Knowledge-Based Systems 213 (2021) 106706

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A deep reinforcement learning based long-term recommender system

Liwei Huang [a], Mingsheng Fu [a,b,*], Fan Li [a], Hong Qu [a], Yangjun Liu [c], Wenyu Chen [a]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
[b] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
[c] Commercial Machine Intelligence, Taoxi Technology Department, Alibaba Group, Hangzhou, China

ARTICLE INFO

Article history:
Received 27 May 2020
Received in revised form 16 December 2020
Accepted 17 December 2020
Available online 24 December 2020

Keywords:
Recommender system
Deep reinforcement learning
Long-term recommendation
Cold-start

ABSTRACT

Recommender systems aim to maximize the overall accuracy for long-term recommendations. However, most of the existing recommendation models adopt a static view, and ignore the fact that recommendation is a dynamic sequential decision-making process. As a result, they fail to adapt to new situations and suffer from the cold-start problem. Although sequential recommendation methods have been gaining attention recently, the objective of long-term recommendation still has not been explicitly addressed since these methods are developed for short-term prediction situations. To overcome these problems, we propose a novel top-N interactive recommender system based on deep reinforcement learning. In our model, the processes of recommendation are viewed as Markov decision processes (MDP), wherein the interactions between agent (recommender system) and environment (user) are simulated by the recurrent neural network (RNN). In addition, reinforcement learning is employed to optimize the proposed model for the purpose of maximizing long-term recommendation accuracy. Experimental results based on several benchmarks show that our model significantly outperforms previous top-N methods in terms of Hit-Rate and NDCG for the long-term recommendation, and can be applied to both cold-start and warm-start scenarios.

1. Introduction

Recommender systems (RS) play an increasingly important role in modern online services. In practice, the top-N recommendation is the most popular form in RS, where collaborative filtering (CF) is the dominant method due to its superior prediction performance.

In general, most of the existing CF-based methods, such as matrix factorization (MF), adopt a static view during the recommendation process [1], which ignore the dynamic and sequential nature of the recommendation problem. These static methods may fail to adapt to new situations, since the user and item states or representations are fixed in the processes of continuous recommendations. As a consequence, same recommendation results are repeatedly presented to a particular user regardless of whether the user is still interested. In fact, recommendation should be considered as a sequential decision process [2], and the adaptability is crucial to RS since personal preferences of users always evolve over time [3]. In addition, those static view methods rely entirely on pre-learned user representation obtained from the historical data of a user, which makes them suffer from the user cold-start problem [4].

In order to fuse the capability of sequential processing, the recurrent neural network (RNN) has been introduced into the RS community recently, which treats recommendation as a sequence prediction problem. Depending on the target, there are two different directions, (1) short-term prediction and (2) long-term prediction [5]. In short-term prediction, the system only needs to predict the result for the immediate next round recommendation. While in long-term prediction, the system should consider the long-term benefits of multiple rounds of recommendations in the future.

Generally, most of the existing sequential recommendation methods are proposed for short-term prediction with supervised learning (SL), and the label is the item exactly chosen by the user at the next step. Consequently, RS are enforced to generate a prediction sequence following a strict order to align with the ground-truth labels. However, in the long-term recommendation scenarios, the correct recommended item for a given step does not have to be the one that the user actually selected, and it can be any user's favorite items. Thus, there are multiple optional correct targets for a given step. Therefore, traditional SL cannot be effectively applied to train the long-term prediction system without clear and unique labels.

To overcome the training problem for long-term prediction, reinforcement learning (RL) [6] is an ideal proposal, since RL does not require an explicit target. Moreover, RL aims to maximize the long-term returns, and this is consistent with the goal of

* Corresponding author at: School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China.
E-mail address: fms@uestc.edu.cn (M. Fu).

https://doi.org/10.1016/j.knosys.2020.106706
0950-7051/© 2020 Elsevier B.V. All rights reserved.

RS, which is trying to achieve high accuracy in the long-term recommendation. However, traditional RL methods suffer from the curse of dimensionality problem when applied to complex systems like RS. Besides, the interaction between the recommendation agent and the user is an important part of RL, whereas it is completely ignored by the existing RNN-based sequential models.

In this paper, we propose a novel top-N model for long-term prediction based on deep reinforcement learning (DRL) to solve the problems mentioned above. To be specific, we employ RNN to adaptively evolve user state to simulate the sequential interaction between user and RS, and then output the top-N recommendation list each time. Thanks to the adaptive user state, our model can effectively deal with the user cold-start problem. Besides, our model can also be applied to warm-start scenarios. To leverage the historical items of user, an additional gated neural network is employed to balance the effects between the RNN adaptive user state and the historical user state. In order to maximize the expected long-term rewards and optimize the model parameters as well, we propose an efficient learning method based on the popular policy gradient algorithm REINFORCE [6]. In addition, an off-line interaction environment for training and testing is constructed. To evaluate the long-term recommendation performance, the popular metrics HR and NDCG are used. Specially, we adopt new evaluation settings for HR and NDCG, since their traditional setting cannot effectively handle long-term recommendation scenarios.

Several experiments are conducted on three real-world benchmarks, which are MovieLens 100K, MovieLens 1M, and Steam, and the results show that our model has promising performance for both cold-start and warm-start scenarios. In summary, the advantages of our model are: (1) the ability of long-term recommendation with high accuracy; (2) user state can be dynamically updated; (3) suitable for both cold-start and warm-start situations without additional content information.

The main contributions of this paper are as follows. (1) We introduce a novel end-to-end neural network-based CF model to address long-term interactive top-N recommendation, which is a context-free model and can solve the cold-start problem well. (2) To effectively leverage historical data for the warm-start scenario, we propose a new extended GRU cell named EMGRU, which can efficiently enhance the recommendation accuracy by incorporating additional historical information. (3) To effectively solve the learning problem of applying REINFORCE, we propose a special learning strategy, which can significantly improve the effectiveness of learning in the experiments. (4) We construct an off-line simulated environment and several metrics to evaluate the long-term recommendation performance, and numerous experiments are conducted to demonstrate the effectiveness of the proposed model.

2. Related works

CF-based methods [7–9] are widely used in RS. Among various CF-based methods, MF-based methods [10–12] are the most popular due to their effectiveness and simplicity. To capture more complex relations, neural network is also introduced to construct CF model [13–17]. However, most existing CF-based methods are static, and rely heavily on historical interaction or content information. As a result, they cannot effectively tackle the cold-start problem without historical information provided.

In most recommendation scenarios, the users' preferences might change over time, and the RS should take this into consideration. Incorporating the time into the recommendation algorithm can greatly enhance its performance [18], and many researches have been done to deal with the temporal information in RS [19–25]. However, the temporal recommender systems have also been affected by the problems of data sparsity and cold-start [26], since sparse matrix lowers the ability of utilizing quality feedback of neighboring users when predicting the suitable rating for the active user.

As one of the most important challenges in RS, the user cold-start problem refers to recommending items to new users whose previous references do not exist in the system. To tackle this issue, trust-aware RS [27–29] and social RS [30–32] have been introduced. In this paper, we propose to deal with the user cold-start problem in a different way by applying the RNN and reinforcement learning algorithm, which will be introduced in detail later.

Recently, RNN-based RS have received wide attention owing to their ability to address sequential data. Hidasi et al. [33] employed RNN in a sequential recommender system to predict the item that the user would click next time. Deriving from it, a large number of improved session-based RS using RNN have emerged [34–37]. However, they all focus on short-term prediction and leave out of the consideration of users' interactions.

Different from traditional RS, interactive recommender systems (IRS) consider users' feedback. More specifically, IRS iteratively construct or refine the representations of users or items with the received feedback from the beginning. As a result, this framework could naturally address the cold-start problem without providing users' experience beforehand. Multi-armed bandit (MAB) is the most popular method [38–41] in IRS. In MAB, each arm denotes a certain item, and upper confidence bound (UCB) based arm selection strategy is employed to choose the recommended items. Particularly, the contextual bandit methods rapidly become popular since they take content features of the item into account [38,42]. Although MAB is one of the most popular methods in RL, it only focuses on addressing the exploitation–exploration problem, and it does not explicitly optimize long-term returns.

DRL has achieved remarkable breakthroughs in many interactive systems, such as Atari games [43] and Go [44]. For DRL-based RS, Chen [45] proposed to use deep Q network (DQN), a value-based RL model, for tip recommendation, which provides keywords among the search results. Zheng [46] employed DQN to construct DRL-based IRS for news recommendation. Besides, in another DQN-based IRS introduced by Zhao et al. [47], two separated RNN are employed to capture sequential positive and negative feedback, respectively. However, the value-based model would quickly become intractable when the action space (number of items) is extremely large [48]. Zhao et al. [49] also proposed a DRL method based on Actor-Critic (AC) framework for the page-wise recommendation, where AC is a hybrid RL method composed by both value-based and policy-based modules. However, the page-wise recommendation is very different from the popular list-wise top-N recommendation, since it needs to consider the layout of items on one page. As a result, this method is more cumbersome than other methods. Compared with the mentioned methods above, our model focuses on long-term top-N recommendation, and the learning strategy as well as the model architecture are significantly different from them.

3. Overall framework

Generally, an RL-based system correlates to the interactions between environment and agent. During training, the parameters of the agent are optimized via the obtained rewards from the continuous interactions between the environment and agent. More specifically, this kind of interaction includes two successive steps: (1) the agent executes an action according to the environment; (2) the environment gives feedback to the agent for the performed action.

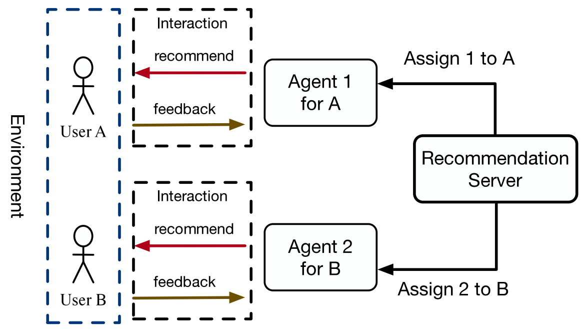

Fig. 1. Overall recommendation processes.

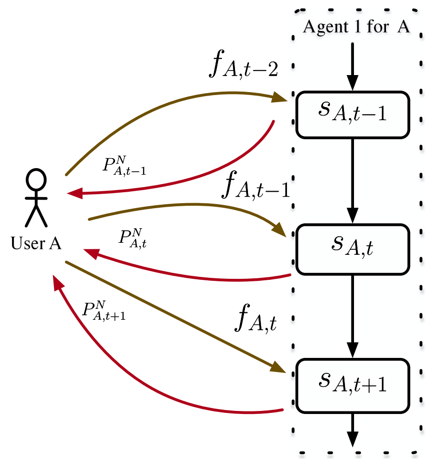

Fig. 2. The recommendation processes for an individual user. At each time t, the red line is the generated top-N recommendation $P^N_{u,t}$ for user u, and the brown line is the feedback $f_{u,t}$ of u for recommendation $P^N_{u,t}$.

In our recommendation scenario, the environment is composed of different users, the agent is our RNN-based CF model, the action is the top-N recommendation list for a particular user, and the feedback is whether the user accepts this recommendation list or not. The overall recommendation processes are exhibited in Fig. 1. As shown in the figure, for each individual user, there is a personal recommendation agent assigned to him/her. Especially, all assigned agents share the same model parameters (the model will be introduced in Section 5). This agent assignment can avoid mutual interference among different users.

Within an individual user, the internal state of RNN is continually updated according to the feedback. This process is illustrated in Fig. 2. For example, at time t, the agent state $s_{A,t}$ for user A is obtained according to the previous feedback $f_{A,t-1}$ and state $s_{A,t-1}$, and then the new recommendation list $P^N_{A,t}$ is generated via $s_{A,t}$. Furthermore, the agent receives the feedback $f_{A,t}$ as well as the corresponding reward for time t. For different users, the feedback sequences vary, which conduces to a relatively large discrepancy in the states for different users.

The main notations used in this paper are described in Table 1.

In general, our overall framework can be divided into 3 parts, (1) state transition $\mathbb{T}$, (2) recommendation list generation $\mathbb{G}$, and (3) user feedback $\mathbb{F}$.

Formally, the state transition process is denoted as follows,

$$s_{u,t} = \mathbb{T}(s_{u,t-1}, f_{u,t-1}, P^N_{u,t-1}), \tag{1}$$

where the new state $s_{u,t}$ is obtained from previous state $s_{u,t-1}$, feedback $f_{u,t-1}$, and recommendation list $P^N_{u,t-1}$.

Table 1
Notation descriptions.

Notation	Description
I	the set of all items (actions)
I_u	the subset of I, which includes items that user u likes
$s_{u,t}$	the tth state of RNN for user u
$P^N_{u,t}$	the tth recommendation list including N items for user u
$f_{u,t}$	the tth feedback responded by user u.
$V_{u,t}$	the tth immediate reward with respect to $f_{u,t}$.
$R_{u,t}$	the tth accumulative reward with respect to $f_{u,t}$.

The recommendation list $P^N_{u,t-1}$ is retrieved from I via Eq. (2).

$$P^N_{u,t} = \mathbb{G}(s_{u,t}, I). \tag{2}$$

The feedback of user u can be obtained from Eq. (3).

$$f_{u,t} = \mathbb{F}(P^N_{u,t}, I_u). \tag{3}$$

Specifically, $\mathbb{T}$ and $\mathbb{G}$ are related to the recommendation agent model, while $\mathbb{F}$ correlates with the environment. To clearly exhibit the details of our model, we will introduce the environment and the recommendation agent separately in the following sections.

4. Environment and interaction

To measure the performance of interactive recommender systems, the learned recommendation model should interact with actual users, and the feedback of users can be affected by various factors, e.g., the layout of recommendation results, the accuracy of predicting the preference of users, and user's perception and trust of the system [50]. Therefore, evaluating recommender systems with an on-line environment is the most appropriate way. However, the on-line environment is not always possible [3], and most current researches on recommender systems use off-line environments to evaluate the proposed models' performance. As a result, the overall environment in this paper is constructed from off-line datasets, and the interaction between the user and the system is simplified to only consider whether the recommended result can hit the user's known interactions in the off-line datasets.

Generally, the off-line dataset is composed by a user–item ratings matrix $\tilde{R} \in \mathbb{R}^{U*M}$ with U users and M items, and the element $\tilde{R}_{u,i}$ in $\tilde{R}$ is the rating with respect to user u and item i. Following [51], explicit rating matrix $\tilde{R}$ is converted into implicit feedback matrix F according to Eq. (4), where the element $F_{u,i}$ indicates whether user u is engaged with item i.

$$F_{u,i} = \begin{cases} 1 & \tilde{R}_{u,i} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

According to the implicit feedback matrix F, I_u can be obtained easily. The items are sorted by timestamp and the value of $F_{u,i}$ for each item $i \in I_u$ must equal 1.

In our environment, the recommendation agent should interact with users one by one. Suppose that user u responds a positive or negative feedback to the recommendation list $P^N_{u,t}$, then the feedback $f_{u,t}$ obtained from the user feedback function $\mathbb{F}(P^N_{u,t}, I_u)$ can be defined as Eq. (5).

$$f_{u,t} = \mathbb{F}(P^N_{u,t}, I_u) = \begin{cases} 1 & P^N_{u,t} \cap I_u \neq \varnothing, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where 1 refers to positive feedback and 0 refers to negative feedback. $P^N_{u,t} \cap I_u \neq \varnothing$ indicates $P^N_{u,t}$ successfully hits at least one item that the user likes. In practice, positive feedback could refer to clicking or purchase. Likewise, negative feedback could refer to ignoring or clicking other items out of the list. When the user performs one of the above actions, the environment automatically responds feedback.

To simulate the interaction between recommendation agent and user in the environment, agents should comply with the following assumptions:

- (1) Each user u has a total of $|I_u|$ rounds of interactions with his/her personal recommendation agent.
- (2) u can select at most one item in $P^N_{u,t}$ at each time t.
- (3) At each time t, if $P^N_{u,t} \cap I_u \neq \varnothing$, user u should select only one item $\hat{a}_{u,t} \in P^N_{u,t} \cap I_u$ with the highest estimated probability, and responds a positive feedback.
- (4) At each time t, if $P^N_{u,t} \cap I_u = \varnothing$, there is no hit item, and user u responds a negative feedback.
- (5) If $i \in P^N_{u,t} \cap I_u$ is selected, it should be removed from I_u to avoid being repeatedly chosen.

Assumption (1) restricts that only up to $|I_u|$ items that the user interested can be found in the offline environment. For example, if a user has 20 samples, the system only runs 20 steps. If the system is able to successfully hit items at all 20 steps, the overall hit rate is 100%. Both assumptions (2) and (5) are for the consistency with traditional models, such as neural CF [51] and session-based RNN [33], in which only up to one correct recommendation exists at time t. Besides, in a real scenario, the user may repeatedly select a certain item. However, most datasets only provide whether the user has selected a certain item. Whether the user will repeat the selection and how many times it is selected are unknown. Hence, similar to most researches, we simplified the interaction to allow only one selection. Assumptions (3) and (4) are to simulate and simplify the behaviors in practice by assuming that a user tends to choose top items in the recommendation list. Although the user may choose one item he likes with a relatively lower score in the recommendation list, we just choose the top one to make the simulated recommendation processes more stable. The complete interactive processes for user u are exhibited in Fig. 3.

In this paper, two kinds of recommendation situations are considered, which are the cold-start scenario and the warm-start scenario, respectively. In the cold-start scenario, there are no historical items provided to the test users. While in the warm-start scenario, whether in training or test, p% historical items in I_u are supplied to construct the users' history. What is more, we will consider the recommendation performance on different p% in the experiments. The detailed differences between the two scenarios are shown in Fig. 4, and we need to predict the unknown parts.

To develop long-term RS for both cold-start and warm-start scenarios, the users in the environment are split into the training group and the test group. During training, the inside neural network parameters of the agent are learned from complete interactions with users in the training group. During test, the long-term recommendation performance for a user is the average results of all steps in the corresponding interactions with users in the test group. Straightforwardly, the overall performance of the system can be evaluated by the recall-based metric *hit rate* and the precision-based metric NDCG, which can be calculated by Eq. (6) and Eq. (7), respectively.

$$hit@N = \frac{\Sigma_u \frac{1}{|I_u|} \Sigma_{t=1}^{|I_u|} f_{u,t}}{\#user}. \tag{6}$$

$$NDCG@N = \frac{\Sigma_u \frac{1}{|I_u|} \Sigma_{t=1}^{|I_u|} \frac{DCG@N(P^N_{u,t})}{iDCG@N}}{\#user}. \tag{7}$$

Generally, the traditional ways of evaluating the performance of top-N recommendation are composed of "Leave One Out" and "negative sampling" [51], which are unable to deal with in long-term recommendation. More specifically, the long-term recommendation performance cannot be revealed via "Leave One Out", since it always utilizes the last item for each user to be the test set. Besides, to evaluate the performance at each step, "negative sampling" should construct a candidate list previously, which includes one definite positive item and a certain number of negative items (for example 99), and then the corresponding top-N list is selected from the candidate list. However, for the long-term recommendation, there are multiple optional positive items at each step. As a result, determining one definite positive item is impossible. Consequently, in our evaluation, we assess the hit or the NDCG over a period, and the candidate list at each step includes all items. The detailed differences between our evaluation method and the traditional methods are shown in Fig. 5.

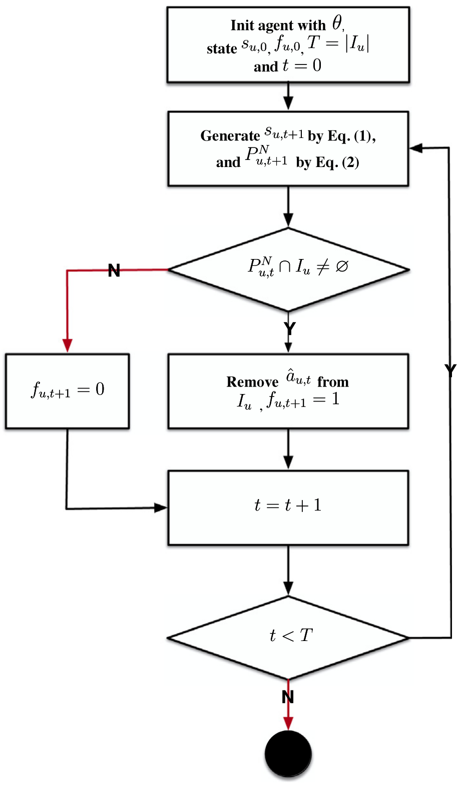
Init agent with θ, state $s_{u,0}$, $f_{u,0}$, $T = |I_u|$ and $t = 0$
Generate $s_{u,t+1}$ by Eq. (1), and $P^N_{u,t+1}$ by Eq. (2)
$P^N_{u,t} \cap I_u \neq \varnothing$
N
Y
Y
$f_{u,t+1} = 0$
Remove $\hat{a}_{u,t}$ from I_u, $f_{u,t+1} = 1$
$t = t + 1$
$t < T$
N

Fig. 3. Complete interactive processes for user u.

5. Recommendation agent

In this section, we will introduce our neural network-based recommendation agent for cold-start and warm-start scenarios.

5.1. Cold-start model

In our agent, recommendation processes are viewed as Markov decision processes (MDP), which provides a more appropriate frame for RS since it takes the long-term effects of each recommendation and the expected value of the corresponding recommendation into account.

To model the decision-making processes for sequential recommendations, we employ the recurrent neural network (RNN) as the basic framework for our model, since the prediction made by RNN for each step correlates to the input of the current step and the RNN state of the last step, which fits well with the nature of MDP.

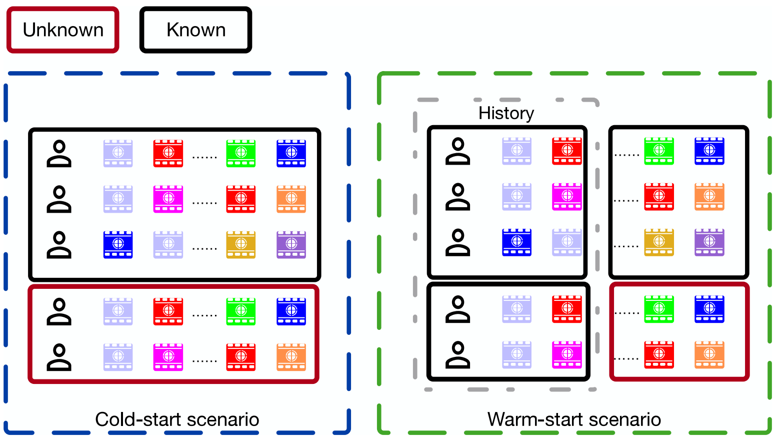

Fig. 4. Cold-start and warm-start scenarios. The objective of recommendation is to find as many items as possible in the unknown part on basis of the known items.

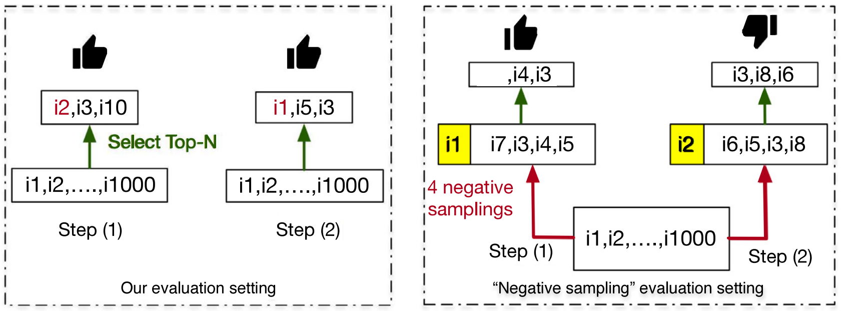

Fig. 5. Our evaluation settings v.s. negative sampling. $|I|$ is 1000, and recommend a user items with order $[i_1, i_2]$.

For sequential prediction problems, the traditional architecture of RNN includes 3 major components, (1) input layer, (2) recurrent layer, and (3) output layer. For the input layer, learnable embedding is usually fed into the neural network since the dense embedding vector can effectively learn the high-level abstracted information of items. For the recurrent layer, GRU and LSTM are the most popular choices. In this paper, we use GRU [52] to build our recurrent layer as its sequential nature and its impressive results in session-based recommender systems [33]. For the output layer, we follow the standard settings by adopting a dense layer with a Softmax operation to predict the recommendation probabilities of items.

In the following, we will gradually exhibit the details of each component in our model from the output layer to the input layer.

The output layer. The goal of recommendation is to generate a recommendation list based on the recommendation probabilities of items made by the RNN. Let $\pi(i|s_{u,t})$ be the recommendation probability of item i given $s_{u,t}$, which is obtained from the RNN via the output layer. Then the generation function $\mathbb{G}$ is defined as follows,

$$P_{u,t}^{N} = \mathbb{G}(s_{u,t}, I) = \underset{i\in I}{TopN}\left(\pi(i|s_{u,t}) \times m_{u,i}^{t}\right), \tag{8}$$

where $m_{u,i}^t$ is the element for item i's masking vector m_u^t. The value of $m_{u,i}^t$ is 0 or 1, which indicates whether the item had been selected by the agent or user before. Note that the initial m_u^0 is a vector of ones in cold-start scenario. In addition, if $m_{u,i}^t = 0$, then $m_{u,i}^{t+1}$ is equal to 0 too.

By introducing the masking vector into Eq. (8), our model can guarantee that the items that have been recommended before would not be recommended repeatedly. Moreover, the way of using masking vectors to forbid the selections of some items is differentiable during training. Then, following most settings of top-N recommender systems, we choose the items with top-n highest scores or selection probabilities to construct the recommendation list.

The recommendation probability $\pi(i|s_{u,t})$ for item i at time t can be yielded by:

$$\pi(i|s_{u,t}) = Softmax(o_{u,t}^{i}), \tag{9}$$

where $s_{u,t}$ is a l-dimensional vector obtained from the recurrent layer (we will show how to obtain $s_{u,t}$ later) and $o_{u,t}^i$ is the ith element in the vector $o_{u,t}$, which is denoted as follows,

$$o_{u,t} = \hat{\sigma}(W_s s_{u,t} + b_s), \tag{10}$$

where $\hat{\sigma}$ is the *relu* activation function, $W_s \in \mathbb{R}^{M\times l}$ and $b_s \in \mathbb{R}^M$ are parameters matrix and bias respectively, and the *Softmax* function is represented as

$$Softmax(o_{u,t}^{i}) = \frac{\exp^{o_{u,t}^{i}}}{\sum_{k\in I}\exp^{o_{u,t}^{k}}}. \tag{11}$$

The recurrent layer. The goal of the recurrent layer is to generate a new RNN state $s_{u,t}$ at time t based on the previous state $s_{u,t-1}$ and the input of recurrent layer $\hat{a}_{u,t-1}$ to simulate the state transition process.

Specifically, $\hat{a}_{u,t-1}$ is obtained from the input layer (we will show how to obtain it later), and the GRU-based recurrent layer can be denoted as follows,

$$s_{u,t} = (1 - Z(x_{u,t}, s_{u,t-1})) \odot s_{u,t-1} + Z(x_{u,t}, s_{u,t-1}) \odot \tilde{S}(x_{u,t}, s_{u,t-1}), \tag{12}$$

where $\odot$ denotes element-wise product, Z is the function for update gate and $\tilde{S}$ is the function to generate candidate state as shown in Eq. (13) and Eq. (14), respectively,

$$Z(x_{u,t}, s_{u,t-1}) = \sigma(W_z x_{u,t} + U_z s_{u,t-1}), \tag{13}$$

$$\tilde{S}(x_{u,t}, s_{u,t-1}) = \tanh(Wx_{u,t} + U(j_{u,t} \odot s_{u,t-1})), \tag{14}$$

where σ is the *sigmoid* activation function, $W_z \in \mathbb{R}^{l\times \hat{l}}$, $U_z \in \mathbb{R}^{l\times l}$ $W \in \mathbb{R}^{l\times \hat{l}}$ and $U \in \mathbb{R}^{l\times l}$ are parameters matrices, and $j_{u,t}$ is the reset gate which can be obtained by

$$j_{u,t} = \sigma(W_j x_{u,t} + U_j s_{u,t-1}), \tag{15}$$

where $W_j \in \mathbb{R}^{l\times \hat{l}}$ and $U_j \in \mathbb{R}^{l\times l}$ are parameters matrices.

The input layer. In the input layer, we aim to transform the user feedback and the result of the previous recommendation step into a dense embedding.

In most existing works for the RNN-based recommender systems, the input is usually an embedding for a certain item. However, in our problem, we need to represent a recommendation list through feedback, which is slightly different from the traditional scenarios. As a result, we first convert the recommendation list to a single item, and use it to represent the previous recommendation result. Then, we incorporate the feedback with the embedding of this representative item to form the input of the recurrent layer.

The first step is to select a representative item $\hat{a}_{u,t-1}$ from the recommendation list. Two different strategies are employed to handle the recommendation results with positive and negative feedback, respectively. For the positive feedback, we can directly use the recommended item as the representative item. However, for the negative feedback, it is unable to determine which item is related to the user. As a result, we use the item with the highest predicted recommendation probability to be the representative item as it can highly indicate the estimated preference of the user. Consequently, the selection of a representative item $\hat{a}_{u,t-1}$ based on the feedback can be formally denoted as follows,

$$\hat{a}_{u,t-1} = \underset{a\in A_{u,t-1}}{argmax}\left(\pi(a|s_{u,t-1}) \times m_{u,i}^{t-1}\right), \tag{16}$$

where $A_{u,t-1}$ is an auxiliary set for different situations of feedback, which can be denoted as Eq. (17).

$$A_{u,t-1} = \begin{cases} P_{u,t-1}^{N} \cap I_u & f_{u,t-1} > 0, \\ P_{u,t-1}^{N} & \text{otherwise.} \end{cases} \tag{17}$$

The second step is to incorporate the polar of feedback into the embedding of $\hat{a}_{u,t-1}$. Suppose that $\hat{e}(\hat{a}_{u,t}) \in \mathbb{R}^{\hat{l}}$ is the $\hat{l}$-dimensional item embedding of item $\hat{a}_{u,t}$, directly using $\hat{e}(\hat{a}_{u,t})$ is unable to distinguish what kind of feedback the representative item is accompanied with. For this purpose, we simply multiply $\hat{e}(\hat{a}_{u,t})$ by the polar of feedback.

Finally, the input of recurrent layer $x_{u,t}$ can be obtained as follows,

$$x_{u,t} = E(\hat{a}_{u,t-1}) = \begin{cases} \hat{e}(\hat{a}_{u,t}) & f_{u,t} > 0, \\ -\hat{e}(\hat{a}_{u,t}) & \text{otherwise.} \end{cases} \tag{18}$$

The overall architecture of our cold-start model is exhibited in Fig. 6. The initial state $s_{u,0}$ is a zeros vector, $f_{u,0}$ is 1, and a dummy token *null* is used to represent the starting action $\hat{a}_{u,0}$. Let $\tilde{\theta}$ be the set of parameters in our model to be learned, which includes $W_z, U_z, W, U, W_j, U_j, W_s, b_s$ and $\hat{e}(*)$. $\tilde{\theta}$ can be leaned via end-to-end, and we will show how to learn $\tilde{\theta}$ in next section.

The key to the cold-start problem is how to represent users with little or no historical interaction data. Traditional models for the cold-start users heavily rely on various side-information of users, e.g., users' profiles, social relations, or content information, to obtain the features of users. In contrast with the traditional methods, our model does not use any side-information or historical rating data (in both training and testing), but it gradually derives the users' features through the feedback with continuous interactions. In other words, the users' features are constructed on the fly rather than denoted by some static and fixed information beforehand.

To better understand the mechanism of making recommendations for the cold-start users, we use an example in Fig. 6 to clearly show how recommendations are made for different users and how to distinguish users without any side-information and ratings.

As shown in the figure, at the first recommendation step $t = 1$ for users A and B, the initial inputs $a(1)$ and $b(1)$ (zero token) and RNN states $s^a(0)$ and $s^b(0)$ (zero vector) for users A and B are identical. Note that the RNN state $s_{A,t}$ and $s_{B,t}$ can be viewed as the users' features at time t. As a result, in the first recommendation steps for users A and B, the recommendation lists $P_{A,1}^3$ and $P_{B,1}^3$ are indeed the same, where $P_{A,1}^3$ means a top-3 recommendation list for user A at step 1, and $P_{B,1}^3$ denotes similarly. However, users A and B my have different reflections for the same recommendation result. For example, user A chooses item i_1 and user B selects item i_3. Such that the inputs of RNN $a(2) = i_1$ and $b(2) = i_3$ for the next time $t = 2$ recommendations and the further obtained RNN states $s_{A,2}$ and $s_{B,2}$ become different, and then the recommendation lists $P_{A,2}^3$ and $P_{B,2}^3$ would also be different. As the interactions progress, the RNN states of different users (users' features) become more and more different. Therefore, the users can be represented and distinguished by their feedback from the previous recommendations.

5.2. Warm-start model

In the warm-start recommendation scenario, there are some historical interactions supplied for each user, which is different from the cold-start scenario. To effectively utilize the historical interactions, we propose a more general recommendation model based on our cold-start model to incorporate historical information for the warm-start recommendation.

Generally speaking, our warm-start model is derived from the cold-start model. Therefore, it shares the same recurrent neural network framework with the cold-start model. Similar to the cold-start model, our warm-start model also consists of three major components, the input layer, the recurrent layer, and the output layer. More specifically, the input layer and the output layer are the same with the cold-start model, while the recurrent layer is modified to utilize the historical information. Therefore, in this subsection, we focus on the modified recurrent layer.

Warm-start recurrent layer. Let $\hat{I}_u$ be the set containing all historical items engaged with user u (note that $\hat{I}_u \cap I_u = \varnothing$, and $m_{u,i}^0 = 0$ if $i \in \hat{I}_u$). Namely, items in user's history are unable to be selected during interactions.

There are two key issues to incorporate the historical items into the original recurrent layer of the cold-start model, which are (1) how to represent the historical items, and (2) how to balance the effects between the representation of historical items and the evolving state. To address these two issues, an additional gated neural network layer with GRU named "External Memory Gated Recurrent Unit (EMGRU)" is proposed. The details of EMGRU are introduced as follows.

The first step is to represent the historical items. Since the amount of historical items varies from different users, we need to aggregate different numbers of items to obtain fixed-size features h_u, so that the other parts of the recurrent layer can handle them. To solve this problem, inspired by Time-SVD, which sums up the embeddings of all historical items to get the aggregation result, a simple sum operation is employed as shown in Eq. (19).

$$h_u = \tanh(\sum_{i\in \hat{I}_u} e(i)), \tag{19}$$

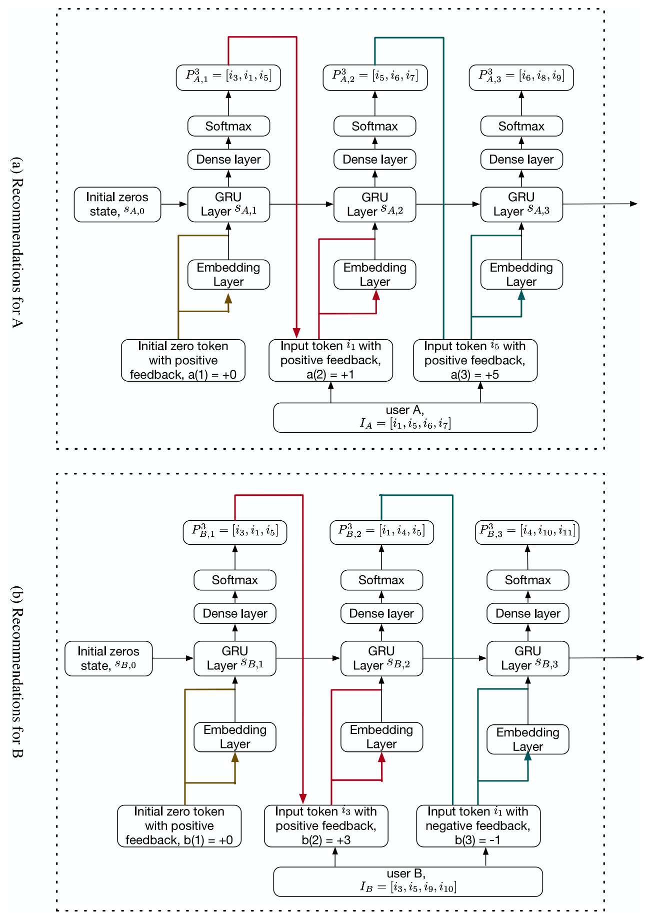
(a) Recommendations for A
$P^3_{A,1} = [i_3, i_1, i_5]$
$P^3_{A,2} = [i_5, i_6, i_7]$
$P^3_{A,3} = [i_6, i_8, i_9]$
Softmax
Dense layer
Initial zeros state, $s_{A,0}$
GRU Layer $s_{A,1}$
GRU Layer $s_{A,2}$
GRU Layer $s_{A,3}$
Embedding Layer
Initial zero token with positive feedback, a(1) = +0
Input token i_1 with positive feedback, a(2) = +1
Input token i_5 with positive feedback, a(3) = +5
user A, $I_A = [i_1, i_5, i_6, i_7]$
(b) Recommendations for B
$P^3_{B,1} = [i_3, i_1, i_5]$
$P^3_{B,2} = [i_1, i_4, i_5]$
$P^3_{B,3} = [i_4, i_{10}, i_{11}]$
Initial zeros state, $s_{B,0}$
GRU Layer $s_{B,1}$
GRU Layer $s_{B,2}$
GRU Layer $s_{B,3}$
Initial zero token with positive feedback, b(1) = +0
Input token i_3 with positive feedback, b(2) = +3
Input token i_1 with negative feedback, b(3) = -1
user B, $I_B = [i_3, i_5, i_9, i_{10}]$

Fig. 6. The cold-start model. There are two examples to exhibit two rounds of top-3 recommendations for two different users A and B. The red and green lines denote the interactions at time 1 and 2, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

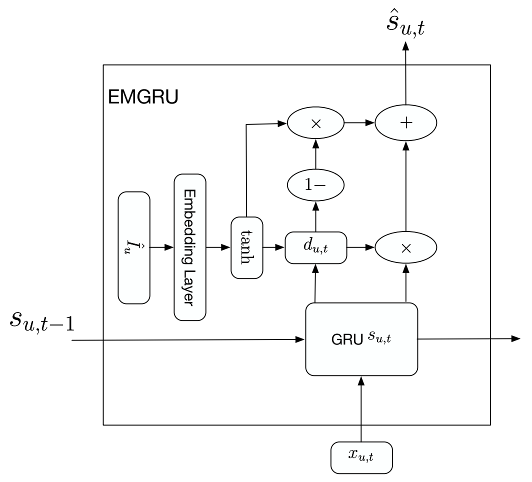

Fig. 7. Illustration of the EMGRU cell.

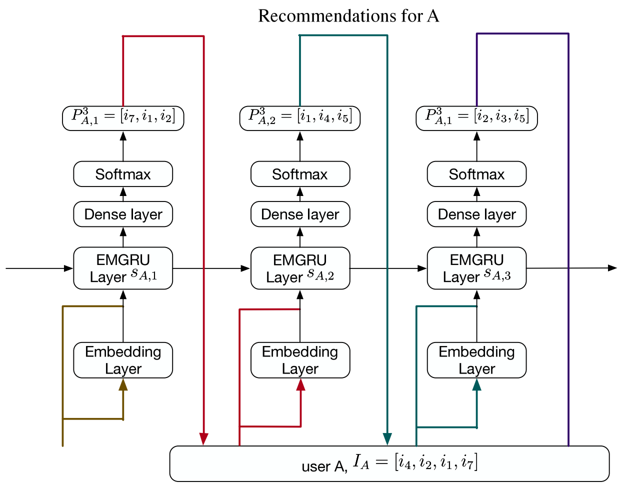
Recommendations for A
$P^3_{A,1} = [i_7, i_1, i_2]$
$P^3_{A,2} = [i_1, i_4, i_5]$
$P^3_{A,1} = [i_2, i_3, i_5]$
Softmax
Dense layer
EMGRU Layer $s_{A,1}$
EMGRU Layer $s_{A,2}$
EMGRU Layer $s_{A,3}$
Embedding Layer
user A, $I_A = [i_4, i_2, i_1, i_7]$

Fig. 8. The warm-start model. It is very similar to our cold-start model, but EMGRU is used instead of traditional GRU, which takes historical data into consideration.

where $e(i) \in \mathbb{R}^{\hat{l}}$ is the $\hat{l}$-dimensional item embedding to denote item i which needs to be learned and is different from $\hat{e}_i$ introduced before. Specifically, in order to avoid the subsequent balance operation being affected by the value range, an additional tanh operation is involved in Eq. (19) to keep the value range of h_u consistent with the state of GRU $s_{u,t}$ whose range is $(-1, 1)$.

The second step is to balance the effects between the representation of historical items h_u and the GRU state $s_{u,t}$. In the previous subsection, $s_{u,t}$ is obtained via the traditional recurrent layer by Eq. (12), and then it is fed to the next output layer to generate the final recommendation result. In this subsection, a new state $\hat{s}_{u,t}$ involving both h_u and $s_{u,t}$ is denoted as follows,

$$\hat{s}_{u,t} = d_{u,t} \odot s_{u,t} + (1 - d_{u,t}) \odot h_u, \tag{20}$$

where $d_{u,t}$ is the important or balance ratio between h_u and $s_{u,t}$.

Considering that the correlation between h_u and $s_{u,t}$ can be alterable for different users or different time steps, $d_{u,t}$ should not be fixed. Similar to the gate operation in GRU, we introduce an additional gate operation to predict the value of $d_{u,t}$, which is denoted as follows,

$$d_{u,t} = \sigma(W_d h_u + U_d s_{u,t}), \tag{21}$$

where $W_d \in \mathbb{R}^{l \times \hat{l}}$ and $U_d \in \mathbb{R}^{l \times l}$ are parameters matrices.

Note that h_u or $\hat{s}_{u,t}$ does not affect state $s_{u,t}$ transition processes $\mathbb{T}$, and h_u only impacts the generation of item selection probability. Namely, the branches of the static part and the dynamic part are independent of each other. The cell of EMGRU is illustrated in Fig. 7.

The final item selection probability $\pi(i|\hat{s}_{u,t})$ is obtained from $\hat{s}_{u,t}$ instead of generating them from state $s_{u,t}$, which can be denoted as follows,

$$\pi(i|\hat{s}_{u,t}) = Softmax(\hat{o}^i_{u,t}), \tag{22}$$

where $\hat{o}_{u,t} = \hat{\sigma}(W_s \hat{s}_{u,t} + b_s)$.

Different from the cold-start model, the set of parameters for the warm-start one is $\hat{\theta} = \tilde{\theta} \bigcup \{e(*), W_d, U_d\}$. Additionally, both warm-start and cold-start models utilize the same training method. The overall architecture of our warm-start model is exhibited in Fig. 8.

6. Learning

In this section, we will introduce how to train our recommendation models from the sequential interactions between agent and environment.

6.1. Reinforcement learning

We propose to learn the parameters θ of the agent (θ can be $\tilde{\theta}$ or $\hat{\theta}$) with REINFORCE [6], a policy gradient algorithm in RL, of which the goal is to maximize the expected long-term recommendation return. Due to its simplicity and effectiveness, REINFORCE has been widely applied to various tasks. And there are two main motivations for applying the REINFORCE algorithm to our model. The first one is that REINFORCE is directly correlated to the action (item) selection probability, which is highly related to the recommendation problem. The second one is that the form of REINFORCE is similar to the training loss of traditional supervised learning algorithms (such as the popular cross-entropy loss). Therefore, REINFORCE makes it much easier for the previously proposed supervised learning-based neural network architectures to adapt to reinforcement learning. In this paper, θ is learned via the episode E_u for each user u, and E_u means a complete interaction process for user u obtained from the agent with current parameters.

Formally, an episode E_u includes immediate reward $V_{u,t}$, state $s_{u,t}$ and action $\hat{a}_{u,t}$ for each time t, which can be denoted as follows,

$$E_u = [s_{u,1}, \hat{a}_{u,1}, f_{u,1}, V_{u,1}, \ldots, s_{u,M}, \hat{a}_{u,M}, f_{u,M}, V_{u,M}], \tag{23}$$

where $s_{u,t}$ is generated from Eq. (12), $\hat{a}_{u,t}$ is acquired from Eq. (16), $f_{u,t}$ is obtained by Eq. (4), and $V_{u,t}$ can be calculated by

$$V_{u,t} = \begin{cases} 1.0 & f_{u,t} > 0, \\ -0.2 & \text{otherwise}, \end{cases} \tag{24}$$

where the values of 1.0 and −0.2 are determined empirically.

To maximize the expected cost J, each action $\hat{a}_{u,t}$ has not only an immediate reward $V_{u,t}$, but also a long-term reward $R_{u,t}$ denoted as,

$$R_{u,t} = \Sigma_{k=0}^{M-k} \gamma^k V_{u,t+k}, \tag{25}$$

where $\gamma \in [0, 1]$ is the discount factor, and the objective function J is

$$J = \mathbb{E}_{s_{u,1}, a_{u,1}, \ldots}[R_{u,t}]. \tag{26}$$

According to REINFORCE, the agent parameters θ can be optimized via gradient ascent as follows,

$$\theta = \theta + \eta \nabla_\theta J, \tag{27}$$

where η is the learning rate, and the gradient $\nabla_\theta J(\theta)$ is

$$\nabla_\theta J = \sum_{t=1}^{M} \gamma^{t-1} R_{u,t} \nabla_\theta \log \pi(\hat{a}_t | s_{u,t}). \tag{28}$$

The episode should be a complete E_u in the standard practice of applying REINFORCE, namely, the parameters should be updated after finishing the interaction processes for user u. However, the lengths of $I*$ for different users vary greatly, for example, $|I_A| = 20$ but $|I_B| = 200$, which leads to large variance regarding $R_{*,t}$ for different users at the same time t. In addition, a long episode would cause good results to be hidden due to the accumulation of excessive negative returns, which makes it difficult for the agent to obtain positive training samples. As a result, the agent learned by traditional REINFORCE cannot achieve satisfied recommendation performance.

To overcome this problem, we propose to split the original E into I_u/B sub-episodes and reboot reward accumulation at the beginning of each sub-episode. The learning processes for $\hat{\theta}$ and θ are likewise. The detailed processes of learning and sub-episodes generation are shown in Algorithm 1 and Algorithm 2, respectively. Note that, in the warm-start scenario, to keep sufficient training data, the overall length of interaction processes is equal to $|I_u \cup \hat{I}_u|$ for each user u in training, and the agent is able to select items in $\hat{I}_u$. However, during the test, the length of interaction processes is still equal to $|I_u|$, and the agent cannot select items in $\hat{I}_u$ for each user u.

Algorithm 1 :Reinforcement Learning

```
Require: training set $U$, learning rate $\eta$, maximum size $B$, initialized $\theta$.
 1: while not stop do
 2:   for $u$, $I_u$,$\hat{I}_u$ in $U$ do
 3:     $count \leftarrow 0$; $s_{u,0} \leftarrow$ zeros; $\hat{a}_{u,0} \leftarrow 0$;
 4:     if $\hat{I}_u$ is $\emptyset$ then
 5:       $m_u \leftarrow$ ones.
 6:     else
 7:       $m_u[i] \leftarrow 0$ for each item $i$ in $\hat{I}_u$.
 8:     end if
 9:     while $count < |I_u + \hat{I}_u|$ do
10:       $E, m_u \leftarrow$ GenerateEpisode($I_u$,$\hat{I}_u$,$s_{u,0}$,$\hat{a}_{u,0}$,$f_{u,0}$,$m_u$, $B$)
11:       $[s_{u,1}, \hat{a}_{u,1}, f_{u,1}, V_{u,1}, \ldots, s_{u,B}, \hat{a}_{u,B}, f_{u,B}, V_{u,B}] = E$
12:       Obtain $R_{u,t}$ according to Eq. (25)
13:       $\theta \leftarrow \theta + \eta \frac{1}{B} \sum_{t=1}^{B} \gamma^{t-1} R_{u,t} \nabla_\theta \log \pi(\hat{a}_t | s_{u,t})$
14:       $count += B$; $s_{u,0} \leftarrow s_{u,B}$, $\hat{a}_{u,0} \leftarrow \hat{a}_{u,B}$, $f_{u,0} \leftarrow f_{u,B}$
15:     end while
16:   end for
17: end while
```

Algorithm 2 : GenerateEpisode

```
Require: $I_u$,$\hat{I}_u$, $s_{u,0}$,$\hat{a}_{u,0}$,$f_{u,0}$, $m_u$, $B$
 1: E=[]
 2: for $t = 1$ to $B$ do
 3:   State transition $s_{u,t} \leftarrow \mathbb{T}(s_{u,t-1}, f_{u,t-1})$ for cold-start or warm-start scenario.
 4:   Acquire Top-N recommendation list $P^N_{u,t} \leftarrow \mathbb{G}(s_{u,t}, I)$ with masking vector $m_u$
 5:   Obtain current feedback $f_{u,t} \leftarrow \mathbb{F}(P^N_{u,t}, I_u)$
 6:   Obtain $\hat{a}_{u,t}$ according to Eq. (16).
 7:   Obtain $V_{u,t}$ according to Eq. (24).
 8:   Append ($s_{u,t}$, $\hat{a}_{u,t}$, $f_{u,t}$, $V_{u,t}$) at the end of $E$.
 9:   if $f_{u,t} > 0$ then
10:     $m_u[\hat{a}_{u,t}] \leftarrow 0$.
11:   end if
12: end for
13: Return $E$, $m_u$
```

6.2. Supervised learning

The other way of training our recommendation agent is to optimize it by SL in short-term prediction scenarios, and then apply it to the long-term test environments. In order to easily transfer the recommendation agent from a short-term scenario to a long-term scenario, the architectures of the neural network for SL and RL should be consistent. And the only difference is that there are explicit labels given to SL, where the labels are the actually selected items by the user at each time step.

Note that I_u is the actual sequence of items selected by user u as time progresses. To maximize the short-term prediction accuracy, we employ *cross-entropy* $\hat{J}$ to be the cost function for SL, which can be denoted as follows,

$$\hat{J} = -\Sigma_{i=1}^{B} \log(\pi(I_{u,i} | s_{u,i})), \tag{29}$$

where θ can be updated by:

$$\theta = \theta - \eta \nabla_\theta \hat{J}. \tag{30}$$

The processes of applying SL are similar to Algorithm 1, and the details are exhibited in Algorithm 3. The differences between

Algorithm 3 : Supervised Learning

```
Require: $U$, $\eta$, $B$, $\theta$.
 1: while not stop do
 2:   for $u$, $I_u$ $\hat{I}_u$ in $U$ do
 3:     $count \leftarrow 0$,$s_{u,0} \leftarrow$ zeros,$\hat{a}_{u,0} \leftarrow 0$.
 4:     if $\hat{I}_u$ is $\emptyset$ then
 5:       $m_u \leftarrow$ ones.
 6:     else
 7:       $m_u[i] \leftarrow 0$ for each item $i$ in $\hat{I}_u$.
 8:     end if
 9:     while $count < |I_u + \hat{I}_u|$ do
10:       $E$, $m_u \leftarrow$ SampleEpisode($I_u$,$\hat{I}_u$, $s_{u,0}$, $\hat{a}_{u,0}$$f_{u,0}$,$m_u$,$B$)
11:       Update $\theta$ according to Eq. (30)
12:       $count += B$, $s_{u,0} \leftarrow s_{u,B}$, $\hat{a}_{u,0} \leftarrow \hat{a}_{u,B}$, $f_{u,0} \leftarrow f_{u,B}$.
13:     end while
14:   end for
15: end while
```

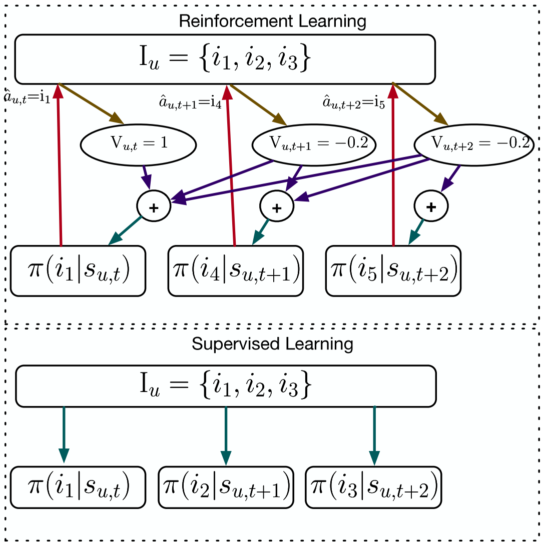

Fig. 9. Applying RL and SL to our agent. Here, green line: "labels"; red line: actions; brown line: immediate rewards; purple line: long-term rewards. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

applying RL and SL are shown in Fig. 9. In contrast with our RL-based learning method, the SL-based one is similar to traditional session-based RNN, in which each step of recommendation has a certain corresponding label to the training signal.

Moreover, the agent can be fine-tuned by SL before applying RL, which can help the RL-based agent start from a relatively good policy instead of a random one. Thus it can accelerate the convergence speed of RL. The overall training processes are shown in Algorithm 4. We also observe that this way of learning can bring significant improvements to the large-scale datasets (see Section Experiments).

Algorithm 4 : Overall Learning

```
1: Initialize $\theta$ with random weights;
2: Run Algorithm 3 to pre-train $\theta$ until stop;
3: Run Algorithm 1 with the pre-trained $\theta$ until converge.
```

6.3. Time complexity

In this subsection, we will analyze the time complexity of our model for making sequential recommendations for a user.

Suppose that the time complexity of generating a recommendation for one time step is $\mathbb{O}(f)$, and $\mathbb{O}(f)$ depends on the architecture of neural network, such as the number of layers, the size of each layer. Specifically, the main framework of our model is based on the RNN with GRU, and the number of layers in our model is 3, including 1 embedding layer for input, 1 recurrent layer with GRU or EMGRU, and 1 dense layer with Softmax for output. In the embedding layer, the main operation is to retrieve the item embedding according to the item index, and the time complexity is $\mathbb{O}(1)$. In the recurrent layer with GRU, the main time complexity comes from 3 matrix–vector product operations, thus, the overall time complexity is $\mathbb{O}(3 \times (d \times h + h \times h))$, where d and h are the size of embedding and output of recurrent layer, respectively. For the recurrent with EMGRU, the main increasing time complexity in contrast with GRU is derived from one additional gate operation, therefore, the time complexity of EMGRU is $\mathbb{O}(4 \times (d \times h + h \times h))$. Consequently, the time complexities of GRU and EMGRU are close, and both of them approximate to $\mathbb{O}(h^2)$ as $h \approx d$ and $h \gg 4$ in our settings. For the output layer, the time complexity is $\mathbb{O}(h \times |I|)$ derived from one matrix–vector product operation in the dense layer. As a result, the time complexity for making a prediction through the neural network is $\mathbb{O}(f) = \mathbb{O}(h^2 + h \times |I|)$. Considering there is a sorting operation for retrieving top-N recommendation list ($\mathbb{O}(|I|log|I|)$), and there are T steps of recommendations for a user, then the overall time complexity for making a sequential recommendation for a user is $\mathbb{O}(T \times (h^2 + h \times |I| + |I|log|I|))$.

7. Experiments

In this section, we conduct several experiments to demonstrate the advantages of our method on long-term recommendations and exhibit the effectiveness of the core components of our model.[1]

7.1. Evaluation datasets and experiment settings

We adopt three offline real-world benchmarks, MovieLens 100K, MovieLens 1M and Steam to evaluate our models.

MovieLens 100K: A widely used benchmark dataset for evaluating CF-based algorithms. It contains around 100 thousand ratings of approximately 1682 movies by 943 users.

MovieLens 1M: It varies from MovieLens 100K in dataset size, and contains around 1 million ratings of approximately 3900 movies by 6040 users.

Steam: A dataset contains the reviews of games by users. Considering Steam dataset is very sparse, and the average interaction number of users is very small (only around 10). For evaluating the long-term performance, we keep the users with at least 50 interactions, and the items with at least 10 interactions. After applying the above filtering pre-process, the number of remaining users and items are 6818 and 8023, respectively. The average number of interactions of users is about 100. Even when filtering is applied, the modified Steam dataset is still more sparse than the MovieLens datasets.

We employ mainly two metrics *hit@N* (Eq. (6)) and *NDCG@N* (Eq. (7)) to measure the performance of the proposed method. Moreover, in some experiments, we introduce the inter-diversity metric [53] (denoted as D@N) to evaluate the diversities of generated recommendation lists for different users, which can be

[1] Our codes are available at https://github.com/lw-liweihuang/KBS-RL/tree/main.

Table 2
Cold-start performance comparisons on MovieLens 100K and 1M.

Method	MovieLens 100K						MovieLens 1M					
	hit@10	N@10	hit@15	N@15	hit@20	N@20	hit@10	N@10	hit@15	N@15	hit@20	N@20
Pop	12.82%	3.87%	15.71%	4.32%	18.20%	4.62%	7.77%	2.50%	10.38%	2.86%	12.61%	3.16%
ε-greedy	29.39%	8.01%	37.33%	8.89%	42.12%	9.28%	20.91%	4.57%	25.43%	4.59%	28.39%	4.65%
L-UCB	31.14%	7.73%	36.04%	7.39%	38.50%	7.82%	20.67%	4.48%	25.07%	4.54%	28.18%	4.59%
sRNN	38.82%	8.58%	43.58%	9.10%	48.17%	9.28%	18.74%	4.40%	24.32%	4.65%	26.25%	5.00%
SASRec	46.70%	12.63%	55.21%	13.48%	57.47%	13.46%	43.48%	11.32%	46.73%	11.40%	51.34%	11.43%
HGN	31.17%	4.50%	44.58%	6.30%	45.06%	8.12%	30.88%	4.70%	32.35%	4.84%	32.96%	4.98%
DQN	38.05%	8.37%	–	–	48.31%	7.11%	31.97%	7.04%	–	–	43.37%	6.20%
sl-cold	46.48%	10.71%	52.90%	10.92%	57.39%	11.46%	32.17%	6.03%	37.08%	6.41%	42.41%	6.68%
rl-cold	59.15%	16.77%	63.22%	15.69%	68.06%	15.67%	44.62%	9.97%	48.73%	9.04%	53.79%	9.03%
sl+rl-cold	57.37%	16.96%	65.94%	17.18%	69.48%	16.93%	45.94%	12.21%	48.94%	10.13%	51.97%	9.96%

calculated as follows.

$$H_u = \frac{1}{m(m-1)} \sum_{u=1}^{m} \sum_{u \neq u'} [1 - (\frac{Q_{uu'}}{N})], \tag{31}$$

where $Q_{uu'}$ is the similarity between the two users' recommendation lists, which is defined as the number of common items in both lists. More details about the diversity metric can be found in [53]. To adapt Eq. (31) to the setting of long-term recommendation, for each user, we select only one recommendation list at the middle of the entire interactions to represent the recommendation result, as Eq. (31) implies each user has only one recommendation list.

In our experiments, 10% of users are randomly selected as the test set, leaving the remaining 90% as the training set. We report the average performance on the test set over five different splits. In the warm-start scenario, the first p% items for user u are kept to construct $\hat{I}_u$, and the rest $1 - p$% items are used to construct I_u. During training, $\hat{I}_u \cup I_u$ is used to train the agent, and there are $|\hat{I}_u \cup I_u|$ steps of interactions between user and agent. However, we only consider whether the agent hits the item in I_u within $|I_u|$ steps during the test. We employ RNN with GRU or EMGRU in our model, and the hidden layer size of RNN is 100, which is the same with the size of embedding. The batch size B is 20, $\gamma = 0.9$. Our model is optimized via Adam optimizer [54] with learning rate $\eta = 0.005$.

7.2. Baseline methods

We compare our model with the following baseline algorithms.

Pop always recommends the most popular items in the training set. Despite its simplicity, it is often a strong baseline.

Linear-UCB (L-UCB) [3] is a linear bandits algorithm, and it is a very popular and well-established algorithm for IRS. In this paper, We adopt the context-free bandit since we do not consider content information. Specifically, the item embeddings are obtained by MF.

ε-greedy [3] is similar to Linear-UCB, but a tuning parameter ε is used to control the balance between exploitation and exploration. Similar to Linear-UCB, the item embeddings are obtained by MF.

Session-based RNN (sRNN) [33] is a RNN-based sequential time-aware model. However, it is proposed for predicting the next item. Namely, it is a model for short-term prediction. In our experiments, the RNN is trained on the short-term prediction task. Then, we directly apply it to the long-term recommendation test environment. Especially, during the test, the input of RNN for each time is the last successful hit item. For the cold-start scenario, the initial state is a zeros vector. For the warm-start scenario, the initial state is the last RNN state corresponding to encode the items in $\hat{I}_u$.

BPR-MF [55] is a static model, which is a popular framework for the Top-N recommendation, and MF is used to be the internal predictor. Especially, it is trained with the "negative sampling" setting.

NeuCF [51] is a static model, which is the state of the art method for the Top-N recommendation. Similar to BPR, it is also trained with the "negative sampling" setting.

DQN [47] is a DRL-based model using DQN, which is a time-aware model too. However, this model needs to hold a certain number of items to construct initial states. To adapt this model to the cold-start scenario, several dummy tokens are allocated at the beginning.

SASRec [56] is a self-attention-based sequential model, which uses an attention mechanism to identify relevant items for predicting the next item.

HGN [57] applies a hierarchical gating network to learn the group-level representations of a sequence of items and adopts the item–item product to explicitly capture the item–item relations.

The hyper-parameters of the above baseline methods have been fine-tuned and report the best performance. Pop, Linear-UCB, ε-greedy are for the cold-start scenario, while BPR-MF and NeuCF can only be used in the warm-start scenario. We adapt the baselines to our interactive processes shown in Fig. 3, but the static models do not involve the state transition since they are unable to update the state and handle feedback. During the test, for each step, corresponding top-N recommendation lists are selected from all items I, and the successful hit item $\hat{a}_{u,t}$ are prevented from being selected repeatedly.

7.3. Comparisons under cold-start scenario

We compare the performance of our model in terms of Hit Rate@N (hit@N) and NDCG@N (N@N) with other baselines under the user cold-start scenario. During the comparisons, there are three different settings with respect to top-N ($N = 10, 15, 20$) are considered. We report the performance of our cold-start model with three different training strategies, which are pure supervised learning ("sl-cold"), pure reinforcement learning ("rl-cold") and hybrid learning which is reinforcement learning with supervised pre-learning ("sl+rl-cold"), respectively. The results of our model as well as the baselines on Movielens 100K and 1M are exhibited in Table 2. As shown in the table, our methods can generally outperform baselines on different settings of top-N recommendation. The results indicate that our model can effectively address the user cold-start problem. Particularly, the architecture of sRNN is similar to our sl-cold model, and both of them are trained by the short-term prediction scenario. However, the feedback acquired from the interactions between user and agent is taken into account in our sl-cold model. As a result, sl-cold is able to exceed sRNN. The performance of SASRec is the closest to ours among all the baseline methods, as it employed a more powerful self-attention architecture than RNN. However, our method can

Table 3
Warm-start performance comparisons on MovieLens 100K.

Method	MovieLens 100K									
	$p = 10\%$		$p = 30\%$		$p = 50\%$		$p = 70\%$		$p = 90\%$	
	hit@10	N@10	hit@10	N@10	hit@10	N@10	hit@10	N@10	hit@10	N@10
Pop	10.96%	2.90%	9.49%	2.16%	8.15%	1.65%	7.41%	1.31%	7.52%	1.48%
BPR	11.94%	3.41%	12.27%	3.25%	12.45%	2.89%	12.71%	2.74%	13.82%	3.23%
NeuCF	12.40%	3.51%	12.75%	3.33%	13.30%	3.02%	13.62%	2.86%	15.02%	3.35%
sRNN	39.06%	8.54%	36.67%	6.92%	32.93%	5.45%	27.08%	4.05%	19.11%	3.66%
SASRec	44.73%	10.21%	44.30%	8.34%	38.81%	5.86%	30.12%	4.04%	16.80%	2.52%
HGN	39.98%	5.58%	33.25%	4.12%	25.70%	3.10%	16.67%	1.96%	7.58%	1.91%
DQN	35.07%	7.72%	–	–	26.59%	5.90%	–	–	14.47%	4.61%
sl-cold	41.37%	8.97%	36.24%	6.76%	31.79%	5.47%	24.66%	3.96%	16.95%	3.40%
rl-cold	57.30%	14.93%	52.37%	12.11%	45.84%	8.88%	37.25%	1.24%	21.04%	3.69%
sl+rl-cold	57.16%	15.54%	52.39%	12.53%	45.93%	9.38%	36.78%	6.32%	21.22%	3.71%
sl-warm	36.93%	8.65%	35.95%	6.79%	33.82%	5.37%	30.47%	4.73%	22.27%	4.42%
rl-warm	56.09%	13.81%	51.16%	11.53%	42.03%	8.37%	38.74%	6.76%	24.23%	4.44%
sl+rl-warm	53.62%	14.34%	50.11%	12.07%	46.99%	9.64%	43.12%	7.90%	30.70%	6.18%

Table 4
Warm-start performance comparisons on MovieLens 1M.

Method	MovieLens 1M									
	$p = 10\%$		$p = 30\%$		$p = 50\%$		$p = 70\%$		$p = 90\%$	
	hit@10	N@10	hit@10	N@10	hit@10	N@10	hit@10	N@10	hit@10	N@10
Pop	6.83%	2.03%	5.73%	1.50%	5.04%	1.18%	4.52%	0.92%	4.10%	0.81%
BPR	4.73%	1.14%	4.73%	1.05%	4.95%	0.96%	4.99%	0.84%	5.15%	0.92%
NeuCF	6.27%	1.74%	6.56%	1.62%	6.65%	1.37%	6.99%	1.26%	7.78%	1.53%
sRNN	16.55%	3.26%	17.16%	3.55%	17.27%	3.37%	17.67%	3.30%	18.52%	3.79%
SASRec	41.62%	9.50%	39.69%	7.51%	34.31%	5.70%	26.29%	3.95%	18.39%	3.47%
HGN	26.82%	2.81%	20.40%	1.99%	15.33%	1.39%	10.42%	0.97%	4.68%	0.69%
DQN	32.53%	7.16%	–	–	25.40%	5.60%	–	–	12.76%	3.44%
sl-cold	31.31%	5.63%	26.99%	4.37%	22.92%	3.37%	17.82%	2.53%	9.63%	1.31%
rl-cold	41.65%	8.74%	36.93%	7.09%	32.07%	5.41%	26.07%	3.92%	14.73%	2.28%
sl+rl-cold	45.15%	11.30%	40.98%	9.31%	33.24%	6.78%	27.28%	4.96%	15.15%	2.50%
sl-warm	23.39%	3.97%	23.23%	3.51%	22.07%	2.98%	16.89%	2.08%	8.59%	1.11%
rl-warm	42.23%	9.02%	35.32%	7.47%	29.53%	5.44%	26.51%	4.19%	16.09%	2.71%
sl+rl-warm	45.06%	11.62%	43.88%	9.91%	37.14%	7.02%	31.58%	5.09%	21.20%	3.61%

outperform it with simpler neural network architecture. For MAB-based methods L-UCB and ε-greedy, feedback is also considered. However, the MAB-based methods are unable to explicitly build the sequential relations during interactions, and also they ignore long-term recommendation completely. Thus, their performance is not good enough. In contrast with the DRL-based method DQN, our model exceeds it significantly since our model explicitly considers the sequential processes of recommendation interactions with continuous feedback. Comparing the performance of sl-cold and rl-cold methods, rl-cold consistently outperforms sl-cold with a large margin, which indicates that RL is more suitable for long-term recommendation problem than SL. Incorporating SL pre-training with RL (sl+rl-cold), the performance in terms of NDCG can be improved, while it cannot always beat rl-cold in terms of HR for cold-start scenario. Note that some cells about the DQN baseline are missing since the time cost for DQN training is exceptionally high compared with other baseline methods. Thus, we just conducted experiments under certain settings in DQN. Moreover, the results under some settings are close, for instance, the result under *hit*@15 is between the results under *hit*@10 and *hit*@20.

7.4. Comparisons under warm-start scenario

The results under warm-start scenario on MovieLens 100K and 1M are shown in Table 3 and Table 4, respectively. In these tables, $p = 10\%$ means 10% items of each user u is kept to be the historical set $\hat{I}_u$ for warm-start, and there are five different settings of p considered in the following experiments.

As can be seen in the tables, both our cold-model and warm-model exceed the baselines with a large margin. In contrast with the warm-model, the historical data $\hat{I}_u$ would be directly discarded by the cold-start model, regardless of the amount of $\hat{I}_u$. Note that the amount of $\hat{I}_u$ is insufficient when p is small. As a result, the cold-start model can even perform slightly better than the warm-start model at $p = 10\%$. However, as p increases, the warm-start model overwhelms the cold-start model significantly, which indicates that taking historical data into account can improve the performance of our warm-start model if the provided historical data is sufficient enough. Besides, when more historical data is available (p is larger), RS should be able to recommend more accurate results (higher HR). However, in our model as well as baselines, all of them are reversed. The reason is that we use off-line datasets to evaluate the performance of different models. For off-line datasets, the total amount of samples ($I_u \cup \hat{I}_u$) is fixed. As a result, for each step, it becomes more difficult for RS to select correct items from a smaller set I_u with a higher p. That is why the results of our method under smaller p are relatively better than the ones on the larger p. What is more, different from the cold-start scenario, pre-training agents by SL can generally improve the performance in terms of both HR and NDCG, and this is especially obvious for the larger dataset MovieLens 1M. Without SL pre-training, rl-warm cannot even outperform sRNN at $p = 90\%$. On the contrary, the gaps between sRNN and sl+rl-warm differ greatly. Note that for the DQN baseline, there are also some missing cells in Tables 3 and 4 because of the high time cost of training DQN under all settings. Therefore, only some results are given out in these tables.

Table 5
Performance comparisons on Steam.

Method	Steam					
	$p = 0\%$			$p = 50\%$		
	hit@10	N@10	D@10	hit@10	N@10	D@10
sRNN	18.71%	2.20%	98.60%	9.83%	1.02%	98.44%
SASRec	20.06%	2.95%	85.82%	17.94%	2.08%	90.39%
HGN	19.86%	2.67%	73.04%	11.59%	1.14%	88.30%
DQN	12.28%	2.29%	72.03%	9.79%	1.35%	71.47%
sl+rl-cold	25.57%	5.59%	95.12%	16.52%	2.66%	95.51%
sl+rl-warm	–	–	–	20.99%	3.21%	98.70%

7.5. Comparisons on sparse dataset steam

In this subsection, we evaluate our model on a more sparse Steam dataset. Here, the performance of our method is measured on three metrics, hit, NDCG, and inter-diversity, under two representative recommendation scenarios, cold-start ($p = 0\%$) and warm-start $p = 50\%$. We compare our method with the baselines using sequential or context information as these methods are similar to ours and can be used in both cold-start and warm-start scenarios. Moreover, these baselines can generally outperform other baselines as shown in previous experiments on the MovieLens datasets. The experiment results are reported in Table 5.

From Table 5, we can observe that using only pure reinforcement learning cannot obtain acceptable performance in the sparse dataset Steam, as the action space (item number) is significantly larger than the MovieLens datasets. Therefore, the pure reinforcement learning model may suffer from the larger action space problem. In large action space, the sampling efficiency during training may decrease since valuable actions are harder to be discovered. On the other hand, using only pure supervised learning is sub-optimal since the long-term effects are completely ignored. However, involving supervised learning as pre-training can help the reinforcement learning-based agent to acquire a better initial policy. As a result, valuable actions can be found more easily at the beginning, and therefore the performance can be significantly improved. What is more, our method can not only achieve high accuracy prediction, but also the recommendation lists for different users are diverse. Note that some cells are missing in the table since there is no historical data (p = 0%) provided to the warm-start model.

7.6. Ablation study

We further investigate the effects of different components in our overall model through ablation studies. There are four main distinctions between our model and traditional RNN-based models: (1) using reinforcement learning; (2) pre-training by supervised learning; (3) combining the feedback with the representative item of the previous recommendation list as input; (4) using the EMGRU cell. We focus our ablation studies on these four major components, and the ablation studies are conducted on three datasets Movielens 100K, 1M and Steam under the cold-start and warm-start ($p = 50\%$) recommendation scenarios. The experimental results for cold-start and warm-start recommendation scenarios are reported in Table 6 and Table 7, respectively, where symbol "$\surd$" indicates the corresponding component is involved in our model.

Under the cold-start recommendation scenario (Table 6), by comparing the Full model with Model-1 and Model-2, we can observe that the model combining supervised learning and reinforcement learning can consistently outperform the models that use only supervised learning or reinforcement learning. In contrast with Model-3, considering feedback as input can further improve the performance of the Full model, which indicates the effectiveness of feedback input. Besides, the performance of Model-4 is the worst among all the considered models, since it does not use any component proposed in this paper.

Under the warm-start recommendation scenario (Table 7), we can obtain the following conclusions. (1) The results of Model-1 and Model-2 also demonstrate the effectiveness of combining supervised learning and reinforcement learning. (2) The performance of Model-3 indicates that combining feedback input can achieve better results, whereas the improvement is not as significant as the results in the cold-start scenario. (3) The performance of Model-4 indicates the effectiveness of EMGRU by taking the historical interactions into account, and EMGRU plays a more critical role than the feedback input component on the relatively sparse dataset Steam. (4) The results of Model-5 and Model-6 also demonstrate the effectiveness of the components proposed in our model.

Table 6
Ablation studies on MovieLens 100K, 1M and Steam under cold-start recommendation scenario ($p = 0\%$).

Model	Components			100K		1M		Steam	
	SL	RL	FI	Hit@10	N@10	Hit@10	N@10	Hit@10	N@10
Full	√	√	√	57.37%	16.96%	45.94%	12.21%	25.57%	5.59%
Model-1	√	×	√	46.48%	10.71%	32.17%	6.03%	15.23%	2.06%
Model-2	×	√	√	57.37%	16.96%	45.94%	12.21%	5.68%	1.05%
Model-3	√	√	×	50.38%	14.76%	41.49%	9.49%	16.90%	3.11%
Model-4	√	×	×	38.01%	8.74%	23.01%	4.24%	13.63%	1.86%

Table 7
Ablation studies on MovieLens 100K, 1M and Steam under warm-start recommendation scenario ($p = 50\%$).

Model	Components				100K		1M		Steam	
	SL	RL	FI	EMGRU	Hit@10	N@10	Hit@10	N@10	Hit@10	N@10
Full	√	√	√	√	46.99%	9.64%	37.14%	7.02%	20.99%	3.21%
Model-1	√	×	√	√	33.82%	5.37%	22.07%	2.98%	12.52%	1.16%
Model-2	×	√	√	√	42.03%	9.64%	29.53%	5.44%	7.59%	1.35%
Model-3	√	√	×	√	43.46%	9.15%	35.20%	6.39%	20.42%	3.14%
Model-4	√	√	√	×	45.93%	9.38%	33.24%	6.78%	16.52%	2.66%
Model-5	√	√	×	×	35.79%	6.94%	33.84%	5.98%	17.19%	2.65%
Model-6	√	×	×	×	24.23%	3.43%	14.93%	1.81%	10.36%	0.98%

7.7. Long-term performance evaluation

The recommendation results of different stages are exhibited in Fig. 10 to show the long-term recommendation performance. Here, the experimental results are obtained from users selected from the corresponding test sets. Specifically, the sizes of I_u for the selected users are above the average size since the long-term recommendation performance can be revealed more clearly by users with a longer history. Meanwhile, the results for a user u are divided into 5 different stages, [0%, 20%), [20%, 40%), [40%, 60%), [60%, 80%) and [80%, 100%], where $[s\%, e\%)$ represents the range from $(s\% \times I_u)$-th to $(e\% \times I_u)$-th steps during overall interactions. Besides, we use e to indicate the range for $[s\%, e\%)$ and report the averaged result in the range.

As exhibited in Fig. 10, the static methods like NeuCF and BPR can still achieve relatively good hit rate at the first range [0%, 20%). However, their performance decreases dramatically to 0% in the following ranges since the recommendation results obtained by the static methods stay almost unchanged. As a result, their corresponding overall HR and NDCG results are poor, as shown in Tables 3 and 4. On the other hand, the sequential

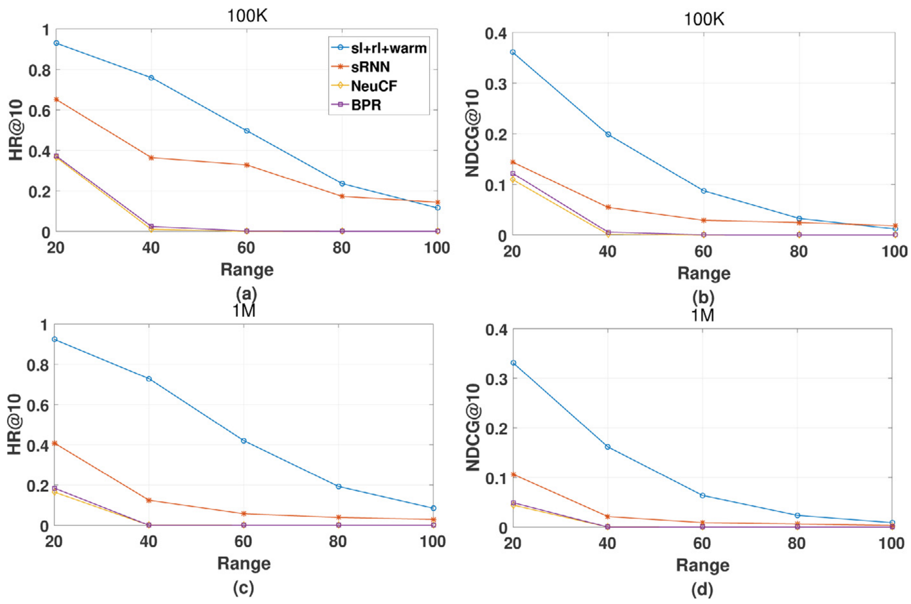

Fig. 10. Performance on different stages with $p = 50\%$.

Table 8
Performance acquired by different EMGRU settings.

Proportion	Methods	100K		1M	
		Hit@10	N@10	Hit@10	N@10
p = 10%	only h	15.57%	5.13%	7.32%	2.02%
	only s	**57.16%**	**15.54%**	**45.15%**	11.30%
	$s+h$	53.13%	13.75%	39.79%	9.91%
	$s+h$ w/ gate	53.62%	14.34%	45.06%	**11.92%**
p = 50%	only h	14.83%	3.86%	7.95%	1.75%
	only s	45.93%	9.38%	33.24%	9.38%
	$s+h$	45.06%	8.89%	**37.30%**	6.85%
	$s+h$ w/ gate	**46.99%**	**9.64%**	37.14%	**9.64%**
p = 90%	only h	15.32%	4.60%	7.57%	1.59%
	only s	21.22%	3.71%	15.15%	2.50%
	$s+h$	27.35%	5.31%	18.98%	3.16%
	$s+h$ w/ gate	**30.70%**	**6.18%**	**21.20%**	**3.61%**

methods like sl+rl-warm and sRNN are able to make a hit at all ranges since the corresponding recommendation results can be updated adaptively. In contrast with sRNN, our sl+rl-warm outperforms it significantly from range [0%, 20%) to [60%, 80%). Besides, our method achieves enough hits in earlier ranges, and then there are fewer items that can be hit in the last stage [80%, 100%] due to the fixed size of I_u. As a result, the performance of sl+rl-warm is almost the same with sRNN in the last range [80%, 100%]. These results indicate that our method can effectively adopt recommendation shifts in the long-term.

7.8. Effect of EMGRU

EMGRU is crucial in our warm-start model, as it can adaptively regularize the effects of dynamic RNN state and static historical state to generate $\hat{s}_{u,t}$. To study the influence of this adaptive balance, we consider 4 different settings of combining RNN state and historical state: (1) only using RNN state (only s); (2) only using historical representation (only h); (3) sum of historical state and RNN state ($s + h$). (4) sum of historical state and RNN state with balance gate ($s + h$ w/ gate). Here, the method "only s" represents our cold-model, and the method "$s + h$ w/ gate" represents our warm-model. The performances acquired by these models with different p settings are exhibited in Table 8.

As shown in the table, the performance achieved by "only h" method is the worst, since h_u is unchangeable throughout the processes of all interactions, and fixed h_u is unable to generate different recommendation results at different time steps. On the other hand, the performance achieved by the method "only s" can exceed the combined methods "$s+h$" and "$s+h$ w/ gate" in terms of HR at the lowest setting $p = 10\%$, since the data amount for h_u is insufficient. However, the combined methods can beat "only s" method as p increases. Moreover, composing h and s adaptively via gate generally has better performance than mixing h and s with equal constant effects. These results indicate that the effects between $s_{u,t}$ and h_u should be dynamically adjusted according to the current situations instead of staying fixed.

7.9. Convergence analysis

To exhibit the effectiveness of the settings in our REINFORCE algorithm, we present the performance curves concerning different optimizing algorithms on the test sets of MovieLens 100K for both cold-start and warm-start ($p = 50\%$) scenarios in Fig. 11. Clearly, the performance obtained by SL with a complete episode (blue line) or with separated sub-episodes (red line) is similar, which indicates that splitting the whole interaction processes into several sub-processes is unable to enhance the recommendation accuracy. Similarly, the performance acquired by the basic REINFORCE algorithm (yellow line) is poor, and it has a large margin in contrast with the SL-based methods. However, the performance can be significantly improved through the proposed special REINFORCE algorithm as shown in the purple line, since separating the overall lengthy episodes and then rebooting the accumulation of rewards can get more useful self-generated training labels for reinforcement learning. Although this way is straightforward, it is notably useful.

7.10. Analysis of recommendation behaviors

To investigate the intrinsic recommendation behaviors of our agent, we exhibit the average results of *Hit@10* and the popularity related to the selected item $\hat{a}_{u,t}$ of each step t for all test users on the 100K dataset. Especially, the results for the cold-start scenario are exhibited in Fig. 12, where the range of step is from 0 to 200

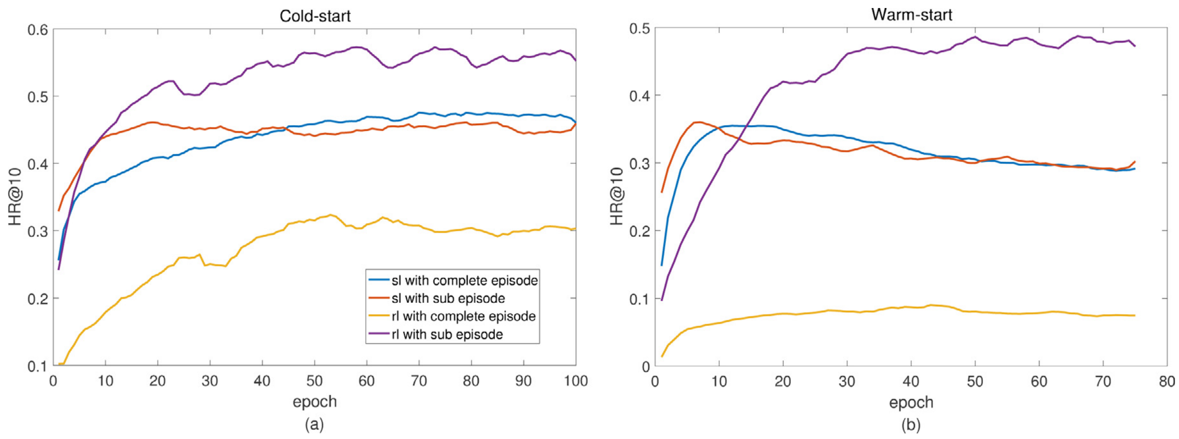

Fig. 11. Learning curves of different learning methods. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

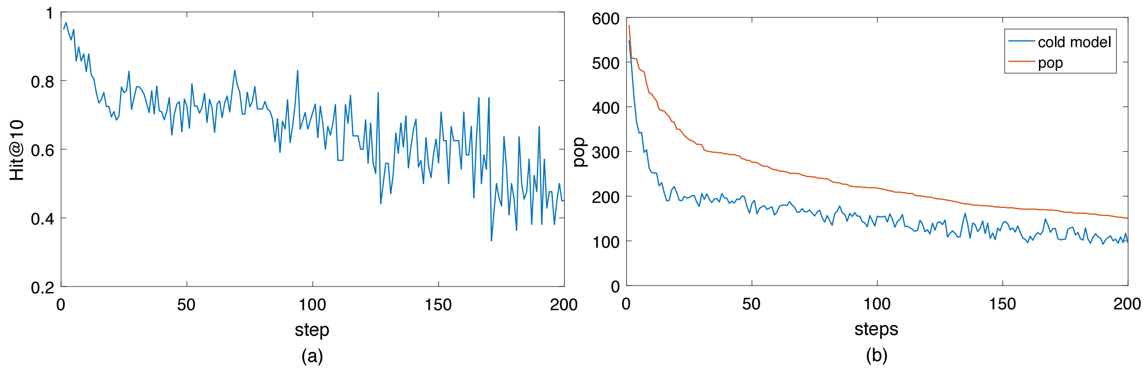

Fig. 12. Recommendation results for cold-start on Movielens 100K. (a):the average of *Hit@10* for each step t of test users; (b): the average of popularity of selected items for each step t of test users. Blue lines: the results on cold-start model. Orange line denotes the distribution of popularities of items, where the point at step t represents the popularity of the tth popular item. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

due to the total step size of 80% users for the cold-start scenario is less than 200.

As shown in Fig. 12(a), the results of *Hit@10* obtained by our agent are more than 50% in most cases. Especially in the range from 0 to about 120, the *Hit@10* results are more than 60%. Interestingly, the agent can achieve successful recommendations at the beginning by recommending the most popular items, which can help the agent quickly build an effective user profile. After more than about 120 steps, the results show clear fluctuations, which are mainly due to the adjustment of recommendations affected by the feedback. Therefore, there are no sustained declines in the results. On the other hand, as shown in Fig. 12(b), our cold-start model tends to recommend the most popular items (popularity above 200) in the early stage, but quickly begins to recommend personalized items (popularity below 200) after about 10 steps. Note that our agent does not simply choose the most popular items for the current step, as shown in the clear margin between the blue and orange curves in Fig. 12(b). In addition, the performance of our method can significantly outperform the one based on popularity. Similar behaviors can also be seen in the warm-start scenario, as shown in Fig. 13. Consequently, our agent will gradually evolve from a wide range of recommendations to personalized recommendations, and can accurately hit the items in the main range.

8. Conclusions

In this paper, we introduced a novel top-N deep reinforcement learning-based recommender system to explicitly tackle the long-term recommendation problem. In our proposed model, the processes of recommendations were viewed as MDP, and we employed RNN to simulate the sequential interactions between agent (recommender system) and environment (user). Especially, our model can be applied to both cold-start and warm-start scenarios. In addition, our model depended on the interactions between environment and agent instead of the content information, making it possible to be applied to environments without adequate content information. The experiment results also demonstrated the superior performance of our model in contrast with traditional top-N methods.

The differences between our model and the existing DRL methods for RS are as follows. (1) Most previous methods studied the simplified top-1 recommendation problem, whereas our work focuses on long-term top-N recommendation, which is closer to the practical recommendation scenarios. (2) Previous methods did not consider the differences between cold-start and warm-start recommendation scenarios. In this paper, two RNN-based models were designed for the cold-start and the warm-start recommendation scenarios, respectively. In both models, we incorporated a new feedback component into the input layer of RNN

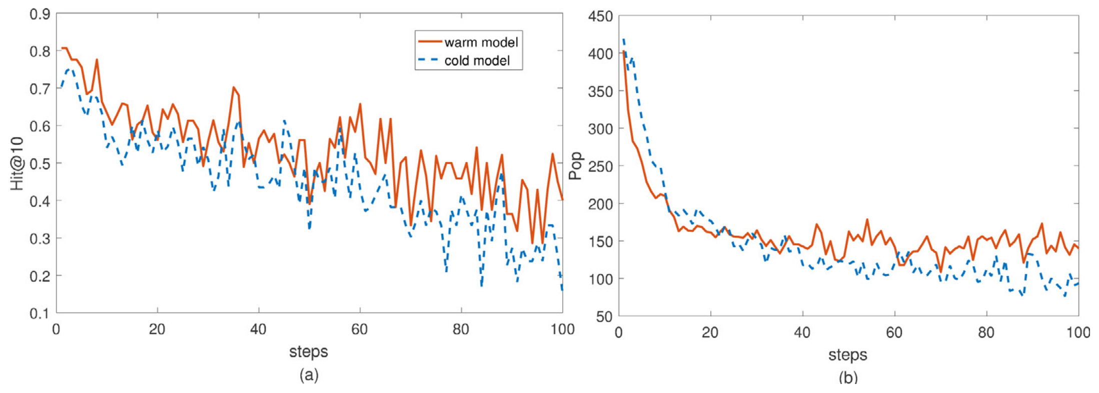

Fig. 13. Recommendation results for warm-start on Movielens 100K with $p\% = 50\%$.

to consider users' feedback effectively. Moreover, a new EMGRU was developed to further improve the recommendation accuracy in the warm-start scenario by utilizing users' historical items. (3) In previous methods, RL algorithms were used to optimize the recommendation accuracy. However, the essence of RL is interacting with the environment through trial and error. As a result, the learning processes might suffer from the sampling efficiency problem. Since recommendation systems have very large action space, the rewarded actions may not be encountered at the early stage of training. Different from the previous methods, our work utilized supervised learning with cross-entropy loss at the early stage of training to obtain better initial recommendation policies, and then adopted the on-policy reinforcement learning algorithm REINFORCE to optimize the long-term recommendation accuracy. Thanks to the similarities between cross-entropy-based supervised learning and REINFORCE, the adaption from supervised learning to reinforcement learning is natural and easy.

In the future, we plan to improve our model by employing more powerful Actor-Critic architecture, which has a lower variance at the cost of significant bias compared with the REINFORCE algorithm. Another direction is extending our algorithm by considering contextual information, which is widely used in IRS since sufficiently utilizing the content features can help construct more promising personal preferences for new users.

CRediT authorship contribution statement

Liwei Huang: Conceptualization, Methodology, Software, Visualization, Writing - original draft, Writing - review & editing. **Mingsheng Fu:** Conceptualization, Methodology, Software, Supervision, Funding acquisition, Writing - original draft, Writing - review & editing. **Fan Li:** Methodology, Visualization, Writing - review & editing. **Hong Qu:** Funding acquisition, Resources, Writing - review & editing. **Yangjun Liu:** Methodology, Writing - review & editing. **Wenyu Chen:** Writing - review & editing.

Declaration of competing interest

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

Acknowledgments

This work was supported by the China Postdoctoral Science Foundation under Grant 2020M673182, and National Science Foundation of China under Grant 61976043.

References

[1] F. Mi, B. Faltings, Adaptive sequential recommendation using context trees, in: IJCAI, 2016, pp. 4018–4019.
[2] G. Shani, D. Heckerman, R.I. Brafman, An MDP-based recommender system, J. Mach. Learn. Res. 6 (Sep) (2005) 1265–1295.
[3] X. Zhao, W. Zhang, J. Wang, Interactive collaborative filtering, in: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, 2013, pp. 1411–1420.
[4] J. Li, K. Lu, Z. Huang, H.T. Shen, Two birds one stone: on both cold-start and long-tail recommendation, in: Proceedings of the 25th ACM International Conference on Multimedia, 2017, pp. 898–906.
[5] R. Devooght, H. Bersini, Long and short-term recommendations with recurrent neural networks, in: Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization, 2017, pp. 13–21.
[6] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[7] Y. Ren, G. Li, J. Zhang, W. Zhou, Lazy collaborative filtering for data sets with missing values, IEEE Trans. Cybern. 43 (6) (2013) 1822–1834.
[8] B. Li, X. Zhu, R. Li, C. Zhang, Rating knowledge sharing in cross-domain collaborative filtering, IEEE Trans. Cybern. 45 (5) (2014) 1068–1082.
[9] L. Xu, C. Jiang, Y. Chen, Y. Ren, K.R. Liu, User participation in collaborative filtering-based recommendation systems: A game theoretic approach, IEEE Trans. Cybern. 49 (4) (2018) 1339–1352.
[10] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[11] X. Luo, M. Zhou, S. Li, Z. You, Y. Xia, Q. Zhu, A nonnegative latent factor model for large-scale sparse matrices in recommender systems via alternating direction method, IEEE Trans. Neural Netw. Learn. Syst. 27 (3) (2015) 579–592.
[12] A. Pujahari, D.S. Sisodia, Pair-wise preference relation based probabilistic matrix factorization for collaborative filtering in recommender system, Knowl.-Based Syst. (2020) 105798.
[13] S. Sedhain, A.K. Menon, S. Sanner, L. Xie, Autorec: Autoencoders meet collaborative filtering, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 111–112.
[14] Y. Zheng, B. Tang, W. Ding, H. Zhou, A neural autoregressive approach to collaborative filtering, in: Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 764–773.
[15] M. Fu, H. Qu, Z. Yi, L. Lu, Y. Liu, A novel deep learning-based collaborative filtering model for recommendation system, IEEE Trans. Cybern. 49 (3) (2018) 1084–1096.
[16] M. Fu, H. Qu, D. Moges, L. Lu, Attention based collaborative filtering, Neurocomputing 311 (2018) 88–98.
[17] R. Yin, K. Li, G. Zhang, J. Lu, A deeper graph neural network for recommender systems, Knowl.-Based Syst. 185 (2019) 105020.
[18] S.M. Daneshmand, A. Javari, S.E. Abtahi, M. Jalili, A time-aware recommender system based on dependency network of items, Comput. J. 58 (9) (2015) 1955–1966.
[19] Y. Ding, X. Li, Time weight collaborative filtering, in: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 2005, pp. 485–492.
[20] Y. Koren, Collaborative filtering with temporal dynamics, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 447–456.
[21] J.Z. Sun, K.R. Varshney, K. Subbian, Dynamic matrix factorization: A state space approach, in: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2012, pp. 1897–1900.

[22] D. Yang, T. Chen, W. Zhang, Y. Yu, Collaborative filtering with short term preferences mining, in: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2012, pp. 1043–1044.

[23] W. Hong, L. Li, T. Li, Product recommendation with temporal dynamics, Expert Syst. Appl. 39 (16) (2012) 12398–12406.

[24] A. Javari, M. Jalili, Accurate and novel recommendations: an algorithm based on popularity forecasting, ACM Trans. Intell. Syst. Technol. (TIST) 5 (4) (2014) 1–20.

[25] F. Rezaeimehr, P. Moradi, S. Ahmadian, N.N. Qader, M. Jalili, TCARS: Time- and community-aware recommendation system, Future Gener. Comput. Syst. 78 (2018) 419–429.

[26] I. Al-Hadi, N.M. Sharef, M.N. Sulaiman, N. Mustapha, Review of the temporal recommendation system with matrix factorization, Int. J. Innov. Comput. Inf. Control 13 (5) (2017) 1579–1594.

[27] P. Massa, P. Avesani, Trust-aware recommender systems, in: Proceedings of the 2007 ACM Conference on Recommender Systems, 2007, pp. 17–24.

[28] G. Guo, J. Zhang, N. Yorke-Smith, Leveraging multiviews of trust and similarity to enhance clustering-based recommender systems, Knowl.-Based Syst. 74 (2015) 14–27.

[29] M.M. Azadjalal, P. Moradi, A. Abdollahpouri, M. Jalili, A trust-aware recommendation method based on Pareto dominance and confidence concepts, Knowl.-Based Syst. 116 (2017) 130–143.

[30] I. Guy, Social recommender systems, in: Recommender Systems Handbook, Springer, 2015, pp. 511–543.

[31] T. Guo, J. Luo, K. Dong, M. Yang, Differentially private graph-link analysis based social recommendation, Inform. Sci. 463 (2018) 214–226.

[32] S. Ahmadian, N. Joorabloo, M. Jalili, Y. Ren, M. Meghdadi, M. Afsharchi, A social recommender system based on reliable implicit relationships, Knowl.-Based Syst. 192 (2020) 105371.

[33] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, in: International Conference on Learning Representations, 2016.

[34] Y.K. Tan, X. Xu, Y. Liu, Improved recurrent neural networks for session-based recommendations, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, 2016, pp. 17–22.

[35] S. Wu, W. Ren, C. Yu, G. Chen, D. Zhang, J. Zhu, Personal recommendation using deep recurrent neural networks in netease, in: 2016 IEEE 32nd International Conference on Data Engineering, ICDE, IEEE, 2016, pp. 1218–1229.

[36] B. Hidasi, M. Quadrana, A. Karatzoglou, D. Tikk, Parallel recurrent neural network architectures for feature-rich session-based recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 241–248.

[37] E. Smirnova, F. Vasile, Contextual sequence modeling for recommendation with recurrent neural networks, in: Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems, 2017, pp. 2–9.

[38] L. Li, W. Chu, J. Langford, R.E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Proceedings of the 19th International Conference on World Wide Web, 2010, pp. 661–670.

[39] J. Kawale, H.H. Bui, B. Kveton, L. Tran-Thanh, S. Chawla, Efficient thompson sampling for online for matrix-factorization recommendation, in: Advances in Neural Information Processing Systems, 2015, pp. 1297–1305.

[40] C. Gentile, S. Li, P. Kar, A. Karatzoglou, G. Zappella, E. Etrue, On context-dependent clustering of bandits, in: Proceedings of the 34th International Conference on Machine Learning, Vol. 70, JMLR. org, 2017, pp. 1253–1262.

[41] H. Zhuang, C. Wang, Y. Wang, Identifying outlier arms in multi-armed bandit, in: Advances in Neural Information Processing Systems, 2017, pp. 5204–5213.

[42] A. Rakhlin, K. Sridharan, BISTRO: An efficient relaxation-based method for contextual bandits, in: ICML, 2016, pp. 1977–1985.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[44] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.

[45] S.-Y. Chen, Y. Yu, Q. Da, J. Tan, H.-K. Huang, H.-H. Tang, Stabilizing reinforcement learning in dynamic environment with application to online recommendation, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1187–1196.

[46] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N.J. Yuan, X. Xie, Z. Li, DRN: A deep reinforcement learning framework for news recommendation, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 167–176.

[47] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, D. Yin, Recommendations with negative feedback via pairwise deep reinforcement learning, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1040–1048.

[48] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, B. Coppin, Deep reinforcement learning in large discrete action spaces, 2015, arXiv preprint arXiv:1512.07679.

[49] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, J. Tang, Deep reinforcement learning for page-wise recommendations, in: Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 95–103.

[50] D. Shin, How do users interact with algorithm recommender systems? The interaction of users, algorithms, and performance, Comput. Hum. Behav. (2020) 106344.

[51] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 173–182.

[52] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, in: NIPS Deep Learning Workshop, 2014.

[53] M. Jalili, S. Ahmadian, M. Izadi, P. Moradi, M. Salehi, Evaluating collaborative filtering recommender algorithms: a survey, IEEE Access 6 (2018) 74003–74024.

[54] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: ICLR, 2015.

[55] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, BPR: Bayesian personalized ranking from implicit feedback, in: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2009, pp. 452–461.

[56] W.-C. Kang, J. McAuley, Self-attentive sequential recommendation, in: 2018 IEEE International Conference on Data Mining, ICDM, IEEE, 2018, pp. 197–206.

[57] C. Ma, P. Kang, X. Liu, Hierarchical gating networks for sequential recommendation, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 825–833.