, an attention mechanism could be applied to J-NCF, which can filter out uninformative content and select the most representative items while providing good interpretability [10]. Finally, as we have found that J-NCF is computationally more expensive than NCF, we plan to optimize the structure and implementation details of our model to make it more efficient.

ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers for their helpful comments and valuable suggestions.

REFERENCES

[1] David Adedayo Adeniyi, Zhaoqiang Wei, and Yongquan Yang. 2016. Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method. *Appl. Comput. Inform.* 12, 1 (2016), 90–108.

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* 17, 6 (2005), 734–749.

[3] Basiliyos Tilahun Betru, Charles Awono Onana, and Batchakui Bernabe. 2017. Deep learning methods on recommender system: A survey of state-of-the-art.*Int. J. Comput. Appl.* 162, 10 (2017), 17–22.

[4] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. 2011. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'11).* ACM, 333–336.

[5] Fei Cai and Maarten de Rijke. 2016. Learning from homologous queries and semantically related terms for query auto completion. *Inform. Proc. Manag.* 52, 4 (2016), 628–643.

[6] Fei Cai and Maarten de Rijke. 2016. A survey of query auto completion in information retrieval. *Found. Trends Inform. Retr.* 10, 4 (2016), 273–363.

[7] Fei Cai, Shangsong Liang, and Maarten de Rijke. 2016. Prefix-adaptive and time-sensitive personalized query auto completion. *IEEE Trans. Knowl. Data Eng.* 28, 9 (Sep. 2016), 2452–2466.

[8] Fei Cai, Ridho Reinanda, and Maarten de Rijke. 2016. Diversifying query auto-completion. *ACM Trans. Inform. Syst.* 34, 4 (June 2016), 25:1–25:33.

[9] Sotirios P. Chatzis, Panayiotis Christodoulou, and Andreas S. Andreou. 2017. Recurrent latent variable networks for session-based recommendation. In *Proceedings of theWorkshop on Deep Learning for Recommender Systems (DLRS'17).* 38–45.

[10] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *Proceedings of theInternational ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17).* ACM, 335–344.

[11] Wanyu Chen, Fei Cai, Honghui Chen, and Maarten de Rijke. 2018. Attention-based hierarchical neural query suggestion. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'18).* ACM, 1093–1096.

[12] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. In *Proceedings of theWorkshop on Deep Learning for Recommender Systems (DLRS 2016).* ACM, 7–10.

[13] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on Top-N recommendation tasks. In *Proceedings of theACM Conference on Recommender Systems (RecSys'10).* ACM, 39–46.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A factorization-machine based neural network for CTR prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17).* AAAI Press, 1725–1731.

[15] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of theInternational ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17).* ACM, 355–364.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of theInternational World Wide Web Conferences (WWW'17).* ACM, 173–182.

[17] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of theInternational ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'16).* ACM, 549–558.

[18] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inform. Syst.* 22, 1 (2004), 5–53.

[19] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'18).* ACM, 843–852.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*.

[21] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'16)*. ACM, 241–248.

[22] Xue Hong-Jian, Dai Xinyu, Zhang Jianbing, Huang Shujian, and Chen Jiajun. 2017. Deep matrix factorization models for recommender systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17)*. 3203–3209.

[23] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of theInternational Conference on Information and Knowledge Management (CIKM'13)*. ACM, 2333–2338.

[24] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored item similarity models for Top-N recommender systems. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, 659–667.

[25] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. 2016. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of theACM Conference on Recommender Systems (RecSys'16)*. 233–240.

[26] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Retrieved from: *arXiv preprint arXiv:1412.6980*.

[27] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, 426–434.

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

[29] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'15)*. ACM, 811–820.

[30] Jianxun Lian, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2017. CCCFNet: A content-boosted collaborative filtering neural network for cross domain recommender systems. In *Proceedings of the International World Wide Web Conferences (WWW'17)*. ACM, 817–818.

[31] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.* 7, 1 (2003), 76–80.

[32] Juntao Liu and Caihua Wu. 2017. Deep learning based recommendation: A survey. In *Proceedings of theInternational Conference on Information Science and Applications (ICISA'17)*. 451–458.

[33] Xiaomeng Liu, Yuanxin Ouyang, Wenge Rong, and Zhang Xiong. 2015. Item category aware conditional restricted Boltzmann machine based recommendation. In *Proceedings of the International Conference on Neural Information Processing (ICONIP'15)*. 609–616.

[34] Kezban Dilek Onal, Ye Zhang, Ismail Sengor Altingovde, Md Mustafizur Rahman, Pinar Karagoz, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, Tyler McDonnell, An Thanh Nguyen, Dan Xu, Byron C. Wallace, Maarten de Rijke, and Matthew Lease. 2018. Neural information retrieval: At the end of the early years. *Inform. Retr. J.* 21, 2–3 (June 2018), 111–182.

[35] Arkadiusz Paterek. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'07)*.

[36] Gopalan Prem, Jake M. Hofman, and David M. Blei. 2013. Scalable recommendation with Poisson factorization. Retrieved from: *arXiv preprint arXiv:1311.1704*.

[37] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of theConference on Uncertainty in Artificial Intelligence (UAI'09)*. 452–461.

[38] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic matrix factorization. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'07)*. Curran Associates Inc., 1257–1264.

[39] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of theInternational Conference on Machine Learning (ICML'07)*. 791–798.

[40] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International World Wide Web Conferences (WWW'01)*. ACM, 285–295.

[41] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Thomas Riedl. 2000. Application of dimensionality reduction in recommender system–A case study. In *Proceedings of the ACM WebKDD Workshop*.

[42] Suvash Sedhain, Aditya Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders meet collaborative filtering. In *Proceedings of the International World Wide Web Conferences (WWW'15)*. ACM, 111–112.

[43] Lei Shi, Wayne Xin Zhao, and Yi-Dong Shen. 2017. Local representative-based matrix factorization for cold-start recommendation. *ACM Trans. Inform. Syst.* 36, 2 (Aug. 2017), 22:1–22:28.

[44] Xiaoyuan Su and Taghi M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Adv. Artific. Intell.* 2009 Article 421425 (2009), 19 pages. https://doi.org/10.1155/2009/421425.

[45] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the GRU: Multi-task learning for deep text recommendations. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'16)*. 107–114.

[46] Tran The Truyen, Dinh Q. Phung, and Svetha Venkatesh. 2009. Ordinal Boltzmann machines for collaborative filtering. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'09)*. 548–556.

[47] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'13)*. 2643–2651.

[48] Chong Wang and David M. Blei. 2011. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'11)*. 448–456.

[49] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, 1235–1244.

[50] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *Proceedings of the International World Wide Web Conferences (WWW'17)*. 391–400.

[51] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of theInternational ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*.

[52] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for Top-N recommender systems. In *Proceedings of the Conference on Web Search and Data Mining (WSDM'16)*. ACM, 153–162.

[53] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 353–362.

[54] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep learning based recommender system: A survey and new perspectives. Retrieved from: *arXiv preprint arXiv:1707.07435*.

[55] Lei Zheng, Vahid Noroozi, and Philip S. Yu.2017. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Conference on Web Search and Data Mining (WSDM'17)*. ACM, 425–434.

[56] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A neural autoregressive approach to collaborative filtering. In *Proceedings of the International Conference on Machine Learning (ICML'16)*. 764–773.

Received September 2018; revised April 2019; accepted July 2019