Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks

Jin Huang
School of Information, Renmin University of China
jin.huang@ruc.edu.cn

Wayne Xin Zhao*
School of Information, Renmin University of China
batmanfly@gmail.com

Hongjian Dou
School of Information, Renmin University of China
hongjiandou@ruc.edu.cn

Ji-Rong Wen
Beijing Key Laboratory of Big Data Management and Analysis Methods
jrwen@ruc.edu.cn

Edward Y. Chang
Research & Innovation, HTC
eyuchang@gmail.com

ABSTRACT

With the revival of neural networks, many studies try to adapt powerful sequential neural models, *i.e.,* Recurrent Neural Networks (RNN), to sequential recommendation. RNN-based networks encode historical interaction records into a hidden state vector. Although the state vector is able to encode sequential dependency, it still has limited representation power in capturing complicated user preference. It is difficult to capture fine-grained user preference from the interaction sequence. Furthermore, the latent vector representation is usually hard to understand and explain.

To address these issues, in this paper, we propose a novel knowledge enhanced sequential recommender. Our model integrates the RNN-based networks with Key-Value Memory Network (KV-MN). We further incorporate knowledge base (KB) information to enhance the semantic representation of KV-MN. RNN-based models are good at capturing sequential user preference, while knowledge-enhanced KV-MNs are good at capturing attribute-level user preference. By using a hybrid of RNNs and KV-MNs, it is expected to be endowed with both benefits from these two components. The sequential preference representation together with the attribute-level preference representation are combined as the final representation of user preference. With the incorporation of KB information, our model is also highly interpretable. To our knowledge, it is the first time that sequential recommender is integrated with external memories by leveraging large-scale KB information.

KEYWORDS

Sequential recommendation, memory network, knowledge base

*Corresponding author

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210017

ACM Reference Format:
Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. 2018. Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3210017

1 INTRODUCTION

With the rapid development of Web techniques, recommender systems (RS) play a more and more important role in matching user needs with rich resources (called *items*) from various online platforms. For building an effective recommender system, a key factor is able to accurately characterize and understand users' interests and tastes, which are intrinsically dynamic and evolving. To achieve this goal, the task of sequential recommendation has been proposed to better satisfy sequential user needs [26], which aims to predict the successive item(s) that a user is likely to interact with given her past interaction records.

Traditional recommendation methods (*e.g.,* standard MF [17]) can't well solve the sequential recommendation task, since they usually model static user-item interactions. For capturing sequential patterns, the classic FPMC model [26] has been proposed to factorize user-specific transition matrix by considering the Markov Chain. A major problem of FPMC is that it still adopts the static representation for user preference. With the revival of neural networks, many studies try to adapt powerful sequential neural models, *i.e.,* Recurrent Neural Networks (RNN), to sequential recommendation [35], including session-based RNN [15], user-based RNN [5] and attention-based RNN [18]. RNN-based models have been shown effective to improve the performance of sequential recommendation [15]. By encoding historical interaction records into a hidden state vector (called *sequential preference representation*), it is possible for these methods to capture dynamic user preference over time and measure the likelihood of the next item. Although the state vector is able to encode sequential dependency, it has limited representation power in capturing complicated user preference. Since the state vector is encoded in a highly abstractive way, it is difficult to capture or recover fine-grained (*e.g.,* attribute or feature level) user preference from the interaction sequence. Furthermore, the latent vector representation is usually hard to understand and

explain. In recommender systems, interpretability is a very important factor to consider [14, 30]. These issues make it challenging to develop an effective and interpretable sequential recommender.

To enhance the capacity of modeling fine-grained user preference in an interpretable way, our idea is to incorporate external knowledge into the sequential recommender. The incorporated knowledge should be rich and flexible to characterize varying context information in different domains. A key problem is what kind of knowledge we can use and how we represent it. In this paper, we propose to link items in recommender systems with existing knowledge base (KB) entities, and leverage structured entity information for improving sequential recommendation. KBs store knowledge in triples of the form (HEAD ENTITY, RELATION, TAIL ENTITY), typically corresponding to attribute information of entities. KBs provide a general way to flexibly characterize context information of entities from various domains. To obtain a compact representation for KB information, we adopt the KB embedding approach (*i.e.,* TRANSE [1]) to mapping entities and relations into low-dimensional vectors, called *KB embeddings*.

The major difficulty in designing the knowledge-enhanced sequential recommender is ~~RNN-based models usually have limited short-term memories~~ [3], which are not suitable to store external knowledge (*e.g.,* KB information) for long-term usage. Inspired by recent progress on improving the memory mechanism of neural networks [19, 21, 34], we propose to augment the RNN-based sequential recommender with external memories. By explicitly setting up an external memory of storage slots, Memory Networks (MN) manipulate the memory according to the received data signal with a set of predefined operations, *e.g., read* and *write*. It has been shown that MNs are effective in memorizing long-term data characteristics [3], which can even evolve and update over time. We use KB information as external knowledge. Considering the structural organization of entity information in KBs, we propose to incorporate KB knowledge via Key-Value Memory Networks (KV-MN). KV-MNs [21] decompose each memory slot into a key vector and a value vector. A nice merit of KV-MN is that we can associate a key vector with a value vector, which supports associative search and read. With KV-MNs, we set a key vector to a relation embedding learned from KB data, corresponding to an entity attribute. Furthermore, given a key vector, we set up a user-specific value vector storing the preference characteristics of a user for the corresponding attribute. In this way, external KB knowledge is effectively incorporated into the KV-MNs. Once the knowledge-enhanced KV-MNs have been prepared, the next question is how to integrate it with RNN-based sequential recommender. Instead of simply merging the output from both components, at each recommendation, we use the sequential preference representation from RNNs as the query to read out the associated content of user-specific KV-MNs, *i.e.,* value vectors. Value vectors will be combined into an *attributed-based preference representation* with attentive weights derived from the sequential preference representation. The attributed-based preference representation together with sequential preference representation are combined as the final representation of user preference. We present the overview of the proposed model in Fig. 1.

To summarize, in this paper, we propose a novel knowledge-enhanced sequential recommender. Our model integrates RNN-based networks (GRU) with KV-MNs. RNN-based networks are good at capturing sequential user preference, while knowledge-enhanced KV-MNs are good at capturing attribute-based user preference. By using a hybrid of RNNs and KV-MNs, it is expected to be endowed with the benefits of both components. Given a hidden sequential preference representation from RNNs, our model is able to transform it into attentive weights over the key vectors corresponding to attributes, which provides attribute-level interpretability. By setting user-specific value vectors, our model is able to learn the characteristics of user preference on some specific attribute, which further provides value-level interpretability.

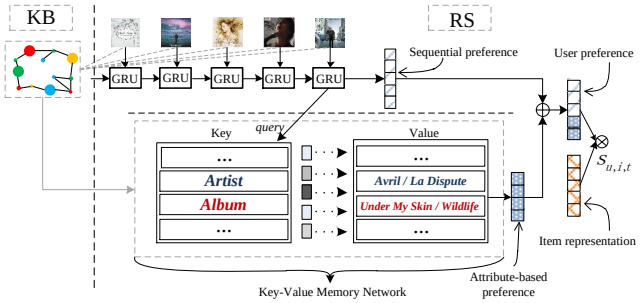


Figure 1: The overview of the proposed model. The model consists of two components, namely RNNs and KV-MNs. By linking RS items with existing KB entities, we enhance the semantic representation of KV-MNs. The RNN component is used to capture sequential preference, while the KV-MN component is used to capture attribute-based preference.

To our knowledge, it is the first time that sequential recommender is integrated with external memories by leveraging existing KB information. For evaluating our model, we prepare four RS datasets, and then link items of the four datasets with FREEBASE entities. Extensive results on the four datasets have shown the superiority of the proposed model in both effectiveness and interpretability.

2 RELATED WORK

Our work is closely related to the studies on recommender systems.

General Recommendation. Early works on recommender systems typically use collaborative filtering (CF) to make recommendations based on matching users with similar "tastes" or interests [13], such as K-Nearest Neighbor (kNN) algorithms [27] and Matrix Factorization (MF) algorithms [17]. The recommendation tasks can be divided into explicit feedback (*e.g.,* rating prediciton) and implicit feedback. For implicit feedback, BPR [25] optimizes the latent factor model with a pairwise ranking loss in a Bayesian framework. Recently, deep neural networks have also been used to enhance the capacity of modeling user-item interaction [12].

Sequential Recommendation. Sequential recommendation predicts the successive item(s) that a user is likely to adopt given her past adoption records. The major idea of previous methods is to capture sequential patterns between consecutive user-item interactions. A classic work is the FPMC model [26], which is a

hybrid model combining MC and MF for next basket recommendation. More recently, representation learning and deep learning have been made great progress, and many new techniques have been adapted to sequential recommendation. As two typical applications of representation learning, HRM [32] captures both sequential behavior and users' general taste by involving transaction and user representations in prediction, and TransRec[10] proposes that items are embedded into a "transition space" where each user is modeled by a translation vector. As one of the most popular neural networks, Recurrent Neural Networks (RNN) together with its variants LSTM and GRU have been widely applied to sequential recommendation, including session-based GRU [15], user-based GRU [5], dynamic recurrent model [36], recurrent recommender network [35], hierarchical personalized RNN model [22] and attention-based GRU [18]. RNN based methods represent users' preference by embedding the historical adoption records into a latent vector. To explicitly capture item- and feature-level sequential patterns, RUM [3] proposes to use memory network to improve sequential recommendation.

Context-aware Recommendation. With the rapid development of recommender systems, various context information has become available [31]. A typical approach to integrating context information in recommendation models is the feature-based MF, such as LibFM [24]. As several representative kinds of context, temporal information [38], social media information [40, 41] and knowledge information [37] have been utilized in recommendation. More recently, deep learning techniques have been utilized to enhance the modeling of context information, including multi-granularity attention [2], RNN with rich features [23], RNN with video semantic embedding [7] and RNN with adaptive context-specific transition matrices [20]. Especially, CKE [39] also uses KB information to improve the performance of collaborative filtering with deep learning. Our work is also related to the works on interpretable recommendation with context information [14, 30].

Our work is closely related to the studies on deep learning for recommender systems. Especially, it shares the common point with RUM [3] in that both have utilized MNs for sequential recommendation. Compared with RUM, our model adopts a separate GRU component for capturing sequential dependency and incorporates KB information for enhancing the modeling of attribute-level user preference. Our knowledge-enhanced memory networks further align key vectors with entity attributes, which makes the recommendation highly interpretable.

3 PROBLEM DEFINITION

We first introduce the notations used throughout the paper. In a recommender system (RS), let $\mathcal{U}$ denote a set of users and $\mathcal{I}$ denote a set of items. Our task focuses on the recommendation scenario with implicit feedback [25, 26], where we only concern whether a user $u \in \mathcal{U}$ has interacted with an item $i \in \mathcal{I}$ at time t. By sorting the interaction records by time ascendingly, we can form the *interaction sequence* for user u, namely $\{i_1^{(u)}, \cdots, i_t^{(u)}, \cdots, i_{n_u}^{(u)}\}$, where $i_t^{(u)}$ is the item that u has interacted with at time t and n_u is the length of interaction records for user u. Following [26], we use the relative time index instead of absolute time index for numbering interaction records.

Besides interaction sequences, we assume that a knowledge base (KB) is also available as the input. A KB is defined over an entity set $\mathcal{V}$ and a relation set $\mathcal{R}$, containing a set of KB triples. A KB triple $\langle e_1, r, e_2 \rangle$ denotes there exists relation $r \in \mathcal{R}$ between two entities e_1 and e_2 from $\mathcal{V}$, stating a fact stored in KB. For example, a KB triple (Avatar, directedBy, JamesCameron) describes that *Avatar* is directed by *James Cameron*. Since we assume it is possible to link RS items with KB entities, RS item set $\mathcal{I}$ can be considered as a subset of KB entity set $\mathcal{V}$, so we have $\mathcal{I} \subset \mathcal{V}$. By linking a RS item with a KB entity, we can obtain all its related KB triples.

Based on these preliminaries, we are ready to define the sequential recommendation task. Given the interaction sequence $\{i_1^{(u)}, \cdots, i_t^{(u)}, \cdots, i_{n_u}^{(u)}\}$ of user u, we would like to infer the item that user u will interact with at time $n_u + 1$. Note that it is straightforward to convert the above task setting into a basket-based [26] or session-based setting [15] by replacing each item $i_t^{(u)}$ by an item subset $\mathcal{I}_t^{(u)} \subset \mathcal{I}$, where $\mathcal{I}_t^{(u)}$ denotes the set of items that u has interacted with at time t. For simplicity and clarity, we keep the next-item setting as the major task setting throughout the paper.

4 THE PROPOSED APPROACH

In this section, we present the knowledge-enhanced sequential recommender. We start with a base sequential recommender using GRU networks, and then augment the base model with Key-Value Memory Networks using entity attribute information from KBs.

4.1 A GRU-based Sequential Recommender

Recurrent Neural Networks (RNN) have been shown effective in capturing and characterizing the temporal dependency in sequence data. A major problem of RNNs is that it suffers from the problem of "vanishing gradients" in dealing with long sequences. To alleviate this problem, two variants, namely the Long Short Term Memory (LSTM) networks [16] and Gated Recurrent Unit (GRU) networks [4], have been proposed. We adopt the GRU network as the base sequential recommender in our work, since it is simpler and contains fewer parameters than LSTM.

Given the interaction sequence $\{i_1, \cdots, i_t\}$[1] of user u, our GRU-based recommender computes the current hidden state vector $\boldsymbol{h}_t^u \in \mathbb{R}^{L_H}$ conditioned on previous hidden state vector $\boldsymbol{h}_{t-1}^u$ as below

$$\boldsymbol{h}_t^u = \text{GRU}(\boldsymbol{h}_{t-1}^u, \boldsymbol{q}_{i_t}; \Theta), \tag{1}$$

where $\text{GRU}(\cdot)$ is the GRU unit [4], $\boldsymbol{q}_{i_t}$ is the embedding vector for item i_t, and Θ denotes all the related parameters of GRU networks. The embedding vector $\boldsymbol{q}_{i_t} \in \mathbb{R}^{L_H}$ is called *item embedding*, which can be fixed or learned. In this way, the predictor encodes the interaction sequence of u into a hidden vector $\boldsymbol{h}_t^u$, which models the sequential preference of u at time t. Hence, we call $\boldsymbol{h}_t^u$ *sequential preference representation* of user u.

To generate the sequential recommendation, we rank a candidate item i by computing the recommendation score $s_{u,i,t}$ according to

$$s_{u,i,t} = g(u, i, t) = \boldsymbol{h}_t^{u\top} \cdot \boldsymbol{q}_i, \tag{2}$$

[1] We omit the superscript of u from the the item indices without loss of clarity.

where $g(u, i, t)$ is the ranking function implemented as the inner product between the sequential preference representation of u and item embedding of i at time t.

Once we obtain the ranking scores, we can recommend items with high scores to a target user.

4.2 Augmenting Sequential Recommender with Knowledge-Enhanced Memory Networks

The GRU-based recommender encodes the user preference into a latent vector, which is less powerful to capture fine-grained preference over attribute or feature dimensions of items. Knowing detailed user interests in the attribute level is particularly useful to improve the base recommender in both aspects of interpretability and performance. To address this issue, our idea is to incorporate entity attribute information from KB into the sequential recommender. Although GRU networks incorporate additional reset and update gates, they still have limited power to memorize and maintain long-term data information [34]. Inspired by recent works which integrate neural networks with external memories [19, 21], we propose to utilize Key-Value Memory Network (KV-MN) to maintain the KB knowledge, and then integrate the KV-MNs with the base sequential recommender.

4.2.1 Modeling Attribute-level User Preference with Key-Value Memory Networks. Memory Networks (MN) use an external memory, which is can be considered as a very large array of slots, for explicitly storing and memorizing information. With external memories, MNs are more capable of capturing and modeling long-term data characteristics [19]. In its original form, MNs treat memory slots functionally equal for external information storage. To further improve the storage of structured context or knowledge information, KV-MNs has been proposed to split a memory slot into a key vector and a value vector, and then associate the key vector with the value vector in a memory slot [21]. Such an architecture perfectly matches the structure of KB triples, which typically correspond to entity attribute information. By storing attribute information (*a.k.a.,* features) of items in the key vectors and attribute-specific user preference in value vectors, we are able to model long-term preference evolving in the attribute level.

We assume that an item set is associated with A kinds of attribute information, which are shared by all the items from the same domain. For example, in the domain of *Movie*, items share the attributes of actors, directors, genres, etc. Formally, we frame the user-specific KV-MNs as a set of vector pairs $\{(\boldsymbol{k}_1, \boldsymbol{v}_1^u), \cdots, (\boldsymbol{k}_A, \boldsymbol{v}_A^u)\}$, where $\boldsymbol{k}_a \in \mathbb{R}^{L_K}$ is the key vector for attribute a and $\boldsymbol{v}_a \in \mathbb{R}^{L_V}$ is the value vector corresponding to attribute a for user u. In this way, we can form a shared key memory matrix $\boldsymbol{K} \in \mathbb{R}^{L_K \times A}$ (called *key matrix* for short) and a user-specific value memory matrix $\boldsymbol{V}^u \in \mathbb{R}^{L_V \times A}$ (called *value matrix* for short) by combining key vectors or value vectors. It is noteworthy that the key matrix $\boldsymbol{K}$ is shared by all the users, since the key matrix summarizes the overall attribute-level characteristics of the item set. We leave the setting of attribute information for $\boldsymbol{K}$ in Section 4.2.2. The value matrix $\boldsymbol{V}^u$ is set to be privately used for each user u, since users are likely to have varying preferences over the shared attributes.

A high-level view of KV-MNs for our recommendation scenario is described as follows. At each time t, the sequential preference representation $\boldsymbol{h}_t^u$ from the GRU network in Eq. 1 is taken as the query to the KV-MNs, which is used to address and visit the memory of key vectors in $\boldsymbol{K}$, and then the associated value vectors are combined using some strategy as the return, called the Read operation. It is likely that $\boldsymbol{h}_t^u$ is not directly computable with the key vectors. Hence, we adopt a Multiple-Layer Perceptron to implement a non-linear transformation, *i.e.,* $\tilde{\boldsymbol{h}}_t^u = \text{MLP}(\boldsymbol{h}_t^u)$. With the transformed vector $\tilde{\boldsymbol{h}}_t^u$, the Read operation can be given in an abstractive form

$$\boldsymbol{m}_t^u \leftarrow \textsc{Read}(\{(\boldsymbol{k}_1, \boldsymbol{v}_1^u), \cdots, (\boldsymbol{k}_A, \boldsymbol{v}_A^u)\}, \tilde{\boldsymbol{h}}_t^u), \tag{3}$$

where $\boldsymbol{m}_t^u$ is a latent vector produced by the KV-MNs given the query $\tilde{\boldsymbol{h}}_t^u$, encoding the attribute-level preference characteristics of user u at time t. We call $\boldsymbol{m}_t^u$ *attribute-based preference representation* of user u. Indeed, as will be shown later, $\boldsymbol{m}_t^u$ can be roughly understood as a linear combination of the user-specific value vectors according to the preference weights over attributes for user u. In this way, we expect the representation of $\boldsymbol{h}_t^u$ emphasizes more on sequential preference, while the representation of $\boldsymbol{m}_t^u$ emphasizes more on attribute-based preference. The two parts complement each other, which is supposed to yield a better performance than either. Once the KV-MNs receives a new interaction record between user u and item i, the Write operation is run using the entity embedding of item i as a reference vector, and then update the associated user-specific value vectors according to some strategy

$$\{\boldsymbol{v}_1^u, \cdots, \boldsymbol{v}_A^u\}^{new} \leftarrow \textsc{Write}(\{(\boldsymbol{k}_1, \boldsymbol{v}_1^u), \cdots, (\boldsymbol{k}_A \boldsymbol{v}_A^u)\}^{old}, \boldsymbol{e}_i), \tag{4}$$

where $\boldsymbol{e}_i$ is some embedding representation of item i, which will be specified later. With the Read and Write operation, we can maintain and monitor the evolving process of attribute-level preferences for users. Note the key vectors will be pre-set and not updated.

4.2.2 Enhancing KV-MVs with KB Information. A key problem to be solved in Section 4.2.1 is how to set the key matrix $\boldsymbol{K}$ with appropriate attribute information from the item side. In the literature, various kinds of context information have been leveraged as useful signals for improving recommendation [7, 39]. Here, we propose to use KB information for setting the key matrix, which is able to flexibly characterize attribute information of entities from various domains. Many large-scale KBs have been released for public usage, such as Freebase [8] and Yago [29]. By linking RS items with existing KB entities, we are able to obtain rich attribute information of RS items from a variety of domains.

Given an item i, let e_i denote its corresponding entity in KB. Since KB is originally framed as a set of triples, we can obtain a set of related triples where e_i plays the head or tail entity. For effectively encoding KB information, we propose to learn a distributed vector $\boldsymbol{e}_i \in \mathbb{R}^{L_E}$ for entity e_i and $\boldsymbol{r} \in \mathbb{R}^{L_E}$ for relation r . To learn KB embedding, we use the simple and efficient model TransE [1] to minimize the loss of the triples $\sum_{\{\langle e_1, r, e_2\rangle\}} \| \boldsymbol{e}_1 + \boldsymbol{r} - \boldsymbol{e}_2 \|$ under suitable regularization constraints. The learned KB embeddings provide a general and compact representation for entities and relations, which is convenient to use and integrate for subsequent usage.

To this end, we have obtained the embeddings for both the entities and relations. KB relations usually correspond to attribute information of entities. Hence, we fill the key matrix by the relation embeddings, *i.e.,* $\boldsymbol{k}_a = \boldsymbol{r}_a$ for each possible attribute a (corresponding to a relation r_a in KB).

4.2.3 Instantiating the Read and Write Operations. Now, we are ready to instantiate the above READ and WRITE operations for the knowledge-enhanced KV-MN. For the READ operation, at time t, we use the following attentive combination for user u

$$\boldsymbol{m}_t^u \leftarrow \sum_{a=1}^{A} w_{t,u,a} \cdot \boldsymbol{v}_a^u, \tag{5}$$

where $w_{t,u,a}$ is the attention weight of the attribute a for user u at time t defined as

$$w_{t,u,a} = \frac{\exp(\gamma \tilde{\boldsymbol{h}}_t^u \cdot \boldsymbol{k}_a)}{\sum_{a'=1}^{A} \exp(\gamma \tilde{\boldsymbol{h}}_t^u \cdot \boldsymbol{k}_{a'})}, \tag{6}$$

where γ is a scaling factor and empirically set to 10 in our work.

For the WRITE operation, the case becomes a bit complicated. Our value vectors store the characteristic representations of user preference *w.r.t* some attribute. At each time of receiving a new interaction with item i by u, we need to decompose its KB embedding $\boldsymbol{e}_i$ into attribute-level updates. The update from item i relative to attribute a is computed as

$$\boldsymbol{e}_a^i = \boldsymbol{e}_i + \boldsymbol{r}_a, \tag{7}$$

where $\boldsymbol{e}_a^i \in \mathbb{R}^{L_E}$ is the update vector of item i for attribute a. The idea is based on the TRANSE model [1], in which we compute the loss of a triple $\langle e_1, r, e_2 \rangle$ by the distance $\| \boldsymbol{e}_1 + \boldsymbol{r} - \boldsymbol{e}_2 \|$. Hence, we can approximate the embedding of a tail entity (*i.e.,* attribute value) by the summation between the embeddings of the head entity and relation. Consider an example about the attribute of *director* for the movie *Avatar*. In TRANSE, $\boldsymbol{e}_{Avatar} + \boldsymbol{r}_{directedby} \approx \boldsymbol{e}_{JamesCameron}$, so we can use $\boldsymbol{e}_{Avatar} + \boldsymbol{r}_{directedby}$ to represent the entity of *James Cameron*. Note that we don't use the directly learned embedding $\boldsymbol{e}_{JamesCameron}$ for the entity "James Cameron", since there are many one-to-multiple relations in KB, where the attribute value can correspond to a set of entities, *e.g.,* actors of "Avatar".[2] Similar to the WRITE operation used in [34], we first compute a gate vector $\boldsymbol{z} \in \mathbb{R}^A$ to determine the proportion of information that is to be updated for each attribute in user-specific value vectors. The gate weight $z_a \in \boldsymbol{z}$ for each attribute a is computed as

$$z_a = \text{sigmoid}(\boldsymbol{v}_a^{u\top} \cdot \boldsymbol{e}_a^i). \tag{8}$$

With the update weight z_a and update vector $\boldsymbol{e}_a^i$, we update each value vector in the value matrix $\boldsymbol{V}^u$ of user u accordingly

$$\boldsymbol{v}_a^u \leftarrow (1 - z_a) \cdot \boldsymbol{v}_a^u + z_a \cdot \boldsymbol{e}_a^i. \tag{9}$$

Once the update process has been completed, the value matrix $\boldsymbol{V}^u$ stores the user preference over explicit entity attributes. Our update operation makes it possible to dynamically monitor and maintain such a long-term user preference in the attribute level.

[2] Another alternative method is to directly integrate the embeddings of multiple value entities. However, it is difficult to learn or set the combination weights for different entities. We leave this part as future work.

4.3 The Complete Sequential Recommender

Our complete sequential recommender is a hybrid of GRU networks and knowledge-enhanced KV-MNs. Given the interaction sequence $\{i_1, \cdots, i_t\}$ of user u, we first adopt the GRU network to derive the sequential preference representation $\boldsymbol{h}_t^u$ using Eq. 1. Then, we use the transformed $\tilde{\boldsymbol{h}}_t^u$ as the query to read the KV-MN, and obtain the corresponding attribute-based preference representation $\boldsymbol{m}_t^u$ using Eq. 5. We use a vector concatenation to combine both representations into a single vector $\boldsymbol{p}_t^u = \boldsymbol{h}_t^u \oplus \boldsymbol{m}_t^u$ for modeling user preference of user u at time t. For the item side, we further concatenate the item embedding $\boldsymbol{q}_i$ in RS and the entity embedding $\boldsymbol{e}_i$ in KBs, namely $\tilde{\boldsymbol{q}}_i = \boldsymbol{q}_i \oplus \boldsymbol{e}_i$. $\boldsymbol{p}_t^u$ and $\tilde{\boldsymbol{q}}_i$ have the same size of $L_H + L_E$. Similar to Eq. 2, we use the inner product between the new representations for users and items as the ranking score,

$$s_{u,i,t} = g(u, i, t) = \text{MLP}(\boldsymbol{p}_t^u)^\top \cdot \text{MLP}(\tilde{\boldsymbol{q}}_i), \tag{10}$$

where $MLP(\cdot)$ is a multilayer perceptron consisting of hidden layers with *tanh* as the activation function. $MLP(\cdot)$ transforms an input vector into an output vector, where both vectors have the same dimension number. Here, we incorporate non-linear transformation to map $\boldsymbol{p}_t^u$ and $\tilde{\boldsymbol{q}}_i$ into the same space.

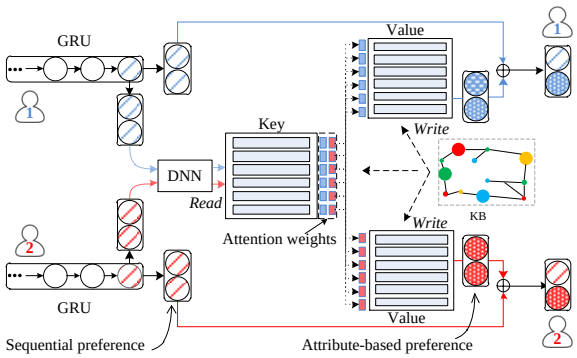


Figure 2: The schematic illustration of the working mechanism of our model.

We present a diagram sketch of our model in Fig. 2. We call our model *Knowledge-enhanced Sequential Recommender* (***KSR***). Our model has the following merits. First, the GRU network is able to effectively capture temporal dependency, yielding a sequential representation for user preference (*i.e.,* $\boldsymbol{h}_t^u$). Second, the KV-MN part is able to characterize the detailed user interests over item attributes, yielding an attribute-based representation for user preference (*i.e.,* $\boldsymbol{m}_t^u$). Third, the *hidden* sequential preference representation (*i.e.,* $\boldsymbol{h}_t^u$) is used to dynamically generate a set of attention weights (*i.e.,* $w_{t,u,a}$) over the *explicit* attributes, which provides the capacity of explaining the latent sequential preference in the attribute level. Putting all together, our model is endowed with the benefits from both GRU and KV-MN, and further enhanced with external structured knowledge information. Hence, our model is expected

to be more powerful in sequential recommendation, effective and interpretable.

In our model, the parameters to learn are from both GRU and KV-MN components. We pre-train the item embedding $\boldsymbol{q}_i$ by using the classic BPR model [25], and fix them in the learning process. We find that such a pre-training technique is important to improve the performance of sequential recommendation. For entity and relation embeddings (*i.e.,* $\boldsymbol{e}_i$ and $\boldsymbol{r}_a$), we learn them using the classic TransE model, and fix them in the learning process. We assume that the key vectors are shared by all the users, and set them to the relation embeddings and fix them in the learning process. While, for other parameters, we adopt the pairwise loss following BPR [25]

$$L = \sum_{u \in \mathcal{U}} \sum_{t=2}^{n_u} \sum_{j \in \mathcal{I}_u^-} \log \sigma(g(u, i_t) - g(u, j)), \tag{11}$$

where n_u is the length of interaction sequence of u in the training set, $\mathcal{I}_u^-$ is a small set of sampled negative items that user u has not interacted with, and $\sigma(\cdot)$ is the sigmoid function. We implement the model with the library of Theano by using *AdaGrad* optimization [6] in mini batches.

5 EXPERIMENTS

In this section, we first set up the experiments, and then present the performance comparison and analysis.

5.1 Experimental Setup

Construction of the Datasets. In our task, we need to prepare both KB and RS data. For KB data, we adopt the one-time Freebase [8] dump consisting of 63 million triples. For RS data, we use four datasets from different domains, namely Last.fm music [28], MovieLens ml-20m [9], MovieLens ml-1m [9] and Amazon book [11]. The Last.fm music dataset is very large, and we take the subset from the last year; for the ml-20m dataset, we take the subset from year 2005 to 2015. Following [10, 26], we only keep the k-core dataset, and filter unpopular items and inactive users with fewer than k records, which is set to 3 in book dataset and 10 for the other datasets. Then, we link filtered items with Freebase entities. With an offline Freebase search API, we retrieve KB entities with item title (*e.g.,* song titles) as queries. Once mulitple entities are returned, we further incorporate at least one attribute as the filter to identify the only correct entity. We only keep the interactions related to the linked items in the final datasets. We group the interaction records by users, sort them according to the timestamps ascendingly, and form the interaction sequence for each user. To train TransE, we start with linked entities as seeds and expand the graph with one-step search. Not all the relations in KBs are useful, we remove unfrequent relations with fewer than 5,000 triples. We summarize the detailed statistics of the datasets in Table 1.

Task Settings. We consider two task settings for evaluation, namely next-item recommendation and next-session recommendation. We fully follow the previous settings [12, 25, 26]. For next-item recommendation, we hold out the last item of the interaction sequence as the test data; For next-session recommendation, we hold out the items from the last session in the interaction sequence as the test data, in which a day is considered as a session. The item or session just before the last has been treated as the validation set. The rest data is treated as the training data. Since our item set is large, it will be time-consuming to enumerate all the items as candidate. Hence, following [12], for each positive item in the test set, we pair it with 100 sampled items that the user has not interacted with, called *negative items*. To make the sampling reliable and representative, out of the 100 negative items, 50 items are sampled randomly, while the rest 50 items are sampled according to the popularity.

Table 1: Statistics of our datasets. #Entities indicates the number of entities that are extended by seed entities with one-step search in KBs for training TransE.

Datasets	#Interactions	#LinkedItems	#Users	#Entities	#Relations
Music	203,975	30,658	7,694	214,524	19
ml-20m	5,868,015	19,533	61,583	1,125,100	81
ml-1m	916,714	3,210	6,040	1,125,100	81
Book	828,560	69,975	65,125	313,956	49

Evaluation Metrics. To evaluate our approach, we adopt a variety of evaluation metrics widely used in previous works [3, 15, 26], including the Mean Average Precision (MAP), the Mean Reciprocal Rank (MRR), Hit Ratio (HR), and Normalized Discounted cumulative gain (NDCG). Note that we don't report the results of Precision@k and Recall@k, since NDCG@k is an improved measure of Precision@k and HR@k is more suitable than Recall@k for our tasks which have very few ground-truth results. We report the average score for all test users.

Methods to Compare. We consider the following methods for performance comparison:

- **BPR** [25]: It optimizes the latent factor model with implicit feedback using a pairwise ranking loss in a Bayesian approach.
- **NCF** [12]: Based on MF, it replaces the inner product with a neural architecture that can learn an arbitrary function from data.
- **CKE** [39]: It first proposes to incorporate KB and other information (*i.e.,* image and text) to improve the recommendation performance. For fairness, we implement a simplified version of CKE by only using the KB information, and exclude the image and text information.
- **FPMC** [26]: It is a classic hybrid model combining MC and MF for next-basket recommendation, which can capture both sequential effects and general interests of users.
- **RUM** [3]: It first proposes to utilize external memories to improve sequential recommendation, which contains two variants, either item-level (RUM^I) or feature-level (RUM^F). Originally, it only characterizes latent features, and we further incorporate KB embeddings to enhance the model.
- **GRU** [15]: It implements an improved version of the GRU network for session-based recommendation, which utilizes session-parallel mini-batch training process and also employs ranking-based loss functions for learning the model.
- **GRU++**: We improve the above GRU model by using the pre-training technique for the items. We train the latent item vectors using the BPR model [25], and use the learned item

representations to set the item embeddings in GRU++. The item embeddings are fixed in the learning process.
- **GRU**$_F$: Quadrana et al. [23] propose to incorporate auxiliary features into GRU networks for improving the sequential recommendation. One of their proposals is to concatenate both one-hot item vector and feature vector as the input vector of GRU networks. We implement an enhanced version by replacing one-hot item vector with pre-trained BPR item vector. The KB embedding is taken as the feature vector.
- **KSR**: It is our model introduced in Section 4.

Our baselines have a comprehensive coverage of the related models. To summarize, we categorize the baselines into eight groups shown in Table 2, according to the *task orientation*, *with/without KB* and *with/without neural models*.

Table 2: The categorization of the comparison methods.

Tasks	KB	Neural (No)	Neural (Yes)
General	Yes	–	CKE
	No	BPR	NCF
Sequential	Yes	–	RUM, GRU$_F$, **KSR**
	No	FPMC	GRU, GRU++

Parameter Setting. All the models have some parameters to tune. We either follow the reported optimal parameter settings or optimize each model separately using the validation set. For our model, we adopt a one-layer GRU network, the hidden layer size L_H is set to 256, the item embedding size is set to 256, the KB embedding size L_E with TransE is set to 50, and the key vector size L_K and the value vector size L_V are set to 50. We will discuss how parameter setting affects the final performance in Section 5.3.

Table 4: The MAP results of the variants with shared or private value matrices for our model.

Methods	Next-Item				Next-Session			
	Music	ml-1m	ml-20m	Book	Music	ml-1m	ml-20m	Book
Shared	0.416	0.351	0.291	0.348	0.218	0.273	0.130	0.341
Private	0.428	0.356	0.294	0.353	0.224	0.276	0.135	0.345

5.2 Results and Analysis

The results of different methods for both sequential recommendation tasks are presented in Table 3. It can be observed that:

(1) Among non-sequential recommendation baselines, BPR performs well on the two dense movie datasets, but poorly on music and book datasets, which are more sparse. Overall, NCF and CKE perform better BPR in more cases, since NCF incorporates a neural architecture to characterize arbitrary user-item interactions, and CKE utilizes the KB embedding as the additional signal to enhance the modeling of sparse user-item interactions. Indeed, by excluding text and image components from CKE, it degenerates into an extended BPR, where a part of the latent item representation is set to the KB embedding of items.

(2) Among sequential recommendation baselines, the classic model FPMC performs worst (but still better than BPR in most cases). FPMC has adopted the similar optimization way as BPR, and the major difference is FPMC further incorporates item-item sequential relatedness in the optimization. The recently proposed RUM model yields a better performance than all the above baselines. RUM adopts the KV-MN architecture for sequential recommendation. For fairness we have also set the key matrix in RUM with attribute embeddings and update the value matrix with KB embeddings of items. We have found that item-level variant RUM^I overall performs better than feature-level RUM^F. A possible reason is that we set the item representations in RUM^I with KB embeddings. Finally, we examine the performance of the three GRU-based sequential recommenders. As we can see, the GRU++ model beats all the other baselines except GRU$_F$ in four datasets. We find that pre-training the item embedding is particularly useful, which can significantly boost the performance. A possible reason is that GRU can't well model long-term user interests. Setting parameters with BPR can alleviate this weakness, since BPR is able to capture overall user and item representations. By incorporating additional features, GRU$_F$ overall works slightly better than GRU++, but the improvement is not large. We speculate that the simple concatenation of item and feature vectors may not be the most suitable way for incorporating auxiliary features.

(3) Finally, we compare our proposed model KSR with all the baselines. It is clear to see that KSR is consistently better than these baselines by a large margin. Our base architecture is the pre-trained GRU network [15], and then we incorporate a knowledge-enhanced KV-MN. Roughly speaking, KSR have the merits of both GRU++ and KV-MN: GRU++ is more capable of characterizing sequential dependency and KV-MN is more capable of characterizing attribute-level preference with KB knowledge. Our experiments indicate these two aspects are important to improve the sequential recommendation. Another potential benefit of KSR over all the baselines is that the recommendation results are highly interpretable, and we will show this in Section 5.4.

5.3 Detailed Analysis of Our Model KSR

Above, we present the main result comparisons of different models. Our model has achieved a significant improvement over all the baselines. In this part, we construct detailed analysis of our model for better understanding why and how it will work. Due to space limit, unless specified, we only report the MAP results of next-item recommendation on the book dataset, while the results on the other three datasets with other metrics are similar and omitted. Among all the baselines, the pre-trained GRU++ performs very well and stably. Hence, we incorporate GRU++ as the only reference baseline for ease of comparison.

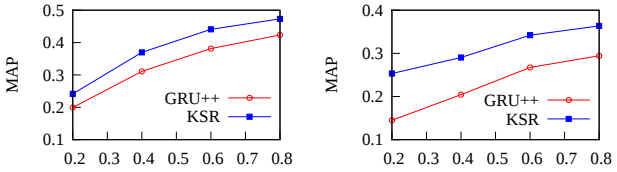
Personalized or Shared Value Matrix. Our KV-MNs consists of a shared key matrix and user-specific (called *private*) value matrices. Now, we study how the configuration of value matrices affects the performance of our model. We implement another variant of KSR with *a shared value matrix* by all the users in the KV-MN. Table 4 presents the MAP results of the two variants on the four datasets. It can be observed that using private value matrices is better than the other variant. The results confirm to our intuition that different users may have varying preference characteristics over item attributes. While, the improvement seems not that large. A major reason is that at each time the sequential preference representation

Table 3: Performance comparison of different methods on next-item and next-session recommendation. "†" indicates the improvement of the KSR over the baseline is significant at the level of 0.01. We also report the improvement ratio of our model over the best performance of all the baselines for each dataset in parentheses.

Datasets	Methods	Next-Item Recommendation				Next-Session Recommendation			
		MAP	MRR	Hit@10	NDCG@10	MAP	MRR	Hit@10	NDCG@10
ml-20m	BPR	0.128†	0.128†	0.276†	0.144†	0.086†	0.165†	0.340†	0.099†
	NCF	0.094†	0.094†	0.205†	0.101†	0.083†	0.162†	0.354†	0.095†
	CKE	0.178†	0.178†	0.382†	0.209†	0.087†	0.153†	0.345†	0.104†
	FPMC	0.129†	0.129†	0.273†	0.144†	0.084†	0.157†	0.328†	0.096†
	RUM^I	0.267†	0.267†	0.523†	0.312†	0.122†	0.193†	0.399†	0.141†
	RUM^F	0.248†	0.248†	0.515†	0.295†	0.121†	0.197†	0.399†	0.141†
	GRU	0.282†	0.282†	0.522†	0.325†	0.079†	0.121†	0.264†	0.085†
	GRU++	0.277†	0.277†	0.549†	0.327†	0.127†	0.195†	0.397†	0.145†
	GRU_F	0.279†	0.279†	0.550†	0.329†	0.127†	0.197†	0.397†	0.146†
	KSR	**0.294** (+6.1%)	**0.294** (+6.1%)	**0.571** (+4.0%)	**0.344** (+5.2%)	**0.135** (+6.3%)	**0.209** (+7.2%)	**0.419** (+5.5%)	**0.156** (+7.6%)
ml-1m	BPR	0.178†	0.178†	0.396†	0.211†	0.171†	0.231†	0.472†	0.210†
	NCF	0.163†	0.163†	0.355†	0.189†	0.141†	0.192†	0.414†	0.172†
	CKE	0.158†	0.158†	0.350†	0.185†	0.134†	0.174†	0.366†	0.160†
	FPMC	0.305†	0.305†	0.549†	0.349†	0.245†	0.287†	0.517†	0.282†
	RUM^I	0.323†	0.323†	0.627†	0.382†	0.252†	0.290†	0.560†	0.298†
	RUM^F	0.263†	0.263†	0.577†	0.323†	0.220†	0.265†	0.555†	0.270†
	GRU	0.315†	0.315†	0.593†	0.368†	0.210†	0.222†	0.439†	0.241†
	GRU++	0.336†	0.336†	0.626†	0.393†	0.256†	0.291†	0.555†	0.304†
	GRU_F	0.340†	0.340†	0.636†	0.399†	0.259†	0.293†	0.559†	0.306†
	KSR	**0.356** (+6.0%)	**0.356** (+6.0%)	**0.655** (+4.6%)	**0.417** (+6.1%)	**0.276** (+7.8%)	**0.313** (+7.6%)	**0.570** (+2.7%)	**0.324** (+6.6%)
Music	BPR	0.227†	0.227†	0.458†	0.265†	0.151†	0.157†	0.320†	0.163†
	NCF	0.386†	0.386†	0.549†	0.413†	0.206†	0.228†	0.378†	0.224†
	CKE	0.371†	0.371†	0.541†	0.399†	0.215†	0.225†	0.386†	0.233†
	FPMC	0.349†	0.349†	0.489†	0.369†	0.140†	0.158†	0.290†	0.151†
	RUM^I	0.386†	0.386†	0.587†	0.422†	0.210†	0.220†	0.395†	0.229†
	RUM^F	0.332†	0.332†	0.562†	0.374†	0.201†	0.212†	0.399†	0.222†
	GRU	0.420†	0.420†	0.538†	0.436†	0.104†	0.131†	0.232†	0.112†
	GRU++	0.403†	0.403†	0.595†	0.437†	0.214†	0.224†	0.397†	0.233†
	GRU_F	0.404†	0.404†	0.594†	0.438†	0.214†	0.225†	0.397†	0.233†
	KSR	**0.427** (+1.7%)	**0.427** (+1.7%)	**0.607** (+2.0%)	**0.460** (+5.5%)	**0.223** (+4.2%)	**0.233** (+4.0%)	**0.403** (+1.5%)	**0.241** (+3.4%)
Book	BPR	0.222†	0.222†	0.505†	0.272†	0.216†	0.221†	0.505†	0.265†
	NCF	0.284†	0.284†	0.513†	0.325†	0.282†	0.290†	0.534†	0.327†
	CKE	0.248†	0.248†	0.494†	0.291†	0.246†	0.252†	0.512†	0.292†
	FPMC	0.147†	0.147†	0.324†	0.171†	0.149†	0.153†	0.338†	0.174†
	RUM^I	0.292†	0.292†	0.596†	0.350†	0.282†	0.288†	0.597†	0.341†
	RUM^F	0.300†	0.300†	0.610†	0.360†	0.278†	0.284†	0.590†	0.335†
	GRU	0.265†	0.265†	0.501†	0.305†	0.141†	0.144†	0.291†	0.157†
	GRU++	0.305†	0.305†	0.619†	0.366†	0.299†	0.305†	0.621†	0.360†
	GRU_F	0.306†	0.306†	0.619†	0.367†	0.299†	0.305†	0.620†	0.360†
	KSR	**0.353** (+15.7%)	**0.353** (+15.7%)	**0.653** (+5.5%)	**0.413** (+12.8%)	**0.345** (+15.4%)	**0.353** (+15.7%)	**0.661** (+6.4%)	**0.407** (+13.1%)

will be used to generate dynamic personalized attention weights. Even the value matrix is shared, the personalized attention weights still provide a flexible mechanism to model varying user preferences. Hence, when efficiency is more important to consider than performance, we can adopt the variant with shared value matrix for reducing model complexity. As will be shown in Section 5.4, using private value matrices is also helpful to improve the interpretability.

Varying the Amount of Training Data. Since our model KSR involves GRU networks and KV-MNs, it contains more parameters to learn and has a higher model complexity than baselines. We study how the performance of KSR model changes with the varying amount of training data. To examine this, we take 20%, 40%, 60% and 80% from the complete training data to generate four new training sets. The test sets can be constructed accordingly. Figure 3 presents the performance tuning *w.r.t.* different ratios of training data. It can be seen that KSR is consistently better than GRU++ with four training sets. Although our model has a more complicated architecture, it is well pre-trained and many parameters related to KB embeddings are fixed, which largely reduces the complexity in practice.

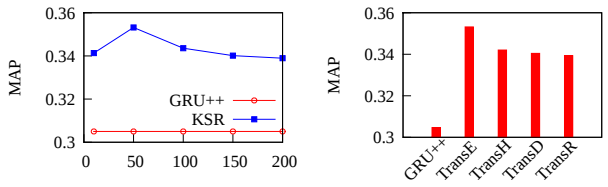


(a) Music dataset. (b) Book dataset.

Figure 3: Performance comparison by varying the amount of training data.

Varying KB Embeddings. An important data resource in our model is the trained KB embeddings. We now study how different embedding methods with varying configurations affect recommendation

performance. We adopt the open source toolkit of OpenKE[3] to implement four KB embedding methods [33], including TransD, TransE, TransH and TransR. We vary the KB embedding size in {50, 100, 150, 200}. In Fig. 4(a), we can see that an embedding size of 50 gives the best performance for TransE. By tuning the embedding size on the other KB embedding methods, we also find an embedding size of 50 overall works well. Then we fix the embedding size as 50, and compare different KB embedding methods. As shown in Fig. 4(b), TransE performs best among all the methods. A possible explanation is that TransE is simpler than the other variants, and its results are more stable on our tasks.



(a) Varying the embedding size of TransE.

(b) Comparing different embedding methods.

Figure 4: Performance tuning with different KB embedding methods.

5.4 Qualitative Analysis on the Recommendation Interpretability

In the previous experiments, we have shown that our model is more capable of generating high-quality sequential recommendations. Another major benefit is that our recommendations are highly interpretable due to the incorporation of KB information in the KV-MNs. Recall that we use $\tilde{\boldsymbol{h}}_t^u$ to compute the dynamic attention weights over the attributes using Eq. 6. Assume we have A attributes in total, our model can produce a distribution of attribute weights for user u at time t, *i.e.,* $\{w_{t,u,a}\}_{a=1}^{A}$. The attribute weights directly provide an attribute-level interpretation for the latent user representation $\boldsymbol{h}_t^u$. Furthermore, the user-specific value vector $\boldsymbol{v}_a^u$ maintains the characteristics of user preference on some attribute a, which further provides a value-level interpretation. To see this, we present an example from the music dataset in Fig. 5.

Attribute-level Interpretability. Fig. 5 presents an interaction sequence of five records from a sample user. Each record consists of two parts: the left part corresponds to the learned attention weights and the right part corresponds to ground-truth information of a song, including title, singer and album. First, the user started with two songs from the same album, which followed the way of listening by *album*. Then, she listened to two more songs from another album. For the fifth song, the user switched to a third album. The interesting point is that its singer is the same as that of previous two songs. Hence, for the last three songs, the user essentially followed a mixture of listening by *album* and listening by *singer*. It is clear that our model has predicted a larger weight on the attribute of *album* till the fourth record, and a larger weight on the attribute of *singer* on the fifth song. This example indicates the user preference is likely to be dynamic and evolving, and our model is able to capture evolving preference over the attributes.

Value-level Interpretability. Suppose it is already known some attribute (*e.g.,* album) plays the key role in determining the interaction behavior of a user, can we further predict how the user will select among a set of entities for that attribute (*e.g.,* the selection of the favorite album in candidate albums)? For convenience, we call the entities (also in KB) corresponding to the attribute value of a RS item *value entities*, *e.g., Deafheaven* is the value entity of attribute *singer* for song *The Pecan Tree*. Recall we have a user-specific value matrix in KV-MNs, which maintains the preference characteristics of a user on some specific attribute. We expect a value vector can reflect user preference over value entities for some attribute. A value vector $\boldsymbol{v}_a^u$ corresponds to a key vector $\boldsymbol{k}_a$ on attribute a. Since the value matrix is updated with KB embeddings of items (Eq. 7 and 9), the learned value vectors $\boldsymbol{v}_a^u$ can be represented in the same space as KB embeddings. Given an attribute, we can directly compute L_1 distance between the embedding of a candidate value entity (*e.g.,* $\boldsymbol{e}_{Deafheaven}$) and the user-specific value vector (*e.g.,* $\boldsymbol{v}_{singer}^u$) from the previous timestamp. Then, we rank the candidate value entities according to the L_1 distance and form a predication ranklist. We present the illustration of value-level interpretation at the bottom of Fig. 5. At the beginning (t_1), the value matrix is not well learned. By training with more records, our value matrix is able to dynamically trace the user preference on some specific attribute. At the fifth record (t_5), it correctly predicts the candidate entities for both *singer* and *album* attributes at the first position.

6 CONCLUSIONS

In this paper, we proposed to extend the GRU-based sequential recommender by integrating it with knowledge-enhanced KV-MNs. Our model was endowed with the benefits of these two components. By heuristically linking RS items with existing KB entities, we leveraged large-scale KB information to improve sequential recommendation. We enhanced the semantic representation in KV-MNs with entity attribute information from KB, which made the recommendations highly interpretable. We constructed four large linked datasets from RS with KBs. The results showed that our model is superior to previous methods in terms of effectiveness and interpretability. Currently, we consider three domains with four datasets, but we believe our approach is applicable to more domains. We will investigate into how our models perform in other domains. In practice, unstructured data or noisy context information is easier to obtain than well-formatted KB information, we will consider extending our model by utilizing such weak signals.

ACKNOWLEDGEMENT

The work was partially supported by National Natural Science Foundation of China under the Grant Number 61502502, Beijing Natural Science Foundation under the Grant Number 4162032, the National Key Basic Research Program (973 Program) of China under Grant No. 2014CB340403, and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG22.

[3] http://openke.thunlp.org/

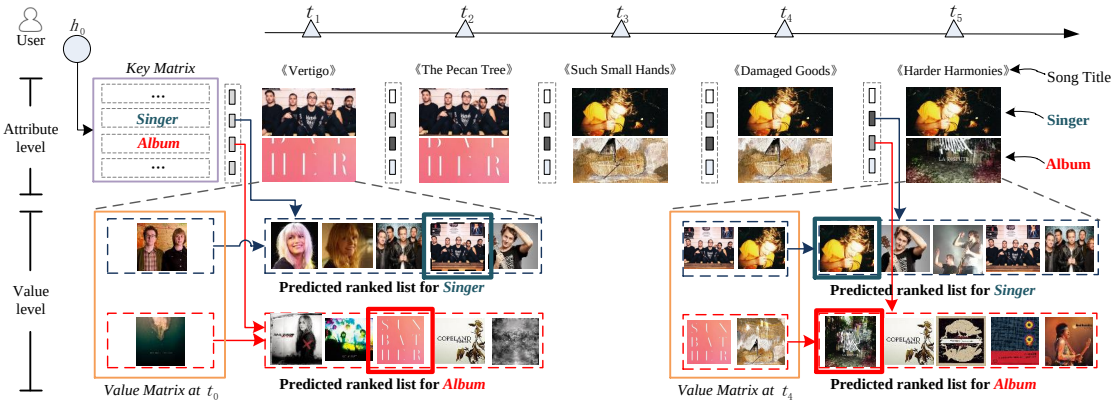


Figure 5: An interaction sequence from a sample user in music dataset. We use dark blue and red to indicate attributes of *album* and *singer* respectively. We present the predictions of our model KSR on attribute weights and value entities. For attention weights (top of the figure), we use color darkness to indicate the value of attention weights: darker is larger. For value entities (bottom of the figure), we show the predicted ranklist of candidate entities for both attributes at time t_1 and t_5 (using value matrices of KV-MN at t_0 and t_4). We highlight the correct entities in predicted ranklists with colored boxes.

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.

[2] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Wei Liu, Wei Liu, and Tat Seng Chua. 2017. Attentive Collaborative Filtering: Multimedia Recommendation with Item- and Component-Level Attention. In *SIGIR*. 335–344.

[3] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential Recommendation with User Memory Networks. In *WSDM*.

[4] Kyunghyun Cho, Bart Van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *Computer Science* (2014).

[5] Tim Donkers, Benedikt Loepp, and JÍźrgen Ziegler. 2017. Sequential User-based Recurrent Neural Network Recommendations. In *ACM RecSys*.

[6] John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[7] Junyu Gao, Tianzhu Zhang, and Changsheng Xu. 2017. A Unified Personalized Video Recommendation via Dynamic Recurrent Neural Networks. In *ACM MM*.

[8] Google. 2016. Freebase Data Dumps. https://developers.google.com/freebase/data. (2016).

[9] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets. *TiiS* 5, 4 (2016), 1–19.

[10] Ruining He, Wang Cheng Kang, and Julian Mcauley. 2017. Translation-based Recommendation. In *ACM RecSys*.

[11] Ruining He and Julian Mcauley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *WWW*.

[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.

[13] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *SIGIR*. 230–237.

[14] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. 2000. Explaining collaborative filtering recommendations. In *CSCW*. 241–250.

[15] BalĺĆzs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based Recommendations with Recurrent Neural Networks. *Computer Science* (2015).

[16] Sepp Hochreiter and JÍźrgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* (1997), 1735–1780.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* (2009), 30–37.

[18] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *CIKM*.

[19] Fei Liu and Julien Perez. 2016. Gated End-to-End Memory Networks. In *EACL*.

[20] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-Aware Sequential Recommendation. In *ICDM*. 1053–1058.

[21] Alexander Miller, Adam Fisch, Jesse Dodge, Amir Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-Value Memory Networks for Directly Reading Documents. In *EMNLP*. 1400–1409.

[22] Massimo Quadrana, Alexandros Karatzoglou, BalĺĆzs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In *RecSys*. 130–137.

[23] Massimo Quadrana, Domonkos Tikk, and Domonkos Tikk. 2016. Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In *ACM Conference on Recommender Systems*. 241–248.

[24] Steffen Rendle. 2012. Factorization Machines with libFM. *ACM TIST* (2012).

[25] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*.

[26] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *WWW*.

[27] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms. In *WWW*.

[28] Markus Schedl. 2016. The LFM-1b Dataset for Music Retrieval and Recommendation. In *ICMR*.

[29] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In *WWW*. 697–706.

[30] Nava Tintarev and Judith Masthoff. 2007. A Survey of Explanations in Recommender Systems. In *ICDE*. 801–810.

[31] Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, and Erik Duval. 2012. Context-Aware Recommender Systems for Learning: A Survey and Future Challenges. *TLT* (2012), 318–335.

[32] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning Hierarchical Representation Model for NextBasket Recommendation. In *SIGIR*. 403–412.

[33] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE TKDE* (2017).

[34] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory Networks. *Eprint Arxiv* (2014).

[35] Chao Yuan Wu, Amr Ahmed, Alex Beutel, How Jing, and How Jing. 2017. Recurrent Recommender Networks. In *WSDM*. 495–503.

[36] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A Dynamic Recurrent Model for Next Basket Recommendation. In *SIGIR*. 729–732.

[37] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: a heterogeneous information network approach. In *WSDM*. 283–292.

[38] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat-Thalmann. 2013. Time-aware point-of-interest recommendation. In *SIGIR*. 363–372.

[39] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *KDD*.

[40] Wayne Xin Zhao, Yanwei Guo, Yulan He, Han Jiang, Yuexin Wu, and Xiaoming Li. 2014. We know what you want to buy: a demographic-based system for product recommendation on microblogs. In *KDD*.

[41] Wayne Xin Zhao, Sui Li, Yulan He, Edward Y. Chang, Ji-Rong Wen, and Xiaoming Li. 2016. Connecting Social Media to E-Commerce: Cold-Start Product Recommendation using Microblogging Information. *TKDE* (2016).