Sequential Recommendation with User Memory Networks

Xu Chen*
TNList, School of Software
Tsinghua University
xu-ch14@mails.tsinghua.edu.cn

Hongteng Xu
College of Computing
Georgia Institute of Technology
hxu42@gatech.edu

Yongfeng Zhang
Department of Computer Science
Rutgers University
zhangyf07@gmail.com

Jiaxi Tang
School of Computing Science
Simon Fraser University
jiaxit@sfu.ca

Yixin Cao
Department of Computer Science and Technology, Tsinghua University
caoyixin2009@163.com

Zheng Qin†
School of Software
Tsinghua University
qinzh@mails.tsinghua.edu.cn

Hongyuan Zha
College of Computing
Georgia Institute of Technology
zha@cc.gatech.edu

ABSTRACT

User preferences are usually dynamic in real-world recommender systems, and a user's historical behavior records may not be equally important when predicting his/her future interests. Existing recommendation algorithms – including both shallow and deep approaches – usually embed a user's historical records into a single latent vector/representation, which may have lost the per item- or feature-level correlations between a user's historical records and future interests. In this paper, we aim to express, store, and manipulate users' historical records in a more explicit, dynamic, and effective manner. To do so, we introduce the memory mechanism to recommender systems. Specifically, we design a memory-augmented neural network (MANN) integrated with the insights of collaborative filtering for recommendation. By leveraging the external memory matrix in MANN, we store and update users' historical records explicitly, which enhances the expressiveness of the model. We further adapt our framework to both item- and feature-level versions, and design the corresponding memory reading/writing operations according to the nature of personalized recommendation scenarios. Compared with state-of-the-art methods that consider users' sequential behavior for recommendation, e.g., sequential recommenders with recurrent neural networks (RNN) or Markov chains, our method achieves significantly and consistently better performance on four real-world datasets. Moreover, experimental analyses show that our method is able to extract the intuitive patterns of how users' future actions are affected by previous behaviors.

KEYWORDS

Sequential Recommendation; Memory Networks; Collaborative Filtering

ACM Reference Format:
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 WSDM'18.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3159652.3159668

1 INTRODUCTION

In many real-world applications, users' current interests are influenced by their historical behaviors. For example, one may purchase accessories such as phone cases or earphones after buying a smart phone; and people may continue to buy the same brand of clothes that they had a good previous experience.

To model this phenomenon, previous methods have been proposed to make sequential recommendations with user historical records. For example, [23] adopted Markov chain to model user behavior sequences, and [19, 31] leveraged recurrent neural networks (RNNs) to embed previously purchased products for current interest prediction.

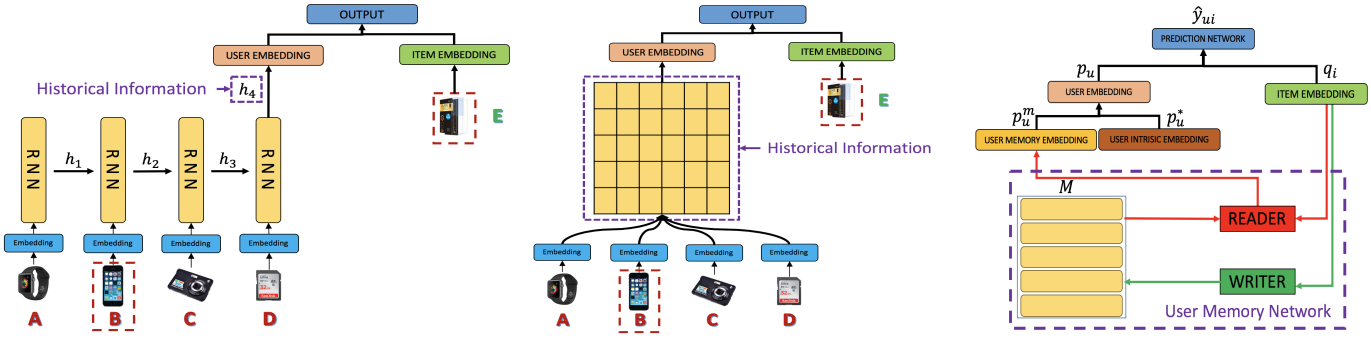
Existing methods have achieved encouraging results, however, they tend to compress all of a user's previous records into a fixed hidden representation. As exampled in Figure 1(a), the key reason for a user to buy a phone case (item E) could be that he bought a phone (item B) before, while the other prior purchases are not necessarily related to the new purchase. However, RNN (and other latent representation approaches) would forcefully summarize all of the prior items (A through D) into a vector, i.e., h_4, which is used to predict the user's next interest.

This lack of discriminating different historical records for next interest prediction leads to two unfavorable consequences: 1) it weakens the signal of highly correlated items for sequential recommendation; and 2) overlooking such signal makes it difficult for us to understand and explain the sequential recommendations.

∗ This work was conducted when the first author was visiting at Georgia Institute of Technology.
† Corresponding author.

WSDM'18, February 5–9, 2018, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-5581-0/18/02…$15.00
https://doi.org/10.1145/3159652.3159668



(a) Sequential recommendation based on RNN (b) General idea to adopt memory for recommendation (c) The general framework to implement our idea

Figure 1: (a) An example of leveraging user historical records for recommendation. The user bought A, B, C and D in the past orderly, these previous records (as embeddings of A, B, C and D) are summarized into h_4 for predicting E. (b) Our general idea of introducing a per user memory matrix to store user historical records. (c) A general framework to implement our idea. (best view in color)

To alleviate the problems, we view user behaviors as a decision making program described as neural turing machines [5], and propose to model user historical records with external memories. With the ability to express, store, and manipulate the records explicitly, dynamically, and effectively, external memory networks (EMN) [25, 28] have shown their promising performance for many sequential prediction tasks, such as question answering (QA) [18], natural language transduction [7], and knowledge tracking (KT) [32]. Instead of merging previous states to make predictions, EMN architecture introduces a memory matrix to store the states separately in memory slots, and then by designing proper operations on this matrix, it achieves significant improvement compared with conventional RNN/LSTM models in many tasks [28].

Inspired by EMN, we propose a novel framework integrating **R**ecommender system with external **U**ser **M**emory networks [5, 25, 28] (**RUM** for short), and further study its intuition and performance on Top-N recommendation tasks. Figure 1(b) and 1(c) illustrate the basic ideas of our proposed framework. For each user, an external user memory matrix is introduced to maintain her historical information, which enriches the representation capacity compared with traditional RNN hidden vectors. When making predictions, the memory matrix will be *attentively* read out to generate an embedding as the user representation, where the attention mechanism learns the different importance of previous records for the current recommendation. After processing each item in a sequence, the memory will be rewritten to update the user histories.

To better explore our idea, we provide two specifications of our framework, i.e., item-level and feature-level RUMs, which model user records on item- and feature-level, respectively. Compared with existing methods, our approach makes finer-grained use of user historical records based on memory networks, which improves the recommendation performance, meanwhile extracts intuitive causality of consumer behaviors based on attentive analyses.

Contributions. In summary, the main contributions of this work include:

• We propose to integrate the insights of collaborative filtering with memory-augmented neural networks for recommendation, which leverages user historical records in a more effective manner. To the best of our knowledge, this is the first attempt to introduce memory-augmented neural networks (MANNs) [5, 28] into the field of recommender systems.

• We investigate two potential memory networks (on item- and feature-level) with different representation and operation designs. We further study and compare their performance for sequential and top-N recommendation tasks.

• We also compare our model with state-of-the-art methods and verify the superiority of our model through quantitative analyses on real-world datasets, which shows that our method is able to leverage user historical records more effectively.

• We further provide empirical analyses to explain how and why an item is recommended by our model, which shows that with the attention mechanism in memory networks, our model is capable of providing intuitive explanations about how a user's historical records affect her current and future decisions.

In the following part of the paper, we first introduce the related work in section 2, and then illustrate our framework in section 3 and 4. In section 5, we verify the effectiveness of our method with experimental results, and the conclusions and outlooks of this work are presented in section 6.

2 RELATED WORK

Our work is essentially an integration of sequential recommendation and memory-augmented neural networks. In the following, we review the related work on these two research directions.

2.1 Sequential Recommendation

In the literature, many models have been proposed to leverage user historical records in a sequential manner for future behavior prediction and recommendation.

By integrating matrix factorization and Markov chains, factorized personalized Markov chains (FPMC) [23] embeds the transition information between adjacent behaviors into the item latent factors for recommendation, and the hierarchical representation model (HRM) [27] further extends the idea by leveraging representation learning as latent factors. These methods mostly model the local sequential patterns between every two adjacent records [31].

To model multiple-step sequential behaviors, [10] adopted Markov chains to provide recommendations with sparse sequences, and [31] proposed the dynamic recurrent basket model (DREAM) to capture global sequential patterns and to learn dynamic user interest representations based on recurrent neural network (RNN). In DREAM, all of a user's historical records are embedded into the final hidden state of RNN to represent her current preference, and this method has achieved significant improvement against HRM and FPMC. Similarly, [14, 26] leveraged user previous clicking and purchasing behaviors to model short-term preferences with RNN for session-based recommendation, and [9] adopted a metric space learning approach to learn additive user-item relations for sequential recommendation. Beyond e-commerce, sequential recommendation has also been applied to various application scenarios such as POI recommendation [3, 4], music recommendation [1, 8, 29], browsing recommendation [35], etc.

Existing models usually implicitly encode user's previous records into a latent factor or hidden state without distinguishing the different role that each record may play when predicting the current interest. In this work, however, we leverage user memory networks to store and manipulate each user's previous records, which helps to enhance the expressive power of user histories.

2.2 Memory-Augmented Neural Networks

With the power to process sequential data effectively, external memory network (EMN) [5, 6, 28] has been emerging in recent years. In a nutshell, it leverages a memory matrix to store historical hidden states, and by properly reading and updating this matrix, it can obtain improved performance on many sequence-oriented tasks. Following this idea, [25] designed an end-to-end memory-augmented model, which requires significantly less supervision during training, and makes it better applicable in real-world settings. Very recently, researchers have successfully adapted the idea of external memory network into several application domains, such as question answering (QA) [18], natural language transduction (NLT) [7], and knowledge tracking (KT) [32].

Typically, EMN consists of two major components: a memory matrix that maintains the states, and a controller that operates (including reading and writing) the matrix. More specifically, most methods [5, 6, 25, 28] adopt the attention mechanism to read the memory matrix, i.e., for an input $\boldsymbol{q}$, they first compute the similarity $S(\boldsymbol{q}, \boldsymbol{m}_i)$ between the input and each memory slot $\boldsymbol{m}_i$ in the memory matrix, and then the attention weights are derived by $w_i = SOFTMAX(S(\boldsymbol{q}, \boldsymbol{m}_i))$, where $SOFTMAX(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$, based on which the memory is attentively read out. For the writing process, [5] updated user memory by considering both content and slot location to facilitate all the locations of the memory, while [24] proposed a purely content-based writing strategy named least recently used access (LRUA) to write memories to either the least used memory location or the most recently used memory location.

In this paper, we aim to apply the idea of EMN to recommender system for leveraging user historical behaviors more effectively, which is yet to be explored in the research community.

3 RUM: RECOMMENDATION WITH USER MEMORY NETWORKS

In this section, we first present our general framework, elaborating how to integrate user memory networks with collaborative filtering. Then, we focus on designing the memory component by describing two specific implementations of the general framework, namely, the item- and feature-level user memory networks, so as to inspect our framework from different perspectives.

3.1 General Framework

For better understanding, we first re-describe the widely used matrix factorization (MF) model as a neural network. Suppose there are N users and M items, and let $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ be the embeddings of user u and item i, then the likeness score of u to i can be predicted as $\hat{y}_{ui} = \boldsymbol{p}_u^T \boldsymbol{q}_i$ in MF. In the context of neural network, the user/item IDs with one-hot format can be used as inputs fed into the architecture, then the look-up layer projects these sparse representations into dense vectors, which correspond to the user/item embeddings in MF models [13]. At last, the likeness score $\hat{y}_{ui}$ is computed as the vector inner product between $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$.

Memory enhanced user embedding. To leverage user historical records in our framework, we generate a user's embedding from two parts (see Figure 1(c)): one is related to the user's memory component that encodes her previous records (named as *memory embedding* in the figure), and the other is a free vector used to encode her intrinsic preference that is not influenced by her previous actions (named as *intrinsic embedding*). The memory embedding is similar to the hidden vector in conventional RNN-based models. However, hidden vectors blindly compress all of a user's historical records into a fixed vector, while in our framework, user records are encoded, stored, and carefully updated through a more expressive structure – the personalized memory matrix $\boldsymbol{M}^u$.

Specifically, for user u, her memory embedding $\boldsymbol{p}_u^m$ is derived by reading $\boldsymbol{M}^u$ according to the current item embedding $\boldsymbol{q}_i$:

$$\boldsymbol{p}_u^m = READ(\boldsymbol{M}^u, \boldsymbol{q}_i) \tag{1}$$

And then, by merging $\boldsymbol{p}_u^m$ with the user intrinsic embedding $\boldsymbol{p}_u^*$, we get the final user embedding as:

$$\boldsymbol{p}_u = MERGE(\boldsymbol{p}_u^*, \boldsymbol{p}_u^m) \tag{2}$$

where $MERGE(\cdot)$ is a function that merges two vectors into one. The particular choice of $MERGE(\cdot)$ in our model is a simple weighted vector addition, that is:

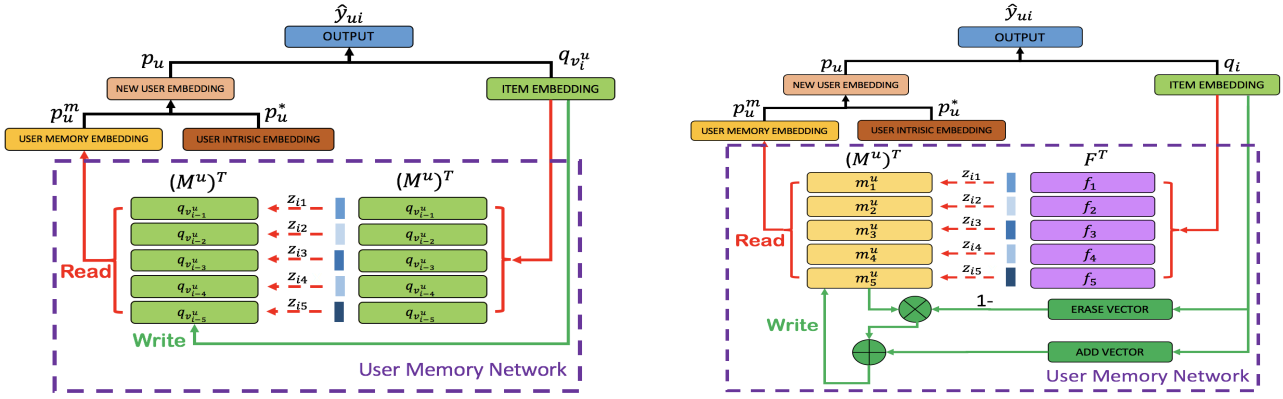
$$MERGE(x, y) \doteq x + \alpha y = \boldsymbol{p}_u^* + \alpha \boldsymbol{p}_u^m \tag{3}$$

where α is a weighting parameter. We have also tested element-wise multiplication and concat operations, but they led to unfavorable performance. Another advantage of our choice is that we can tune the weighting parameter α to study the effect of incorporating memory mechanism for recommendation, which will be shown in the experiments.

Prediction function. When making predictions, we feed the final user embedding $\boldsymbol{p}_u$ and the item embedding $\boldsymbol{q}_i$ into a function:

$$\hat{y}_{ui} = PREDICT(\boldsymbol{p}_u, \boldsymbol{q}_i) \tag{4}$$

where $PREDICT(\cdot)$ is an arbitrary prediction function, or even a prediction neural network as in [13]. Here, we choose the sigmoid



(a) Item-level RUM

(b) Feature-level RUM

Figure 2: Two specific implementations of our RUM framework. To better illustrate the model overflow, we plot $(M^u)^T$ and F^T instead of M^u and F in the figure, so that each row corresponds to a memory slot in the figure. (best view in color)

inner product $\hat{y}_{ui} = \sigma({\boldsymbol{p}_u}^T \cdot \boldsymbol{q}_i)$ as a specific implementation, because it gives us better training efficiency for our data. However, it is not necessarily restricted to this function and many others can be used in practice according to the specific application domain.

At last, we adapt binary cross-entropy as our loss function for model optimization, and the objective function to be maximized is:

$$\begin{aligned} l_{RUM} &= \log \prod_{(u,i)} (\hat{y}_{ui})^{y_{ui}} (1-\hat{y}_{ui})^{1-y_{ui}} - \lambda ||\Theta||_F^2 \\ &= \sum_u \sum_{i \in I_u^+} \log \hat{y}_{ui} + \sum_u \sum_{i \in I/I_u^+} \log(1-\hat{y}_{ui}) - \lambda ||\Theta||_F^2 \end{aligned} \tag{5}$$

where Θ is the model parameter set, y_{ui} is the ground truth that would be 1 if u has purchased i, and 0 otherwise. I is the set of all items, and I_u^+ is the set of items that u purchased arranged in purchasing order, namely, $I_u^+ = \{v_1^u, v_2^u, \cdots, v_{|I_u^+|}^u\}$, where v_j^u is the j-th item purchased by u. We uniformly sample the negative instances from unobserved item set $I_u^- = I/I_u^+$, and it should be noted that a nonuniform sampling strategy might further improve the performance, and we leave the exploration as a future work.

In this equation, we maximize the likelihood of our predicted results by the first two terms, and regularize all the model parameters to avoid over fitting by the last term. In the training phase, we learn the parameters via stochastic gradient descent (SGD).

Memory updating. After each purchasing behavior, we update the user memory matrix $\boldsymbol{M}^u$ to maintain its dynamic nature by:

$$\boldsymbol{M}^u \leftarrow WRITE(\boldsymbol{M}^u, \boldsymbol{q}_i) \tag{6}$$

In the following, we will describe how the personalized memory matrix $\boldsymbol{M}^u$ encodes user behaviors, and how to design the $READ(\cdot)$ and $WRITE(\cdot)$ operations on both item- and feature-levels.

3.2 Item-level RUM

In this section, we first implement our idea by extending previous methods in a straightforward manner (see Figure 2(a)). Similar to existing models [14, 23, 26, 27, 31], we regard each item as a unit, and model the impact of previously purchased items on the following ones. Many works [23, 27] have shown that users' recent behaviors can be more important to the current decisions. As a result, for each user u we make the memory matrix $\boldsymbol{M}^u$ store the embeddings of the u's recently purchased items, as denoted in the purple dashed box in Figure 2(a).

Suppose the set of items purchased by user u is defined as $I_u^+ = \{v_1^u, v_2^u, \cdots, v_{|I_u^+|}^u\}$ (arranged in purchasing order), where v_i^u is the i-th item purchased by u, let $\boldsymbol{p}_u \in R^D$ and $\boldsymbol{q}_{v_i^u} \in R^D$ be the embeddings of user u and item v_i^u, and suppose there are K columns (i.e., memory slots) in the memory matrix, namely, $\boldsymbol{M}^u \in R^{D \times K} = \{\boldsymbol{m}_1^u, \boldsymbol{m}_2^u, \cdots, \boldsymbol{m}_K^u\}$, where $\boldsymbol{m}_k^u \in R^D$ is the k-th column vector of $\boldsymbol{M}^u$. We design the reading and writing operations as follows.

Reading operation. Intuitively, the previous products may have different impacts on the current item, and the more influential product should be valued more in the final memory embedding. Formally, in our model, when making prediction on a user-item pair (u, v_i^u), we first adopt a similar way as FPMC [23] to compute the impacts between the items in user memory $\boldsymbol{M}^u$ and the current item by:

$$w_{ik} = (\boldsymbol{q}_{v_i^u})^T \cdot \boldsymbol{m}_k^u, \; z_{ik} = \frac{\exp(\beta w_{ik})}{\sum_j \exp(\beta w_{ij})}, \; \forall k = 1, 2, \cdots, K \tag{7}$$

where β is the strength parameter. Then, we use z_{ik}'s as the attention weights to derive u's memory embedding, which provides us with the ability to access user historical behaviors according to their influences on the current item:

$$\boldsymbol{p}_u^m = \sum_{k=1}^{K} z_{ik} \cdot \boldsymbol{m}_k^u \tag{8}$$

Different from previous models, we do not forcefully merge all the item embeddings in the reading process. On the contrary, we first store them in $\boldsymbol{M}^u$ individually, and then attentively pay more attention to some of the items, which provides us a more finer-grained method to utilize user historical records.

Writing operation. As mentioned above, users' recent behaviors usually play more important roles for current predictions. As a result, we adopt a simple first-in-first-out mechanism to maintain the latest interacted items in the user memory matrix $\boldsymbol{M}^u$. Specifically, the memory matrix $\boldsymbol{M}^u$ for user u always stores the most recent K items. Suppose the current item is $\boldsymbol{q}_{v_i^u}$, then the memory

matrix is $\boldsymbol{M}^u = \{\boldsymbol{q}_{v^u_{i-1}}, \boldsymbol{q}_{v^u_{i-2}}, ..., \boldsymbol{q}_{v^u_{i-K}}\}$. When writing the memory, the earliest item would be replaced, and $\boldsymbol{M}^u$ is updated to $\{\boldsymbol{q}_{v^u_i}, \boldsymbol{q}_{v^u_{i-1}}, ..., \boldsymbol{q}_{v^u_{i-K+1}}\}$. Note that when the memory is not full, the item is directly added without replacing any other entry.

3.3 Feature-level RUM

Inspired by the insights of classical latent factor models (LFM) [17] for recommendation, we further explore to implement the RUM framework on the feature-level. In LFM, we assume that users may consider a set of product features when making purchasing decisions, and each embedding dimension in LFM represents a latent feature in the product domain, where the features span a latent representation space. LFM then estimates a user's preference on these features as a vector in this space.

In this work, we explicitly model such latent features with the power of memory networks. Intuitively, a user's preference on these features should be dynamically reflected by his/her purchasing behaviors. For example, if a user has an excellent experience on a newly purchased iPhone, then she may continue to select Apple products on the 'brand' feature in the future.

Inspired by these intuitions, we maintain the user preference on different features in the memory matrix, which would be read to generate user memory embeddings, and be written by each item she purchased. More specifically, we formulate our method into a key-value memory neural network [21], as shown in Figure 2(b). We first design a global latent feature table (GLFT) to store the feature embeddings, and when making predictions, an item will interact with this table to identify its related features. For each user, we leverage the user memory matrix $\boldsymbol{M}^u$ to encode her likeness on the features in GLFT. Based on the above identified features, the target user will attentively merge the columns in $\boldsymbol{M}^u$ to obtain her memory embedding. Same as the global feature space in LFM, the global latent feature table here is shared across all the users, while the memory matrix is maintained per-user level in a personalized manner. At last, we update the user memory matrix $\boldsymbol{M}^u$ using the item embeddings.

Formally, we let $\boldsymbol{p}_u \in R^D$ and $\boldsymbol{q}_i \in R^D$ be the embeddings of user u and item i. Suppose there are K latent features in our system, and the global latent feature table is $\boldsymbol{F} = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_K\}$, where $\boldsymbol{f}_k \in R^D$ is the embedding of feature k. For a user u, we define her memory matrix as $\boldsymbol{M}^u = \{\boldsymbol{m}^u_1, \boldsymbol{m}^u_2, \cdots, \boldsymbol{m}^u_K\}$, where $\boldsymbol{m}^u_k \in R^D$ is the embedding of u's preference on the feature k.

Reading operation. To make the reading process differentiable, we adapt soft-attention mechanism to read the user memory matrix. Specifically, when making prediction for the user-item pair (u, i), we first compute i's relativeness with each feature in the global latent feature table by:

$$w_{ik} = \boldsymbol{q}_i^T \cdot \boldsymbol{f}_k,\ z_{ik} = \frac{\exp(\beta w_{ik})}{\sum_j \exp(\beta w_{ij})},\ \forall k = 1, 2, \cdots, K \quad (9)$$

where β is still the strength parameter, and we also linearly merge the slots in user u's memory matrix using the derived z_{ik} attentions to compute her memory embedding:

$$\boldsymbol{p}^m_u = \sum_{k=1}^{K} z_{ik} \cdot \boldsymbol{m}^u_k \quad (10)$$

Writing operation. Inspired by neural turing machine (NTM) [5], when writing the user memory matrix $\boldsymbol{M}^u$, it will be erased first before new information is added.

Specifically, we first derive a D dimensional erase vector $\boldsymbol{erase}_i \in R^D$ from q_i by:

$$\boldsymbol{erase_i} = \sigma(\boldsymbol{E}^T \boldsymbol{q}_i + \boldsymbol{b}_e) \quad (11)$$

where $\sigma(\cdot)$ is the element-wise sigmoid function, and $\boldsymbol{E}$ and $\boldsymbol{b}$ are the erase parameters to be learned. Given the attention weights and the erase vector, the feature preference memory is updated by:

$$\boldsymbol{m}^u_k \leftarrow \boldsymbol{m}^u_k \odot (\mathbf{1} - z_{ik} \cdot \boldsymbol{erase}_i) \quad (12)$$

where $\odot$ is element-wise product, $\mathbf{1}$ is a column-vector of all 1's. Therefore, the elements of a memory location are reset to zero only if both the weight at the location and the erase element are one. The memory vector is left unchanged if either the weight or the erase signal is zero [32].

After erasing, an add vector $\boldsymbol{add}_i \in R^D$ is used to update the feature preference memory by:

$$\boldsymbol{add}_i = \tanh(\boldsymbol{A}^T \boldsymbol{q}_i + \boldsymbol{b}_a),\ \boldsymbol{m}^u_k \leftarrow \boldsymbol{m}^u_k + z_{ik} \cdot \boldsymbol{add}_i \quad (13)$$

where $\boldsymbol{A}$ and $\boldsymbol{b}_a$ are the add parameters to be learned. This erase-add update strategy allows forgetting and strengthening user feature preference embeddings in the learning process, and the model can determine which signals to be weakened and which to be strengthened by learning the erase and add parameters automatically.

4 DISCUSSIONS AND FURTHER ANALYSIS

To provide more insights of our proposed model, we analyze the relationship between the item- and feature-level RUMs, and then further relate our method with previous ones by comparing it with matrix factorization (MF) and factorized personalized Markov chains (FPMC).

4.1 Item- v.s. Feature-level RUM

Generally speaking, both item- and feature-level RUMs are specific implementations of the same framework shown in Figure 1(c). However, they manage user historical information from different perspectives. The item-level RUM regards each item as a unit, and directly store the item embeddings in the memory matrix, which is designed to capture item-to-item transition patterns. While in the feature-level RUM, the historical information is leveraged in a feature-centered manner. The memory matrix is used to store the embeddings of user preferences on different latent features, and each item is indirectly utilized to change these embeddings.

When using these models in real-world applications, there is actually an "explanation-effectiveness" trade-off: item-level RUM can explicitly tell us which items in the past are more important for the current decision, which provides the system with certain explanatory ability. However, feature-level RUM can obtain better performance with finer-grained modeling in a "black box". We will discuss more details with experimental results in the following.

4.2 Relationship between RUM and MF

As shown in Figure 1(c), RUM will reduce to traditional MF when the user memory network is set to be unavailable, i.e., when the user memory embedding is set as an all-0 vector.

However, by enabling the user memory network, RUM can collect valuable information from the historical behaviors, which can bring improved performance in the task of Top-N recommendation as shown in our following experiments.

4.3 Relationship between RUM and FPMC

Both RUM and FPMC leverage user historical behaviors to predict the current ones. To model item-to-item transition patterns for each user, FPMC builds a tensor factorization model, and optimizes it under the Bayesian personalized ranking (BPR) criterion. The objective function is as follows [23, 27]:

$$l_{FPMC} = \sum_{u} \sum_{T_t^u} \sum_{i \in T_t^u} \sum_{i' \notin T_t^u} \log \sigma(\hat{x}_{u,t,i} - \hat{x}_{u,t,i'}) - \lambda ||\Theta||_F^2 \quad (14)$$

where T_t^u is u's t-th basket, Θ is the model parameter set, and $\hat{x}_{u,t,i}$ is predicted by:

$$\hat{x}_{u,t,i} = \boldsymbol{p}_u^T \cdot \boldsymbol{q}_i + \frac{1}{|T_{t-1}^u|} \sum_{l \in T_{t-1}^u} \boldsymbol{q}_l^T \cdot \boldsymbol{q}_i \quad (15)$$

When applied to the task of sequential recommendation, each basket in FPMC contains only one item, thus,

$$l_{FPMC} = \sum_{u} \sum_{i \in I_u^+} \sum_{i' \in I / I_u^+} \log \sigma(\hat{x}_{u,l,i} - \hat{x}_{u,l,i'}) - \lambda ||\Theta||_F^2 \quad (16)$$

and,

$$\hat{x}_{u,l,i} = \boldsymbol{p}_u^T \cdot \boldsymbol{q}_i + \boldsymbol{q}_l^T \cdot \boldsymbol{q}_i \quad (17)$$

where I_u^+ is the set of u's purchased items (arranged in purchasing orders), namely, $I_u^+ = \{v_1^u, v_2^u, \cdots, v_{|I_u^+|}^u\}$, and v_j^u is the j-th item purchased by u. I is the set of all items, and l is the item purchased just before i.

To show how our model degenerates to FPMC, we use only one memory slot in the item-level RUM (i.e., $K = 1$), and set the weighting parameter in $MERGE(\cdot)$ as 1 (i.e., $\alpha = 1$), then we have:

$$\begin{aligned} \hat{y}_{u,i} &= \sigma(\boldsymbol{p}_u^T \cdot \boldsymbol{q}_i) = \sigma(MERGE(\boldsymbol{p}_u^*, \boldsymbol{p}_u^m)^T \cdot \boldsymbol{q}_i) \\ &= \sigma((\boldsymbol{p}_u^* + \boldsymbol{p}_u^m)^T \cdot \boldsymbol{q}_i) = \sigma\Big((\boldsymbol{p}_u^* + \sum_{k=1}^{K} z_{ik} \cdot \boldsymbol{m}_k^u)^T \cdot \boldsymbol{q}_i\Big) \\ &\overset{\gamma_1}{=} \sigma((\boldsymbol{p}_u^* + \boldsymbol{m}_1^u)^T \cdot \boldsymbol{q}_i) \overset{\gamma_2}{=} \sigma((\boldsymbol{p}_u^* + \boldsymbol{q}_l)^T \cdot \boldsymbol{q}_i) \\ &= \sigma(\boldsymbol{p}_u^{*T} \cdot \boldsymbol{q}_i + \boldsymbol{q}_l^T \cdot \boldsymbol{q}_i) \end{aligned} \quad (18)$$

where step γ_1 holds because $z_{ik} = z_{i1} = 1$ when $K = 1$, and step γ_2 holds because $\boldsymbol{m}_1^u$ is exactly the embedding of the previously purchased item $\boldsymbol{q}_l$ according to our settings in item-level RUM.

By regarding the user intrinsic embedding $\boldsymbol{p}_u^*$ of RUM (in Eq.(18)) as the user embedding $\boldsymbol{p}_u$ of FPMC (in Eq.(17)), we have $\hat{y}_{u,i} = \sigma(\hat{x}_{u,l,i})$, and we can thus rewrite Eq.(5) as:

$$l_{RUM} = \sum_{u} \sum_{i \in I_u^+} \log \sigma(\hat{x}_{u,l,i}) + \sum_{u} \sum_{i \in I / I_u^+} \log(1 - \sigma(\hat{x}_{u,l,i'})) - \lambda ||\Theta||_F^2 \quad (19)$$

Comparing Eq.(16) and (19), we see that FPMC shares the same prediction function (Eq.(17)) with 1-order item-level RUM. However, their optimization methods are slightly different. For a triplet (u, i, i'), where (u, i) is an observed interaction and (u, i') is unobserved, FPMC learns the model by maximizing the margin of u's likeness between i and i', while RUM tries to maximize u's likeness on i and minimize u's likeness on i', respectively. Actually, we can also use Bayesian personalized ranking (BPR) to optimize RUM, in that way, FPMC equivalents a 1-order item-level RUM for sequential recommendation.

Based on the above analyses, we can see that RUM is a very general recommendation framework. On one hand, RUM is a generalization of many existing recommendation models, and on the other hand, RUM can provide us with the opportunity to explore other promising models by adapting the merge function, predict function, and the reading/writing strategy to other choices.

5 EXPERIMENTS

In this section we evaluate our proposed models. We begin by introducing the experimental setup, and then report and analyze the experimental results.

5.1 Experimental Setup

5.1.1 ***Datasets.*** We conduct our experiments on the Amazon dataset [1] [11, 20]. This dataset contains user-product purchasing behaviors from Amazon spanning May 1996 - July 2014. We evaluate our models on four product categories, including Instant Video, Musical Instrument, Automotive and Baby Care. To provide sequential recommendations, we select those users with at least 10 purchasing records for experiments, and the statistics of the final datasets are shown in Table 1.

Table 1: Statistics of our datasets.

Datasets	#Users	#Items	#Ratings	Density
Instant Video	1353	7786	33726	0.32%
Musical Instrument	2129	20928	55382	0.12%
Automotive	1144	24590	48706	0.17%
Baby Care	2379	14037	77198	0.23%

5.1.2 ***Evaluation methods.*** For each model, we define the generated recommendation list for user u as $R_u = \{r_u^1, r_u^2, \cdots, r_u^N\}$, where N is the number of recommended items, r_u^i is ranked at the i-th position in R_u according to the predicted score. Suppose the set of $u's$ interacted items in the test data is T_u, and there are totally M users in our system, we thus use the following measures for evaluation:

• **Precision (P), Recall (R) and F_1-score:** we adopt per-user average instead of global average for better interpretability [16]:

$$\begin{aligned} P@N &= \frac{1}{M} \sum_u P_u@N = \frac{1}{M} \sum_u \frac{|R_u \cap T_u|}{|R_u|} \\ R@N &= \frac{1}{M} \sum_u R_u@N = \frac{1}{M} \sum_u \frac{|R_u \cap T_u|}{|T_u|} \\ F_1@N &= \frac{1}{M} \sum_u F_{1u}@N = \frac{1}{M} \sum_u \frac{2 \cdot P_u@N \cdot R_u@N}{P_u@N + R_u@N} \end{aligned} \quad (20)$$

• **Hit-ratio (HR):** Hit-ratio gives the percentage of users that can receive at least one correct recommendation, which has been

[1] http://jmcauley.ucsd.edu/data/amazon/

widely used in previous work [16, 30]:

$$HR@N = \frac{1}{M}\sum_{u} I(|R_u \cap T_u|) \tag{21}$$

where $I(x)$ is an indicator function whose value is 1 when $x > 0$, and 0 otherwise.

• **NDCG:** The normalized discounted cumulative gain, which evaluates ranking performance by taking the positions of correct items into consideration [15]:

$$NDCG@N = \frac{1}{Z}DCG@N = \frac{1}{Z}\sum_{j=1}^{N}\frac{2^{I(|\{r_u^j\}\cap T_u|)}-1}{\log_2(j+1)} \tag{22}$$

where $I(x)$ is the indicator function as above, and Z is a normalization constant, which is the maximum possible value of $DCG@N$.

5.1.3 ***Baselines.*** We adopt the following representative and state-of-the-art methods as baselines for performance comparison:

• **MostPopular (MP):** This is a non-personalized static method, where the most frequently purchased items are ranked in descending order of frequency to make recommendations.

• **BPR:** Bayesian personalized ranking, which is a popular method for top-N recommendation [22]. We adopt matrix factorization as the prediction component for BPR.

• **FPMC:** Factorized personalized Markov chains, which is one of the stat-of-the-art models for sequential recommendation based on Markov chains [23]. Each purchased item in our data is regarded as a basket in this method.

• **DREAM:** The dynamic recurrent basket model, which is the stat-of-the-art sequential recommendation method based on recurrent neural networks [31].

5.1.4 ***Parameter settings.*** When implementing our method, the model parameters are first randomly initialized according to the uniform distribution, and then updated by conducting stochastic gradient descent (SGD). The learning rate of SGD is determined by grid search in the range of $\{1, 0.1, 0.01, 0.001, 0.0001\}$, and the number of memory slot K is empirically set as 20. We primarily set the weighting parameter $\alpha = 0.2$ in MERGE function, and the effect of using different α settings is also studied in the experiments.

For each purchased item (i.e., positive instance), we uniformly sample one negative instance from the uninteracted items of the user. The embedding dimension D and regularization parameters are determined by grid search in the range of $\{10, 20, 30, 40, 50\}$ and $\{0.1, 0.01, 0.001, 0.0001\}$, respectively.

In our experiments, each user's purchasing records are ordered in purchasing time, and the first 70% items of each user are used for training, while the remaining are used for testing. We recommend 5 items ($N = 5$) for each user.

5.2 Overall Performance of Our Models

We first study the performance of our item- and feature-level RUMs under default settings ($\alpha = 0.2$). Results are shown in Table 2, and we have the following observations.

The non-personalized Most Popular approach gives the most unfavorable performance in nearly all the cases. Since it does not consider user's personalized information, this observation highlights the importance of personalization in recommendation tasks.

As expected, by profiling users individually and optimizing the ranking-based objective function directly, BPR performs better than MP in most cases.

FPMC and DREAM can achieve better performance than BPR on most measures, while the difference between these two methods is not very significant. On considering that the key difference between BPR and FPMC is that the latter models user historical records in a sequential manner, this observation verifies that sequential recommendation can help to improve performance in real-world systems.

It is interesting to see that DREAM achieves better performance than FPMC on instant videos. DREAM models the multi-step behavior instead of pair-wise behavior of users, which has the ability to better capture users' long-term interests on video preference. However, on other datasets the pair-wise short-term influence between adjacent items may be more informative to predict the next behavior, but DREAM equally merges all the previous items into a single hidden vector through RNN, which may weaken the pair-wise signal for sequential recommendation. This highlights the importance of our motivation that a carefully designed mechanism is needed to automatically determine which previous item(s) are important for current interest prediction.

Encouragingly, we find that either item- or feature-level RUMs achieved better performance than the best baselines in most cases. These results indicates the effectiveness of our proposed methods for sequential recommendation, which is actually not surprising because the memory mechanism as well as the reading and writing designs in our model provide better expressive power to model user historical records.

By modeling item relations on a finer-grained level, the feature-level RUM outperforms item-level RUM on most measures. This is intuitive in that the function of two products may not be directly related, but they may share some features in common which may affect user decisions, for example, they are of the same brand.

5.3 Influence of the Weighting Parameter α

In this subsection, we are curious about whether and how the memory mechanism is helpful to sequential recommendation. To do so, we analyze the influence of the weighting parameter α, which determines the importance of the memory embedding against the intrinsic embedding in the MERGE function.

Specifically, we study the performance of our models on $F_1@5$ by tuning α in the range of $0 \sim 1$ with a step size of 0.1. Results are shown in Figure 3.

We see that when the memory component is unavailable (i.e., $\alpha = 0$), feature- and item-level RUMs reduce to the same model and share the same performance. In this case, both of them give unfavorable results across all the datasets. When memory network is incorporated, the performance increases drastically, and the best results are achieved when $\alpha \approx 0.2$. However, when the weight of memory embedding continues to rise, the performance goes down, and this observation is consistent on four datasets.

This result means that considering sequential influences from a user's recently purchased items indeed helps to provide better recommendations, but putting too much focus on this recent purchase signal may weaken a user's intrinsic preference. These results further verified the frequent observation in the research community

Table 2: Summary of the performance for baselines and our models. The first block shows the baseline performances, where starred numbers are the best baseline results; the second block shows the results of our item-level and feature-level RUM models. Bolded numbers are the best performance of each column, and all the numbers in the table are percentage numbers with '%' omitted.

Dataset	Instant Video					Musical Instrument					Automotive					Baby Care				
Measures@5(%)	P	R	F_1	HR	NDCG	P	R	F_1	HR	NDCG	P	R	F_1	HR	NDCG	P	R	F_1	HR	NDCG
MP	1.110	1.521	1.227	5.178	1.112	0.931	1.211	0.996	4.464	1.053*	0.805	0.481	0.589	3.937	0.880	0.858	0.719	0.753	4.163	0.923
BPR	1.198	1.624	1.300	5.917	1.243	0.987	1.261*	1.044*	4.699	0.930	0.822*	0.481	0.593	3.917	0.909*	0.958	0.771	0.819	4.668	1.016
FPMC	1.301	1.612	1.322	5.917	1.400	1.006*	1.170	1.014	4.793*	1.017	0.812	0.483*	0.594*	3.921*	0.873	1.009*	0.800*	0.853*	4.962*	1.026*
DREAM	1.312*	1.652*	1.342*	6.097*	1.401*	1.003	1.163	1.004	4.613	1.013	0.792	0.463	0.576	3.812	0.793	0.979	0.776	0.823	4.762	1.006
RUM (I)	1.302	1.846	1.449	6.213	1.290	1.005	1.243	1.047	4.652	**1.106**	0.823	0.485	0.591	4.022	0.921	1.051	0.878	0.918	5.088	1.025
RUM (F)	**1.405**	**1.905**	**1.491**	**6.287**	**1.448**	**1.032**	**1.308**	**1.094**	**4.734**	1.035	**0.842**	**0.501**	**0.622**	**4.111**	**0.963**	**1.070**	**0.918**	**0.951**	**5.432**	**1.103**

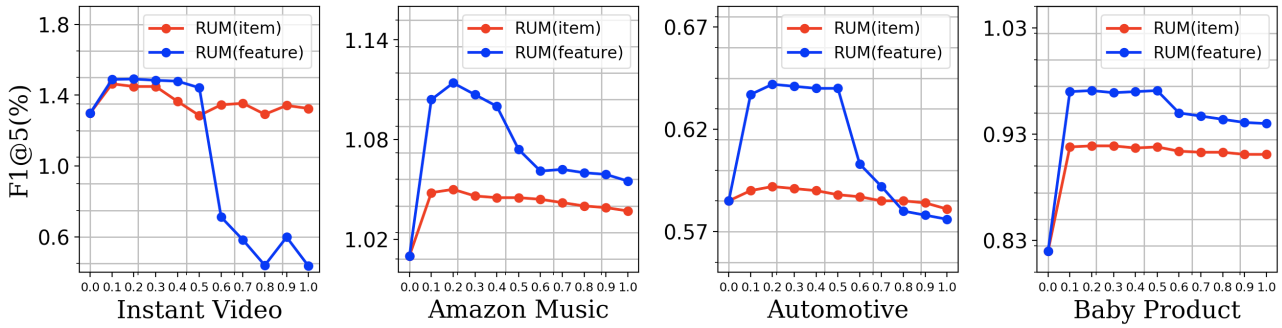


Figure 3: Performance of our item- and feature-level RUM models under difference choices of weighting parameter α.

that both short-term and long-term user preferences are important for personalized recommendation.

5.4 Intuition of Attention Weights in Item-level RUM

To illustrate intuition of item-to-item transitions, we present some example users in Figure 4, which are sampled from the results of item-level RUM with 5 memory slots ($K = 5$) on the Baby Care dataset. When a user purchased her i-th item (corresponding to the x-axis, denoted in a blue grid), we plot a length-5 column vector in her subfigure (e.g., the boxed vector in the upper-middle subfigure), and the vector element corresponding to position j on the y-axis is the attention weight z (in Eq.(7)) that this user casts on her j-th purchased item. When the user continues to purchase items, the most recently purchased 5 items are maintained in the memory, thus the column vectors shift from bottom-left to upper-right. The darker the color, the higher attention is casted on the item. Based on this figure, we have the following interesting observations.

Generally, the grids near the upper border of the diagonal strips are darker in color. This means that the most influential items in the past are usually near to the current behavior. This further confirms the assumption behind item-level RUM that recent behaviors are more important for current decisions. However, the specific position of the most influential item is not always the most recent one, which may explain why FPMC did not achieve favorable performance by only modeling the pair-wise adjacent behaviors.

Besides, we find two types of interesting user behavior patterns.

1) Some behavior sequences are continuously triggered by the most recent action (denoted in the green solid boxes), which is in line with the assumptions of FPMC [23]. We call this as the "one-to-one" behavior pattern, and a real-world example is that a user purchased some infant formula, and then bought a feeding bottle, which caused her to buy some nipples, and these nipples further made her buy a nipple cleaner.

2) Sometimes, a sequence of multiple behaviors are mostly influenced by the same previous action (denoted in the brown dotted boxes), while the relations between themselves are not so important. We call this the "one-to-multiple" pattern. A real-world example is that a mom bought a crib for her baby, after that she bought a water-proof mattress protector and a mosquito net for the crib, and further bought a bed bell to decorate the crib. In such cases, our RUM model with attention mechanism can automatically learn the importance of previous items based on large-scale user purchasing records, which is better than FPMC that assumes adjacent influence, and also better than RNN that merges all of the previous behaviors.

Based on these discovered patterns, the item-level RUM can interpret recommender system from the sequential behavior perspective, which is different from previous methods that usually leverage side information (e.g. user textual reviews [2, 12, 33, 34]) for explanations.

6 CONCLUSION

In this paper, we proposed to leverage external memory networks integrated with collaborative filtering for sequential recommendation. To do so, we designed the RUM sequential recommendation framework, and provided the item-level and feature-level specifications of the framework. Quantitative experimental analyses verified the effectiveness of our framework, and qualitative analyses verified the intuition behind our framework.

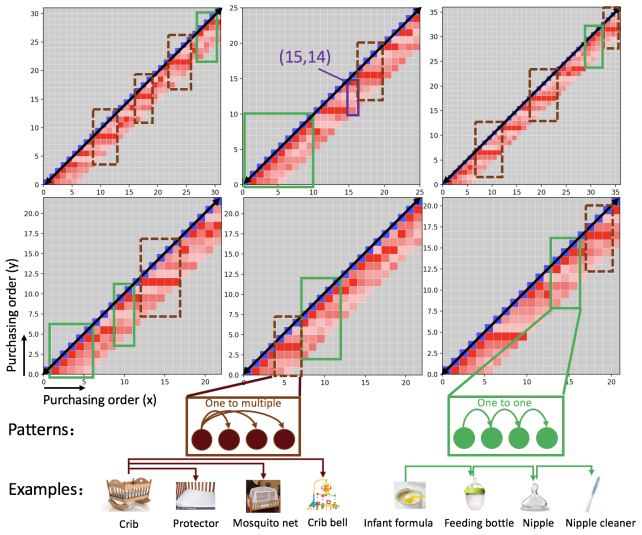


Figure 4: Illustration of item-to-item transitions. Each subplot is a sampled user, the x-axis and y-axis represent the user's purchased items orderly. The diagonal strip represents the current items. The color of an element $(i, j)(j < i)$ is proportional to the attention on previous item j when purchasing current i. For example, in the purple box of the second subplot, the darkest grid $(15, 14)$ means that the 14-th item is the most influential one among the most recent five for the 15-th purchase. Two types of user behavior patterns (one-to-one and one-to-multiple) as well as their examples are illustrated in the bottom.

This is a first step towards our goal for recommendation based on explicit user memory modeling, and there is much room for further improvements. By introducing side information such as user reviews and product images, we can align the memory units in feature-level RUM with different semantics, and thus we can build a more explainable recommender system. Besides, our RUM model is a framework with the ability of flexible generalizations, as a result, we can study other types of memory network designs to adapt our framework to different application scenarios.

ACKNOWLEDGMENT

This work was supported in part by NSF IIS-1639792, NSF IIS-1717916 and NSF CMMI-1745382. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *KDD*.
[2] Xu Chen, Zheng Qin, Yongfeng Zhang, and Tao Xu. 2016. Learning to rank features for recommendation over multiple categories. In *SIGIR*.
[3] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-Interest Recommendation.. In *IJCAI*.
[4] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation.. In *IJCAI*.
[5] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* (2014).
[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. (2016).
[7] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *NIPS*.
[8] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latent topic sequential patterns. In *RecSys*.
[9] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. *Recsys* (2017).
[10] Ruining He and Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *ICDM*.
[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, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. Trirank: Review-aware explainable recommendation by modeling aspects. In *CIKM*.
[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.
[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. *ICLR* (2016).
[15] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*.
[16] George Karypis. 2001. Evaluation of item-based top-n recommendation algorithms. In *CIKM*.
[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* (2009).
[18] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*.
[19] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-aware sequential recommendation. In *ICDM*.
[20] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*.
[21] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *EMNLP* (2016).
[22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
[23] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*.
[24] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *ICML*.
[25] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *NIPS*.
[26] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on DLRS*.
[27] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR*.
[28] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory Networks. *ICLR* (2015).
[29] Xiang Wu, Qi Liu, Enhong Chen, Liang He, Jingsong Lv, Can Cao, and Guoping Hu. 2013. Personalized next-song recommendation in online karaokes. In *Recsys*.
[30] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. 2010. Temporal recommendation on graphs via long-and short-term preference fusion. In *KDD*.
[31] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A dynamic recurrent model for next basket recommendation. In *SIGIR*.
[32] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. 2017. Dynamic Key-Value Memory Networks for Knowledge Tracing. In *WWW*.
[33] Yongfeng Zhang. 2015. Incorporating phrase-level sentiment analysis on textual reviews for personalized recommendation. In *WSDM*.
[34] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *SIGIR*.
[35] Yongfeng Zhang, Min Zhang, Yiqun Liu, Chua Tat-Seng, Yi Zhang, and Shaoping Ma. 2015. Task-based recommendation on a web-scale. In *BigData*.