Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation

Shaoyun Shi[1], Weizhi Ma[1], Min Zhang[1]*, Yongfeng Zhang[2], Xinxing Yu[3], Houzhi Shan[3],
Yiqun Liu[1], and Shaoping Ma[1]
[1]Department of Computer Science and Technology, Institute for Artificial Intelligence,
Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China
[2]Department of Computer Science, Rutgers University, NJ, USA
[3]Zhihu, Beijing, China
shisy17@mails.tsinghua.edu.cn, z-m@tsinghua.edu.cn

ABSTRACT

Modeling large scale and rare-interaction users are the two major challenges in recommender systems, which derives big gaps between researches and applications. Facing to millions or even billions of users, it is hard to store and leverage personalized preferences with a user embedding matrix in real scenarios. And many researches pay attention to users with rich histories, while users with only one or several interactions are the biggest part in real systems. Previous studies make efforts to handle one of the above issues but rarely tackle efficiency and cold-start problems together.

In this work, a novel user preference representation called Preference Hash (PreHash) is proposed to model large scale users, including rare-interaction ones. In PreHash, a series of buckets are generated based on users’ historical interactions. Users with similar preferences are assigned into the same buckets automatically, including warm and cold ones. Representations of the buckets are learned accordingly. Contributing to the designed hash buckets, only limited parameters are stored, which saves a lot of memory for more efficient modeling. Furthermore, when new interactions are made by a user, his buckets and representations will be dynamically updated, which enables more effective understanding and modeling of the user. It is worth mentioning that PreHash is flexible to work with various recommendation algorithms by taking the place of previous user embedding matrices. We combine it with multiple state-of-the-art recommendation methods and conduct various experiments. Comparative results on public datasets show that it not only improves the recommendation performance but also significantly reduces the number of model parameters. To summarize, PreHash has achieved significant improvements in both efficiency and effectiveness for recommender systems.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

User Preference Modeling, Recommender System, Neural Recommendation, Cold Start Problem, Efficiency and Effectiveness

ACM Reference Format:
Shaoyun Shi, Weizhi Ma, Min Zhang, Yongfeng Zhang, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2020. Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401119

1 INTRODUCTION

Recommender system has become essential in various web services. However, due to millions or even billions of users and recommendation candidates every day, it is not easy to design an efficient and effective recommendation algorithm in real scenarios.

The personalized preference feature vector for each user is the key to the recommendation. Most of the previously proposed models generate a preference feature vector for each user and store them together as a feature matrix. For example, Matrix Factorization (MF) based Collaborative Filtering (CF) methods usually represent each user as a specific vector [16, 22, 23]. Some recent deep recommendation models also use an embedding layer to map each user to a preference vector [14, 25, 39] (the feature matrix is in the embedding layer). Most of them achieved excellent performance.

However, these methods can hardly apply to real-world recommender systems due to the massive number of users. Different from CF models proposed in research, most of the current deep neural models in large-scale real-world recommender systems are feature-based models, like Wide&Deep [5] proposed by Google and DeepFM [8] by Huawei. The reason is that storing such a user embedding matrix has too many parameters, which brings unacceptable memory cost and significantly slows down the model. On the other hand, there are always users with few interactions (Cold-Start Problem). Feature vectors of cold users may result in poor effectiveness, and for most algorithms, we cannot update users’ feature vectors unless retraining the whole model.

In previous studies, considering the efficiency, some models, such as FISM [20] and AutoRec [33], represent each user as a set of items

SIGIR ’20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401119

* Corresponding Author

This work is supported by the National Key Research and Development Program of China (2018YFC0831900) and Natural Science Foundation of China (Grant No. 61672311, 61532011). Dr Weizhi Ma has been supported by Shuimu Tsinghua Scholar Program.

he/she has interacted. For example, a simple way to form the user preference vector is by summing up vectors of items the user has interacted with. But in the cold scenario, there is limited interaction history to capture user preferences. To tackle this problem, some researchers try to utilize some side information, such as content features [34] and social information [18], to learn a representation of the user. However, side information is not always available and cannot fully show his/her personal preferences. Some hybrid algorithms combine content features with personalized user vectors (usually stored in a user preference matrix), while they still have unacceptable memory and time cost.

Different from previous studies that focus on handling one of the challenges, we propose a user preference representation methodology called Preference Hash (PreHash) to tackle the two problems together. It is not an independent but a flexible module that takes the place of user embedding matrices in various previous models. PreHash learns a user's preference vector in a new way, which has two parts: the history part and the hash part. Both parts represent user preferences in a more efficient way than using a feature matrix. The history part uses an attention network to find information related to the target item in the user's history interactions and then forms a user history preference vector. It dynamically captures user preferences in different aspects. In the hash part, there are some buckets, and each of them stores the preferences of similar users. The most important function of the hash part is that for each input user, it tries to find some warm users who have interacted with similar items as the target user who might have rare interacts. In this way, the model makes better preference modeling by referring to the interaction history of warm users for target ones no matter whether they are cold not not. It helps in real applications where warm users only take a small proportion of the total.

Furthermore, PreHash stores a much smaller matrix of preference vectors rather than an entire user vector matrix. Also, by representing users as their interacted items and some similar warm users, their representations are updated online (without retraining the model), which also shows the efficiency of our model. In the experiments conducted on different large-scale datasets, by taking the place of the user embedding matrix, PreHash is verified to be helpful to multiple models in different scenarios. In summary, PreHash not only significantly reduces the number of parameters but also help improves the recommendation performance.

The main contributions of this work are listed as follows:

(1) PreHash is a flexible user preference representation module that helps previous models to deploy in real-world systems. It significantly reduces the model parameters and works well on both rich-history and rare-interaction users. Thus it is a meaningful trial to bridge the gap between research and industry applications.

(2) In the proposed PreHash, users' preferences are dynamically updated when they make new interactions, and no re-train is required as fix user embedding matrix does in previous work. It not only boosts the performance of normal users but also handles with cold-start ones.

(3) Experimental results on multiple large-scale real-world datasets show that PreHash is flexible to work with various recommendation algorithms and is able to improve the recommendation performances significantly. It helps various models apply in the real large-scale recommendation scenario.

2 RELATED WORK

2.1 User Preferences in Traditional Recommendation Algorithms

Traditional recommendation algorithms include two major approaches, i.e., Collaborative Filtering (CF) and Content-Based (CB) methods. CF methods are based on the user-item interactions, which can either use explicit user feedback [23, 32] or implicit user feedback [15, 16, 31]. Matrix Factorization (MF) is the most typical implementation of CF. Some famous recommendation algorithms such as BiasedMF [23] and SVDPP [22] are all based on MF. Preferences of users are stored in the user embedding matrix, which is not suitable for large-scale users, as well as users with rare interactions.

The main idea of CB methods is to take advantage of the content information of the users and items, such as income & occupations of users, textual descriptions of books, melodies of music, and names of items. By analyzing the interaction data, CB methods model the latent relationship between those features and user preferences [30]. For example, Basu et al. use hybrid features that combine elements of social and content-based information, making it possible to achieve more accurate recommendations [1]. However, CB methods usually perform no better than CF methods, which directly model user preferences by history interactions.

Although traditional recommendation algorithms have achieved excellent performance and been widely used for many years, they are weak in modeling non-linear high-order feature interactions.

2.2 User Representation in Deep Recommendation Models

Deep neural networks can model non-linear features and their high-order interactions. In recent years, the influence of deep learning has spread to the areas of information retrieval and recommendation. It helps to enhance the ability of traditional methods to model the non-linearity in data. For example, Convolutional Neural Networks (CNN) helps model the visual information [9, 11]. Recurrent Neural Networks (RNN) can be used to model the text data [24, 43]. They all can help form better user representations. But most of them embed user preferences into the ID embeddings. A huge user embedding matrix limits the application of these models in real-world recommender systems with millions or even billions of users. Besides, they are also suffering from the cold-start problem.

There also exist some models which take both content features as well as end-to-end ID embeddings as input features. For example, Wide&Deep [5] proposed by Google combines the deep neural network with a linear model. In the NFM [12] and DeepFM [8], both sparse IDs and content features are embedded into vectors to model feature interaction. These methods have shown remarkable performance on many datasets. However, in practical large-scale industrial systems, they cannot take user IDs as inputs for the model size and efficiency. Although better modeling the content features is helpful, collaborative filtering is important for effective recommendations to capture personalized user preferences.

2.3 Cold-Start Recommendation

The cold-start problem must be carefully considered. There are usually a large number of users with only a few historical interactions, which makes it hard for the model to capture their preferences. Many approaches utilize user content information and combine the CF and CB to handle the cold-start problems [29, 37]. Some proposed models adaptively balance the importance of different types of information for each user-item pair [34].

There are also other kinds of information that can be used to improve the cold-start recommendation, such as social relationships between users, cross-domain, and cross-media information. Based on the idea that users who have social relationships may have similar preferences, many efforts have been made to make use of social information [2, 18, 42]. Transfer learning methods try to learn users' preferences from other domains and apply them to the current domain, and some remarkable improvements have been achieved [7, 26, 28, 40]. Besides, Ma et al. [27] introduce users' content information from other platforms (user posts on Twitter) to learn extra user features for the recommendation. Their proposed model outperforms other methods in cold-start scenarios.

However, although these methods with side information help, side information is not always available. Besides, most of the mentioned models make predictions in a hybrid way, which do not consider memory and time cost in real-world systems.

3 USER PREFERENCE HASH

3.1 General Idea

The basic idea of PreHash is to learn a user's preference vector in two ways:

(1) Items that the user interacted recently.
(2) Other users that have similar preferences to this user.

Specially, PreHash uses a history part to model a user's history preferences (user history vector) from his/her interaction history. An attention network is used to retrieve the preferences related to the target item. Besides, to find users have similar preferences to this user, we design a hash part to model his/her hash preference vector (user hash vector). Finally, a Route Attention is adopted to combine the two parts (history vector and hash vector) and get the final user preference vector.

Note that in PreHash, it is unnecessary to maintain all users' feature vectors. For each warm user, the hash vector directly comes from a specific bucket vector. It is just like the lookup of an embedding matrix. But different from one-to-one user embedding matrix, in PreHash, users with fewer interactions will also store their preferences in these buckets (training the embeddings). Allocating each warm user with one fixed bucket reduces the conflicts of storing their preferences and provides more stable and dense CF. Sharing buckets of warm users with others helps recommend to users without sufficient history. Detail information will be introduced in the next sessions.

PreHash is more about a module that improves existing deep models to be able to work in the real large-scale environment. In which case is more flexible than previously proposed methods. To introduce the proposed PreHash and how it works with previous recommendation methods clearly, we take BiasedMF [23] as the base model and propose PreHash enhanced BiasedMF, which is noted as BiasedMF$_{PreHash}$. Note that BiasedMF$_{PreHash}$ is an example for introducing PreHash module, PreHash is able to work with other models too. Figure 1 is an overview of BiasedMF$_{PreHash}$.

In recommendation tasks, suppose there are a set of n users $U = \{u_i\}$ and a set of m items $V = \{v_j\}$. We use the bold fonts to represent vectors, such as $\mathbf{u}$ and $\mathbf{v}$ are the vectors of user u and item v respectively. The original prediction of BiasedMF comes from the dot product of user vector $\mathbf{u}$ and item vector $\mathbf{v}$, plus the biases:

$$p = \mathbf{u}^\top \mathbf{v} + b_u + b_v + b_g \tag{1}$$

where b_u, b_v, b_g are the user, item, and global bias respectively. BiasedMF stores a user preference matrix $\mathbb{U}$, in which each column is a user's preference vector. The proposed PreHash improves the BiasedMF by substituting the function of the user vector matrix and generating a user representation $\mathbf{u'}$ to replace $\mathbf{u}$. In this way, PreHash saves a large number of parameters and helps generate better preference vectors for each user.

In the following subsections, we will introduce the two parts in PreHash module one by one.

3.2 History Part

As shown in Figure 1, PreHash generates user history vectors according to their interaction history in History Part. For each user u, the l items he/she has interacted with are noted as $v_1^u ... v_l^u$. The history part builds a history vector of each user to capture his/her preferences from interacted items. Considering that items reveal different aspects of user preferences. For a specific target item, user history interactions have different importance. So PreHash uses an attention network (History Attention) to adaptively weight-sum the item vectors:

$$\begin{aligned} w_j^l &= \frac{exp(\mathbf{v}^\top \mathbf{v}_j^u)}{\sum_{k=1}^{l} exp(\mathbf{v}^\top \mathbf{v}_k^u)} \\ \mathbf{u}^v &= \sqrt{l} \sum_{j=1}^{l} w_j^l \mathbf{v}_j^u \end{aligned} \tag{2}$$

where $\mathbf{v}$ is the vector of the target item, w_j^l is the attention weight of the j-th interacted item in the user history. The user history vector $\mathbf{u}^v$ comes from the weighted sum of the interacted items' vectors. $\sqrt{l}$ is the history length normalization by taking the number of historical interactions into consideration. We believe that embedding the richness information of user history into their vectors is helpful, especially to consider whether he/she is a warm user.

Our PreHash has mainly two advantages of taking this kind of attention:

- Items, which a user interacted with, are consistent with user preferences to a different extent. Some items are typical representatives, but others may not. For example, buying some clothes may reveal a user's favorite color, but buying some daily necessities, like a bottle of water, contains less information.
- To predict whether a user likes a target item, it is corresponding to some specific aspects in the user preferences. The attention network is right suitable here to adaptively select the relevant items in the user history to capture the user preference to the target item. For example, if the model wants to predict whether a user likes iPhone or not, it should better consider the digital products the user bought but not books.

Note that as the user interacting with more items, new users will become rare-interaction users and finally warm users. The history part directly takes user history as input, so there is no need to retrain the model, which is a significant difference with the user preference matrix based methods. The user history vector $\mathbf{u}^v$ is

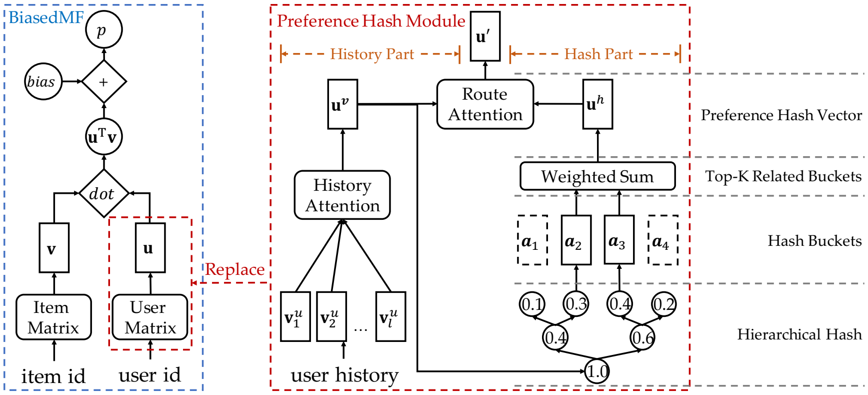

Figure 1: An example of BiasedMF$_{PreHash}$ (PreHash enhanced BiasedMF).

then fed into the hash part for further use. Besides, it is a part of the final generated user preference vector too.

3.3 Hash Part

Only with the history vector is not enough to capture the user preference, as some fresh users do not have sufficient historical feedback (their history vectors provide limited information). So we design a hash part to get a user's additional preference features based on previous warm users who interact with similar items to the target user. Warm users are users who have interacted with a lot of items, and the system knows their preferences well. The function of the hash part is to store and calculate the preferences of all users with limited spaces.

In the hash part, users with similar preferences share some buckets to store their preferences. In this way, these warm users are referenced when there come other users. It will be very helpful when there are rare-interaction users. Besides, as the user interacting with more items, his/her hash vectors will also dynamically change. It is different from the user embedding matrix and significantly reduces the requirements and costs for retraining the model.

The hash part mainly consists of three components, namely hash buckets, hierarchical hash, and top-k weighted. The workflow is shown in Figure 1, and we will introduce them one by one.

3.3.1 Hash Buckets. Suppose we have $h (h \ll n)$ buckets $A = \{a_i\}$, and their vectors are $\mathbf{a}_1 ... \mathbf{a}_h$ ($h = 4$ in Figure 1). The buckets are used to store the preferences of some similar users. Firstly, we bind each bucket a_i with a warm user u_{a_i}. We call these selected warm users "Anchor User" and their final hash vectors directly comes from the corresponding hash buckets a_i, i.e.:

$$\mathbf{u}_j^h = \begin{cases} \mathbf{a}_i, & u_j \text{ is an anchor user} \\ hash(\mathbf{u}_j^v), & \text{otherwise} \end{cases} \tag{3}$$

where $hash(\cdot)$ is the hash part which can be regarded as a function that takes the user history vector as input and output the user hash vector. We bind each bucket an anchor user for mainly the following reasons:

• These buckets provide references for users to enrich their preference representations. We need sufficient information (preferences of warm users) stored in each bucket.

• Warm users have plenty of history interactions, and their preferences usually have more clear difference with each other than among other users. Using one bucket to store multiple warm users affects the accuracy of modeling preferences of these users.

• In the user matrix of BiasedMF and many other similar models, there are a large number of vectors of rare-interaction users contains little information, which wastes a lot of resources. Obtaining the vector of anchor users is similar to the user embedding matrix that maps a user to a vector, but PreHash only builds that matrix for anchor users. Besides, we know that CF works better in dense data than sparse data. Conducting CF among anchor users are more efficient and effective. In this way, PreHash significantly reduces the model parameters but remains the CF process between the anchor users.

• Other users with fewer history interactions are dynamically hashed to different buckets as they interact with more and more items, and we should not fix their hash vectors. But these warm users have more stable history preferences, and it will not bring too much deviation if we fix their preference vectors.

These warm users can be predefined in various ways, whether manually selecting or by some clustering methods. For simplicity, we choose the users with the most interactions in our implementation here.

For other users, PreHash uses a hierarchical hash to find the top related buckets, and weight-sums these bucket vectors to form the preference hash vector. Vectors of buckets are also trained by interactions of these users, so their preferences are distributed in the corresponding buckets. The difference to the anchor users is that each of them does not have a specific fixed bucket to store their preferences but shares with others. And their buckets may dynamically change as they interacted with more items.

3.3.2 Hierarchical Hash. To better calculate a specific user's preference vector, we design a hierarchical hash structure in PreHash. The structure will be able to find useful buckets that store the preferences of users who have interacted with similar items.

With the history vector $\mathbf{u}^v$ of a user as the input, hierarchical hash works like a fully-connected tree where the root node allocates its weight to children nodes and finally leaves (various buckets). Let n_j^i denotes the j-th node in layer i. Each node n_j^i has a decision vector $\mathbf{n}_j^i$, and it inherits a relevance weight r_j^i from its parent node. Suppose that node n_j^i has c children nodes $(n_k^{i+1} ... n_{k+c-1}^{i+1})$, then it allocates the weight r_j^i to them by:

$$r_{k+x}^{i+1} = r_j^i \frac{exp(\mathbf{u}^{v\top}\mathbf{n}_{k+x}^{j+1})}{\sum_{y=0}^{c-1} exp(\mathbf{u}^{v\top}\mathbf{n}_{k+y}^{j+1})}, \quad x = 0, ..., c-1 \tag{4}$$

where r_{k+x}^{i+1} is the relevance weight of the x-th child. Note that relevance weights of all nodes on the same level have a total of 1, e.g., $\sum_j r_j^i = 1$, and initially, on the root $r_1^1 = 1$. The number of leaves equals to the number of buckets, and finally, the relevance weights of leaves are the weights of buckets corresponding to the current user history. The decision nodes here are somehow similar to the keys in memory networks. The functions of them are both to get the relevance weights of all the buckets (memory slots). Note that the designed hash structure is different from existing memory networks in various aspects. The detailed comparison is shown in Section 3.3.4.

As there are often thousands and even more representative anchor users in large-scale data, it is a little hard to differentiate these buckets in one step. So we propose a hierarchical hash structure to spread the load and achieve the goal step by step. It automatically clusters the users at different levels. We use three layers in our experiments. More or fewer layers are also tried, but this setting provides a more stable result.

3.3.3 Top-K Weighted Sum. To form the user preference hash vector, instead of the weighted sum of all the buckets, PreHash only considers the most related K buckets. The reason is that it takes much time and space to weight-sum such a large number of bucket vectors. It is also unnecessary because the weights usually have long-tail small values (which contribute little even considering them but significantly increase the cost of back-propagation). Let r_j from the last layer in the hierarchical hash denote the relevance weight of bucket a_j. When making predictions, the intuitive idea is to calculate the preference hash vector of user u as follows:

$$\mathbf{u}^h = \frac{1}{\sum_{j=1}^{K} r_{t_j}} \sum_{j=1}^{K} r_{t_j} \mathbf{a}_{t_j} \tag{5}$$

where $t_1 \ldots t_K$ are the indexes of the largest K relevance weights and thus $\mathbf{a}_{t_0} \ldots \mathbf{a}_{t_K}$ are the most relevant K bucket vectors. K is usually much smaller than the number of hash buckets, i.e., $K \ll h$.

However, if we keep always considering the top-K relevant buckets during the training process, we observe that the module tends to visit only a small number of buckets and performs badly. We think the reason is that some well-trained bucket vectors get higher weights than others. If the module always focuses on the vectors with the largest weights during training, they will be trained to store the preferences of too many users. Thus, many other buckets are never visited, and their corresponding decision nodes are rarely trained. These buckets have no chance to be updated and keep irrelevant. To tackle the problem, PreHash chooses K random buckets when training each sample:

$$\mathbf{u}^h = \frac{1}{\sum_{j=1}^{K} r_{s_j}} \sum_{j=1}^{K} r_{s_j} \mathbf{a}_{s_j} \tag{6}$$

where $t_1 \ldots t_K$ are K randomly selected indexes. In this way, for each training sample, PreHash explores some random buckets for the target user. The module is then trained to increase the weights of relevant buckets and decrease those irrelevant. The procedure can be regarded as an exploration of trying different buckets. Each sample visits different buckets in different epochs. The hash part takes the user history vectors as input, so better hashing a user also improves the users with similar preferences. Finally, the module is expected to calculate the relevance for all the buckets properly.

3.3.4 Comparison with Memory Networks. The hash part has similar functions as some memory networks [4, 17, 35]. Both of them find some relevant buckets/slots and weight-sum the corresponding vectors to generate a new vector. However, there are five major differences between the hash part and a memory network:

(1) Regular key-value memory networks in recommendation usually have no more than one hundred of slots [4, 17, 36]. The number of buckets in the hash part is more than thousands, which is much larger. (2) Each anchor user is bound to a specific bucket and skips the hierarchical hash procedure. Memory networks have no such direct visit (e.g., anchor users will get their hash vectors directly). (3) Due to a large number of hash buckets, the hash process has a hierarchical design. Memory networks usually use one layer of memory keys, which directly map to slots. (4) The hash part weight-sums some of the buckets for the efficiency. Otherwise, it will take much more time to train and run the module. Memory networks usually take all the memory slots. (5) During the training, the hash part selects the buckets randomly rather than according to the hash weights. This way provides an exploration of different hash buckets. Otherwise, the performance of the hash cannot be guaranteed. Memory networks always refer to the relevance weight.

From the comparison, it can be concluded that a regular memory network cannot achieve the same functions as the hash part. Each bucket of the hash part refers to a cluster. The hash part is more suitable for storing a large number of clusters and estimating which buckets the input may belong to. Memory networks are usually used to store and retrieve information. Information in memory networks are heterogeneous, and slots have different latent meanings.

3.4 Module Output

To generate the final user preference vector, PreHash uses an attention network (named as Route Attention) to combine the vector of $\mathbf{u}^v$ from History Part and $\mathbf{u}^h$ from Hash Part. The user history vector $\mathbf{u}^v$ records the preferences from the items the user has interacted with. The user hash vector $\mathbf{u}^h$ provides preferences of other users who have interacted with similar items. For a user with plenty of historical feedbacks, $\mathbf{u}^v$ may be enough to capture his/her preferences. But for new users, it is better to utilize the $\mathbf{u}^h$ so as to explicitly reference the history of other users. It is reasonable and essential to use the Route Attention to adjust sources of the user preference vector dynamically.

We combine the two vectors to form the final user representation $\mathbf{u}'$ in the following steps.

$$\begin{aligned}
e_u^v &= \mathbf{h}^T f(\mathbf{W}\mathbf{u}^v + \mathbf{b}), \quad e_u^h = \mathbf{h}^T f(\mathbf{W}\mathbf{u}^h + \mathbf{b}) \\
w_u^v &= \frac{exp(e_u^v)}{exp(e_u^v) + exp(e_u^h)} = 1 - w_u^h \\
\mathbf{u}' &= w_u^v \mathbf{u}^v + w_u^h \mathbf{u}^h
\end{aligned} \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{t \times d}, \mathbf{b} \in \mathbb{R}^t, \mathbf{h} \in \mathbb{R}^t$ are the parameters of attention network, and t denotes the hidden layer size of the attention network and d is the user vector size. f is the non-linear activation function and we use *relu* in our implementation.

The user preference $\mathbf{u}'$ is the final output of PreHash.

3.5 Cooperating with Different Models

PreHash works for models that have a user embedding matrix to store user vectors. Taking BiasedMF as an example, we replace the user feature matrix $\mathbb{U}$ in BiasedMF with the proposed PreHash module, as shown in Figure 1. When the preference vector of user u is needed, instead of searching $\mathbf{u}$ from user matrix, PreHash will calculate $\mathbf{u}'$ for him/her. The loss function and training strategy of BiasedMF$_{PreHash}$ are the same as BiasedMF.

PreHash also works similarly for other models. For example, in Wide&Deep [5], if it takes the user ID as one of the features, there must be an embedding layer which contains the corresponding weight matrix. PreHash replaces the user matrix or embedding layer, which has the same function as taking user IDs as inputs and outputting the user preference vectors but fewer parameters. The module parameters include the bucket vectors $\{\mathbf{a}_i\}$, hash decision node vectors $\{\mathbf{n}_j^i\}$, and the Route Attention parameters. They all can be randomly initialized and trained together with these models with no changes to their loss functions. In the next sections, we will verify if the proposed PreHash is able to boost the performance of combined recommendation algorithms.

4 EXPERIMENTAL SETTINGS

4.1 Datasets

We conduct experiments on four public datasets from Amazon in different categories [10] and the public dataset in RecSys Challenge 2017. Some detailed information of the datasets is shown in Table 1.

Table 1: Statistics of evaluation datasets.

Dataset	#Interaction	#User	#Item
Grocery & Gourmet Food	1,297,156	768,438	166,049
Pet Supplies	1,235,316	740,985	103,288
Video Games	1,324,753	826,767	50,210
Books	22,507,156	8,026,324	2,330,066
RecSys Challenge 2017	6,470,857	1,497,020	1,306,054

- **Amazon Dataset** is a public e-commerce dataset. The dataset contains reviews and ratings of items given by users on Amazon, a popular e-commerce website. We use four sub-datasets of different categories in the whole dataset: Grocery & Gourmet Food, Pet Supplies, Video Games, and Books. Note that the Books dataset is much large than others, which is used to verify the efficiency of PreHash in our experiments.
- **RecSys Challenge 2017** (RSC2017) focuses on the job recommendation task. The provided dataset contains plenty of both users and items features, which is an excellent evaluation scenario for some features based models like Wide&Deep and ACCM.

4.2 Baselines

In our experiments, PreHash is cooperated and compared with the following models:

- **BiasedMF** [23]. It is a famous matrix factorization model.
- **NeuMF** [14]. It conducts CF with a neural network. It is one of the most famous neural collaborative filtering models.
- **Wide&Deep** [5]. It combines the deep neural network and linear models and takes both content features and ID embeddings as inputs. Wide&Deep is verified as one of the best deep neural recommendation models in many different scenarios.
- **ACCM** [34]. It is a state-of-the-art model works on both warm and cold scenarios by combining the CF and CB.

Besides, we also compared the BiasedMF$_{PreHash}$ and NeuMF$_{PreHash}$ with some CF models which also take user history as inputs:

- **FISM** [20]. It is a state-of-the-art item-based CF model in which the user preference vector comes from the sum of historical item vectors instead of a user embedding matrix.
- **SVD++** [22]. It is a well-known CF method that integrates both explicit and implicit feedback.

4.3 Training and Evaluation

Each dataset is split into training, validation, and test sets. We ensure that all users in the validation and test sets have at least one interaction in the training set. We leave the last two positive interactions of at most 100,000 users into validation and test sets, respectively. It is called the leave-one-out evaluation in recommendation, which is widely used in literature [3, 13, 20].

Our experiments are conducted on top-n recommendation tasks. We use the pair-wise training strategy [31] to train all the models, including baselines. Pair-wise training is a commonly used training strategy in many ranking tasks which usually performs better than point-wise training. For each positive interaction v^+, we randomly sample an item the user dislikes or has never interacted with before as the negative sample v^- in each epoch. Then, the loss function of the models is:

$$L = -\sum_{v^+} \log\left(sigmoid(p(v^+) - p(v^-))\right) + \lambda_\Theta \|\Theta\|_F^2 \qquad (8)$$

where $p(v^+)$ and $p(v^-)$ are the prediction results of v^+ and v^-, respectively, and $\lambda_\Theta \|\Theta\|_F^2$ is the ℓ_2-regularization. The loss function encourages the predictions of positive interactions to be higher than the negative samples.

In evaluation, we leave the last positive interaction of at most 100,000 users, and sample 100 v^- for each v^+. The models are evaluated according to the rank of v^+ in these 101 candidates. This method is widely adopted by previous works [14, 38, 41].

We adopt NDCG@10 [19] to evaluate the top-10 lists of models and NDCG@1 to assess whether the positive item is ranked first. Note that NDCG@1 actually equals to Precision@1 (P@1 for short).

4.4 Implementation Details

All the models, including baselines, are trained with Adam [21] in mini-batches at the size of 128. The learning rate is 0.001, and early-stopping is conducted according to the performance on the validation set. Models are trained at most 200 epochs. To prevent models from overfitting, we use both ℓ_2-regularization and dropout. The weight of ℓ_2-regularization λ_Θ is set between 1×10^{-7} to 1×10^{-5} and dropout ratio is set to 0.2. Vector sizes of all the user, item, and feature vectors are 64. The number of buckets (anchor users) h is 1024, and we select $K = 128$ of them to form the hash preference vectors without particular illustrations. For the hierarchical hash, we use a 3-layer structure with [1-64-1024]. We run the experiments with 5 different random seeds and report the average results. All models are trained with a GPU (NVIDIA GeForce GTX 1080Ti) with 11GB GPU memory. Codes are provided at https://github.com/THUIR/PreHash.

https://www.amazon.com/
http://jmcauley.ucsd.edu/data/amazon/index.html
http://www.recsyschallenge.com/2017/

Table 2: Performance on Amazon datasets.

	Grocery & Gourmet Food		Pet Supplies		Video Games		Books	
	NDCG@10	P@1	NDCG@10	P@1	NDCG@10	P@1	NDCG@10	P@1
BiasedMF [23]	0.4014	0.2386	0.4667	0.2733	0.4919	0.2849	Out Of Memory[1]	
NeuMF [14]	0.3788	0.2174	0.4334	0.2442	0.4410	0.2401	Out Of Memory	
SVD++ [22]	0.3990	0.2372	0.4717	0.3030	0.4857	0.3106	Out Of Memory	
FISM [20]	**0.4143**	0.2553	0.4911	0.3047	0.4929	0.2921	0.4669	0.3198
NeuMF$_{PreHash}$	0.3772	0.2373	0.4829	0.3062	0.5391	0.3448	0.4687	0.3259
BiasedMF$_{PreHash}$	0.4127	**0.2647***	**0.5102***	**0.3310***	**0.5720***	**0.3772***	**0.5265***	**0.3842***

1. Out of GPU memory during the training process, same for other tables.

*. Significantly better than the other models with $p < 0.05$, same for other tables.

5 EXPERIMENTAL RESULTS AND ANALYSIS

We conduct some experiments on various datasets to answer the following research questions:

RQ1 Does the proposed PreHash improve the recommendation effectiveness?
RQ2 Does PreHash improve the model efficiency?
RQ3 Does PreHash work for different models?
RQ4 Does PreHash provide better recommendations for rare-interaction users?

5.1 Effectiveness of PreHash

To answer RQ1 and verify PreHash improves the performance of recommendation models, we combine the PreHash with BiasedMF and NeuMF.

The overall performance of BiasedMF$_{PreHash}$ and NeuMF$_{PreHash}$ in Amazon datasets, compared with other baselines, is shown in Table 2. Due to the lack of content features in Amazon datasets, it is unsuitable for applying feature-based deep models (Wide&Deep, ACCM) here. So the comparison between them will be conducted on the RSC2017 dataset. As the space is limited, the performances of all models on RSC2017 dataset will be reported in Section 5.3.

First, as for the baseline models, it is clear that the original NeuMF and BiasedMF perform worst among the models. The reason is that CF methods usually are effective with sufficient feedbacks. However, the datasets in our experiments all have a sparsity larger than 99.99%, so a naive MF method cannot handle such sparse datasets. SVD++, which takes all the user historical feedbacks into consideration, performs slightly better than BiasedMF. It indicates that explicitly modeling the user history is helpful. FISM outperforms other baselines significantly. It has no such a user preference matrix as BiasedMF & SVD++ and makes recommendations based on item similarities. The good performance of FISM verifies that MF based methods perform badly on large-scale sparse data, and it is essential to consider representing user preferences in other ways. Besides, FISM is based on item similarity, whose results show that representing users as their interacted items sometimes performs better than using a user embedding matrix. It is also verified by recent findings that ItemKNN works better than many state-of-the-art neural models [6].

Then, looking into the performance of our model, we can see that if we replace the user preference vectors in BiasedMF with the PreHash, the model achieves the best results on all the datasets except the NDCG@10 on Grocery & Gourmet Food (comparable, not significantly worse). And in all scenarios, BiasedMF$_{PreHash}$ (NeuMF$_{PreHash}$) makes great improvements to the original BiasedMF (NeuMF) and other baselines, showing the effectiveness of PreHash. The reason is that for all users, the history part of PreHash dynamically retrieves the preferences related to the target item in the user history. And for each target user, the hash part helps find some related buckets storing the preferences of users interacted with similar items. Their history helps recommend to the target user, especially for those rare-interaction users. In this way, PreHash improves the performance of both rich-history and rare-interaction users. PreHash makes greater improvements on Video Games because the dataset has much fewer items than the others. With sufficient interactions, all items representations are learned well and help better form the user preference vector in PreHash. Note the origin BiasedMF and NeuMF cannot be trained on our devices because of the limitation of GPU memory. But BiasedMF$_{PreHash}$ and NeuMF$_{PreHash}$ significantly reduces the number of model parameters, especially on the datasets in which users are much more than items.

5.2 Efficiency of PreHash

To better verify the efficiency of PreHash (RQ2), we show the number of model parameters and training time per epoch on Amazon datasets in Table 3. The following observations can be made. Firstly, PreHash reduces the model parameters of BiasedMF and NeuMF at a large margin. On some datasets with many more users than items, it saves more than 90% of parameters because there is no more a huge user preference matrix. It will directly affect the cost of training, deploying, and running of recommender algorithms in real-world systems. Secondly, although PreHash trades some time for space, results show that models with PreHash take only slightly more time than the original ones. (The statistics may have small fluctuations due to the busyness of the running devices.) PreHash is slower than indexing a vector from a user matrix, but the significant reduction of parameters and improvements of performance deserve the small cost. Thirdly, BiasedMF, NeuMF, and SVD++ cannot be trained on the *Books* dataset because the GPU memory size exceeded. Although they can be trained on CPUs, the time cost will be unacceptable. It means that we need many more resources to run these models on large datasets. In real-world recommender systems, the issue is more prominent. FISM, BiasedMF$_{PreHash}$ and NeuMF$_{PreHash}$ finish the training because they have no such a huge user vector matrix. BiasedMF$_{PreHash}$ and NeuMF$_{PreHash}$ even have fewer parameters than the FISM (saved lots of memory resource) and take comparable time to finish an epoch, which verifies the efficiency of PreHash.

Table 3: Number of parameters and training time per epoch on Amazon datasets.

	Grocery & Gourmet Food		Pet Supplies		Video Games		Books	
	#Para	Time/epoch	#Para	Time/epoch	#Para	Time/epoch	#Para	Time/epoch
SVD++ [22]	71.4M	0.10h	61.5M	0.08h	60.2M	0.09h	822M	Out Of Memory
FISM [20]	22.2M	0.10h	14.1M	0.08h	7.3M	0.08h	309M	2.74h
NeuMF [14]	119.6M	0.14h	108.1M	0.13h	112.3M	0.11h	1326M	Out Of Memory
NeuMF$_{PreHash}$	21.4M	0.14h	13.3M	0.12h	6.5M	0.12h	298M	2.69h
BiasedMF [23]	60.7M	0.10h	54.9M	0.07h	57.0M	0.07h	673M	Out of Memory
BiasedMF$_{PreHash}$	**10.9M**	0.11h	**6.8M**	0.11h	**3.4M**	0.10h	**152M**	2.83h

Table 4: PreHash enhanced models on RSC2017.

Model	#Para	NDCG@10	P@1
SVD++	266M	0.7696	0.6190
FISM	170M	0.7819	0.6477
NeuMF	359M	0.7756	0.6371
NeuMF$_{PreHash}$	167M	**0.7851***	**0.6550***
BiasedMF	182M	0.7834	0.6465
BiasedMF$_{PreHash}$	85M	**0.7978***	**0.6680***
Wide&Deep	182M	0.8285	0.7144
Wide&Deep$_{PreHash}$	85M	**0.8396***	**0.7278***
ACCM	182M	0.8297	0.7215
ACCM$_{PreHash}$	85M	**0.8351***	**0.7267***

5.3 Flexibility of PreHash

As introduced in Section 3.5, PreHash is not designed for a specific model. We have combined PreHash with some MF based models (BiasedMF, NeuMF) in previous sections. Now we are going to verify if deep neural algorithms (Wide&Deep, ACCM) with various content features and sparse user ID features as inputs can consider being enhanced by PreHash (RQ3). The Amazon datasets contain few content features of users or items, which are not suitable for feature-based deep models. We use RSC2017 dataset to evaluate these models, because the dataset provides plenty of both user and item content features for the model to capture user preferences and item characteristics. Previous MF based models and baselines are also reported here.

Results in Table 4 show that on RSC2017, all of the models are enhanced by PreHash and achieve significant improvements, including the two feature-based deep models. It verifies that PreHash is flexible to be applied to various models, no matter traditional models or deep neural models, no matter MF models or feature-based models. Wide&Deep and ACCM perform significantly better than other models because they utilize more content features.

Note that BiasedMF, Wide&Deep, and ACCM have a similar number of parameters because the user and item embedding matrix dominates the parameters. Although hundreds of content features and fully-connected deep layers bring tens of thousands of parameters, they are negligible comparing to the huge user and item embedding matrix. The remaining parameters in the PreHash enhanced models are most item embeddings. These embeddings are replaceable by item vectors learned from plenty of item features in real-world recommender systems.

5.4 Performance of Rare-Interaction Users

To study whether PreHash improves the recommendation performance for rare-interaction users (RQ4), we use the RSC2017 to evaluate all of the models on different groups of users (RSC2017 is applied to include feature-based deep models for comparison). The users in the test set of RSC2017 are divided into three groups according to how many items they have interacted with in the training set. Users who have less than 3 interacted items are regarded as rare-interaction users ($l \leq 3$), and rich-history users are those who have 10 or more historical interactions ($l \geq 10$). Other users are normal users ($3 < l < 10$). We evaluate models with or without PreHash separately on these three groups of users.

Table 5: Performance (NDCG@10) of users with different number of history interactions l on RSC2017.

Model	$l \leq 3$	$3 < l < 10$	$l \geq 10$
NeuMF	0.7215	0.7919	0.8029
NeuMF$_{PreHash}$	**0.7336***	**0.8071***	**0.8254***
BiasedMF	0.7325	0.8013	0.8109
BiasedMF$_{PreHash}$	**0.7428***	**0.8149***	**0.8285***
Wide&Deep	0.7793	0.8405	0.8581
Wide&Deep$_{PreHash}$	**0.7871***	**0.8563***	**0.8787***
ACCM	0.7828	0.8429	0.8563
ACCM$_{PreHash}$	**0.7883***	**0.8556***	**0.8615***

Experimental results are shown in Table 5. It is clear that all models perform better on rich-history users than rare-interaction users. It is reasonable because warm users have more history interactions that help models capture their preferences. Among the baseline models without PreHash, NeuMF and BiasedMF perform the worst, especially on rare-interaction users, because they do not utilize the content features as the other two deep models. The four models achieve significant improvements on all three groups of users after enhanced by PreHash. It is not easy to make such improvements for rare-interaction users without utilizing external information. PreHash is powerful, for the hash part explicitly takes the preferences of warm users who have interacted with similar items of target users to help make recommendations. Together with the history part, it also provides more accurate preference modeling for warm users by filtering the noises in history and modeling item relevance.

6 FURTHER ANALYSIS

6.1 Ablation Study

To better understand the influence of different parts in PreHash, we conduct an ablation study on Video Games dataset. Results are shown in Table 6.

If we weight-sum the historical item vectors with the History Attention, and replace the user preference vector in BiasedMF with the

Table 6: Results of ablation study on Video Games.

Model	NDCG@10	P@1
BiasedMF	0.4919	0.2849
BiasedMF$_{History\ Part}$	0.5613	0.3668
BiasedMF$_{Jaccard}$	0.5609	0.3679
BiasedMF$_{K\text{-}means}$	0.5595	0.3629
BiasedMF$_{PreHash}$	**0.5720***	**0.3772***

output history vector $\mathbf{u}^v$ of History Part in PreHash (without Hash Part), i.e., BiasedMF$_{History\ Part}$, it makes significant improvements. The reason is that the latent vectors in the MF-based CF model can hardly capture the user history and preferences when the data is super sparse. Besides, different items contain distinct amounts of information about user preferences, and they are related to the target item in various degrees. The History Attention helps capture these differences among the items adaptively. To compare PreHash with some clustering/matching methods, we adopt K-means on the user history to form h clusters, and use the cluster-ID to represent the users (BiasedMF$_{K\text{-}means}$). We also try that after select h warm users as anchors, for each non-anchor user, we calculate the Jaccard similarities between he/she and every anchor user based on their interacted items. And we assign the non-anchor user to the anchor user (bucket) with the largest Jaccard similarity (BiasedMF$_{Jaccard}$). Either K-means and Jaccard can replace the hash part, but they both perform worse than PreHash, which shows the effectiveness of PreHash to model similar user preferences. Finally, with a Hash Part, the BiasedMF enhanced by the entire PreHash (BiasedMF$_{PreHash}$) achieves the best results. The Hash Part improves the ability of the model to handle users without sufficient history, especially those rare-interaction users, which is one of the toughest problems in real-world recommender systems.

6.2 Parameter Study

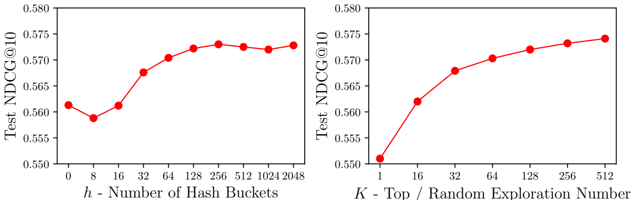

Figure 2: Performance of BiasedMF$_{PreHash}$ on Video Games with different amounts of hash buckets.

We also study the performances when BiasedMF$_{PreHash}$ with different numbers of hash buckets and different top numbers in the hashing process (or random exploration number in the training process). The results on Video Games dataset are shown in Figure 2 (K is no larger than 128 when changing h, and h is fixed to 1024 when changing K). $h = 0$ means there is no hash part, and the history vector $\mathbf{u}^v$ replaces the user vectors in BiasedMF. As shown in the figure, a small number of hash buckets even work worse than no hash buckets. We think the reason is that storing too many users' preferences in a bucket vector makes the vectors overload and probably chaotic. As the number of hash buckets increasing, the model performance keeps improving. It is because these buckets store different categories of users, and thus, more buckets differentiate more categories of users. These buckets help model the preferences of different users without enough historical information. The performance stops growing when the number of hash buckets is larger than thousands. The reason is that too many buckets increase the difficulty of the hash process, and a more powerful hash strategy is needed in such cases. Similarly, by exploring and considering more hash buckets (larger K), the performance grows because of more accurate preference modeling and stops increasing when the K is large enough. We finally use $h = 1024$ and $K = 128$ for both effectiveness and efficiency.

6.3 Case Study

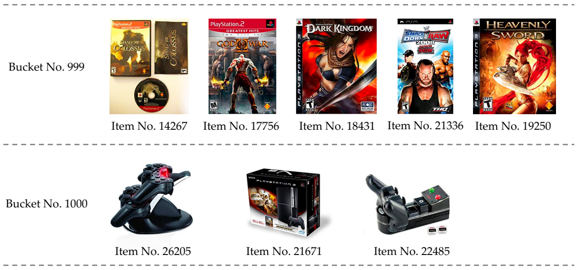

Figure 3: Case study on Video Games.

To find out what PreHash has learned in the buckets, we take some buckets (No.999 and No.1000) as examples. We collect the history of users in each bucket and rank items according to how many users in the bucket bought the item. Figure 3 are the most popular items in the two buckets (in the provided metadata, we can only find information of three items among the top five). It is clear that after the hierarchical hash, the two buckets are related but different from each other. Popular items in the bucket No.999 are CDs of games on Sony Play Station 2/3, and most of them are about fighting. Items in the bucket No.1000 are hardware devices such as Sony Play Station 3 and its matching handles and chargers. The results show the effectiveness of PreHash, which stores different kinds of preferences in different buckets.

7 CONCLUSIONS

The main idea of the paper comes up when we try to deploy some deep CF models on an industry recommender system. Most of these models build a user embedding matrix that needs unacceptable computing resources. So we attempt to represent all the users with limited embeddings (hash buckets). In this work, a novel Preference Hash (PreHash) module is proposed to model large-scale users including rare-interaction ones in the recommendation. Previous studies rarely tackle efficiency and cold-start problems together. Besides, PreHash can cooperate with various recommendation models by replacing the user vector matrix. Our experiments show that PreHash not only improves the performance of different models but also reduces the parameters. It is an efficient and effective module that helps apply previously proposed models to large-scale real-world recommender systems.

In the future, we consider improving PreHash by designing better anchor user selection methods. It is important to consider the typicality of anchor users and how to place them in different buckets. Besides, how to improve the item representation together with users is also an interesting problem.

http://snap.stanford.edu/data/amazon/productGraph/metadata.json.gz

REFERENCES

[1] Chumki Basu, Haym Hirsh, and William Cohen. 1998. Recommendation as classification: using social and content-based information in recommendation. In *Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*. American Association for Artificial Intelligence, 714–720.

[2] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. 2019. An Efficient Adaptive Transfer Neural Network for Social-aware Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 225–234.

[3] 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 the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 335–344.

[4] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. ACM, 108–116.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 7–10.

[6] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 101–109.

[7] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 278–288.

[8] 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 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 1725–1731.

[9] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*. ACM, 507–517.

[10] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 507–517.

[11] Ruining He and Julian McAuley. 2016. VBPR: visual Bayesian Personalized Ranking from implicit feedback. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI Press, 144–150.

[12] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 355–364.

[13] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2354–2366.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. ACM, 173–182.

[15] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 549–558.

[16] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. IEEE Computer Society, 263–272.

[17] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 505–514.

[18] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the 4th ACM Conference on Recommender systems*. ACM, 135–142.

[19] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[20] Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 659–667.

[21] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–434.

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

[24] Piji Li, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam. 2017. Neural rating regression with abstractive tips generation for recommendation. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 345–354.

[25] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*. ACM, 811–820.

[26] Zhongqi Lu, Erheng Zhong, Lili Zhao, Evan Wei Xiang, Weike Pan, and Qiang Yang. 2013. Selective transfer learning for cross domain recommendation. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 641–649.

[27] Weizhi Ma, Min Zhang, Chenyang Wang, Cheng Luo, Yiqun Liu, and Shaoping Ma. 2018. Your Tweets Reveal What You Like: Introducing Cross-media Content Information into Multi-domain Recommendation.. In *IJCAI*. 3484–3490.

[28] Seungwhan Moon and Jaime G Carbonell. 2017. Completely Heterogeneous Transfer Learning with Attention-What And What Not To Transfer.. In *IJCAI*. 2508–2514.

[29] Seung-Taek Park and Wei Chu. 2009. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*. ACM, 21–28.

[30] Michael J Pazzani and Daniel Billsus. 2007. Content-based recommendation systems. In *The adaptive web*. Springer, 325–341.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

[32] Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine learning*. ACM, 880–887.

[33] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[34] Shaoyun Shi, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Attention-based Adaptive Model to Unify Warm and Cold Starts Recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 127–136.

[35] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*. 2440–2448.

[36] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 729–739.

[37] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. DropoutNet: addressing cold start in recommender systems. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 4964–4973.

[38] Chenyang Wang, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. 2019. Modeling Item-Specific Temporal Dynamics of Repeat Consumption for Recommender Systems. In *The World Wide Web Conference*. ACM, 1977–1987.

[39] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1235–1244.

[40] Lin Xiao, Zhang Min, Zhang Yongfeng, Liu Yiqun, and Ma Shaoping. 2017. Learning and transferring social and item visibilities for personalized recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 337–346.

[41] Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. 2019. Relational Collaborative Filtering: Modeling Multiple Item Relations for Recommendation. *arXiv preprint arXiv:1904.12796* (2019).

[42] Tong Zhao, Julian McAuley, and Irwin King. 2014. Leveraging social connections to improve personalized ranking for collaborative filtering. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*. ACM, 261–270.

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