Article

Efficient Tree Policy with Attention-Based State Representation for Interactive Recommendation

Longxiang Shi [1], Qi Zhang [2], Shoujin Wang [3], Zilin Zhang [4], Binbin Zhou [1], Minghui Wu [1,*] and Shijian Li [4]

1. College of Computer and Computing Science, Hangzhou City University, Hangzhou 310015, China; shilx@zucc.edu.cn (L.S.); bbzhou@zucc.edu.cn (B.Z.)
2. DeepBlue Academy of Sciences, Shanghai 200336, China; zhangqi_cs@bit.edu.cn
3. Data Science Institute, University of Technology Sydney, Sydney, NSW 2007, Australia; shoujin.wang@uts.edu.au
4. College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China; zilinzhang@zju.edu.cn (Z.Z.); shijianli@zju.edu.cn (S.L.)
* Correspondence: mhwu@zucc.edu.cn

Abstract: Nowadays, interactive recommendation systems (IRS) play a significant role in our daily life. Recently, reinforcement learning has shown great potential in solving challenging tasks in IRS, since it can focus on long-term profit and can capture the dynamic preference of users. However, existing RL methods for IRS have two typical deficiencies. First, most state representation models use left-to-right recurrent neural networks to capture the user dynamics, which usually fail to handle the long and noisy sequential data in real life. Second, an IRS always needs to handle millions of items, leading to a large discrete action space in RL settings, which has not been fully addressed by the inefficient existing works. To bridge these deficiencies, in this paper, we propose attention-based tree recommendation (ATRec), an efficient tree-structured policy with attention-based state representation for IRS. ATRec uses an attention-based state representation model to effectively capture the user's dynamic preference hidden in the long and noisy sequence of behaviors. Moreover, to improve the learning efficiency, we propose an efficient tree-structured policy representation method, in which a complete tree is devised to represent the policy, and a novel parameter-sharing strategy is introduced. Extensive experiments are conducted on three real-world datasets and the results show the proposed ATRec obtains 42.3% improvement over some of the state of the arts methods in the hit rate and 21.4% improvement in the mean reciprocal rank of the top 30 ranked items. Additionally, the learning and decision efficiency can also be improved at an average of 35.5%.

Keywords: reinforcement learning; deep learning; interactive recommendation system

1. Introduction

With the prosperous development of information technology, nowadays, interactive recommender systems (IRS) play an important role in personalized services to help us discover information and satisfy our needs. TikTok's personalized video recommendations (http:www.tiktok.com, accessed on 10 April 2023), Spotify's music recommendation (http:www.spotify.com, accessed on 10 April 2023), and Amazon's product recommendation (http:www.amazon.com, accessed on 10 April 2023) are a few examples of how they have become an indispensable part in our daily lives [1–3]. Different from conventional recommendation systems, an IRS suggests items based on user behavior and consecutively refines the recommendations based on users' feedback [4,5]. Due to the interactive nature of such recommendation settings, IRS should be able to capture the dynamic preference of users' behavior and perform planning to optimize long-term performance. Conventional recommendation methods such as matrix factorization, content-based filtering, and learning-to-rank regard the recommendation process as a static one and fail to capture the

dynamic preferences of users. Furthermore, most of the conventional methods recommend items that can achieve immediate satisfaction of users, while ignoring those items that may lead to more profitable rewards in the future [6–8].

Recently, reinforcement learning (RL) [9] has shown great potential in modeling dynamic interaction behaviors and pursuing long-term rewards [10,11]. Naturally, as a promising method, RL has been introduced into IRS to solve the aforementioned particular challenges [12–15]. Typically, an RL-based IRS provides recommendations to users in an interactive way: each time, the IRS recommends a series of items to the user and the user browses these items and provides feedback; then, IRS refines the recommendation strategy and suggests new items depending on the feedback. The state modeling, which is frequently based on the users' historical behaviors, is an important part of the RL agent. Existing state representation works for IRS generally adopted unidirectional recurrent neural networks (RNN) to model the states, and encoded the states with the hidden representation of the RNN model. Such works can be found in [5,6,13,16,17]. However, real-world data are usually noisy and do not rigidly follow left-to-right orders [18,19]. The long sequence of user historical records always contains items that are irrelevant to users' future choices, which refers to noisy patterns [20,21]. For example, consider the user's action sequence in Figure 1. Based on their historical choices, we can infer that they are probably a science fiction fan. Besides, they may also watch some movies that are newly released or suggested by friends, as depicted in timesteps 4 and 5. In the IRS setting, the RL agent obtains the user ratings 4 and 5 at timesteps 4 and 5 with the two irrelevant items, respectively. The unidirectional RNN state representation model may be confused by the two ratings and may ignore the fact that they are a science fiction fan. This situation may be aggravated with the increase of the sequence length, as the noisy patterns vary. Some works adopted attention-based methods [4,22] to address this issue. For example, DRR-att [4] adopts an attentive network to deal with noisy dependencies. However, DRR-att directly inputs the item vector to the attentive networks and ignores the feedback from users, which makes it difficult for dealing with long sequence data. Therefore, it is essential to design a state encoder that can handle the long and noisy sequence in real-world IRS.
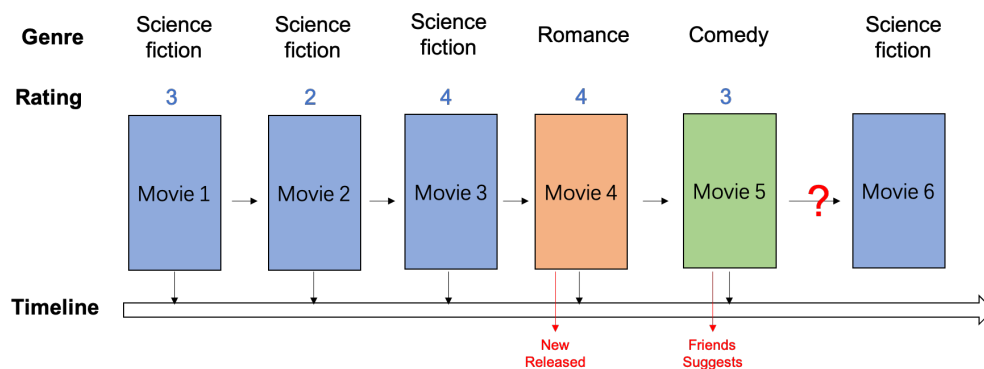

Figure 1. A motivation example.

Furthermore, adopting RL methods for a recommendation scenario with vast discrete action space is challenging, because the IRS often contains millions of items. Factoring the large discrete action space into a considerably smaller one is an obvious solution. For example, DDPG-KNN [23] mapped the discrete action space to a low-dimensional continuous action space and selected actions based on their similarities. Such works are inefficient as the computation of action similarities is time-consuming. An alternative strategy is to design a specific neural network to facilitate learning the tasks in large action spaces. For instance, DRN [24] and DEERS [6] adopt a refined Deep Q-Networks (DQN) [25] to learn policies with large action spaces. However, learning the DQN-based methods always involves maximum operation over the actions, which is inefficient when the number of actions is large. Recently, TPGR (tree-structured policy gradient for recommendation) [5] was proposed, which organized the policy into a tree structure and is efficient in both

learning and decision making. However, TPGR becomes inefficient during evaluation, especially when calculating the top-K ranked items, which is serious in recommender systems since the computation of each action choice needs to traverse the tree.

To improve the ability and efficiency for dealing with long and noisy dependencies that occur in real-world applications, in this paper, an attention-based tree-policy recommendation (ATRec) method is proposed, which can effectively and efficiently capture user dynamics under long and noisy patterns. In addition, an efficient tree-structured policy network is devised, which can further improve the efficiency of TPGR with a refined tree policy model. Specifically, a complete tree to represent the policy and incorporate a parameter-sharing strategy is used to improve both the learning and evaluation efficiency. The proposed method is evaluated on three well-known benchmark datasets to illustrate the effectiveness and efficiency of the proposed methods. To summarize, the contributions of this paper are as follows:

- An attention-based state representation model for IRS is proposed that can effectively capture the user dynamics even when the sequential data are long and noisy.
- An efficient tree-structured policy is devised that can improve the learning and decision efficiency of TPGR through reinforcement learning.
- The proposed model is evaluated with state-of-the-art methods and the results demonstrate that the proposed method is effective and efficient on three benchmark datasets.

2. Literature Review

2.1. State Representation Modeling for a Deep-Reinforcement-Learning-Based Interactive Recommendation

State representation modeling for deep RL-based IRS is crucial to model the user behavior during the recommendation process. Therefore, many previous studies have investigated this topic in the past decades. Among the state-of-the-art methods, sequential models are the most popular adopted. Models such as recurrent neural networks (RNN), and convolutional neural networks (CNN) are widely used in modeling the sequential dependencies over the user–item interactions in sequential recommender systems and IRS [26]. For instance, long short-term memory (LSTM) [13,27,28] and gated recurrent unit (GRU) [6,16] have been applied in IRS to capture the long-term dependencies over user interactions in IRS. In [5], simple recurrent unit (SRU) [29] was used to improve the learning efficiency in IRS. In [30], CNN was also introduced to IRS to model user behaviors. Despite the effectiveness of the above methods, most of these methods used left-to-right sequential models to model the sequential dependencies. They encoded each user's current state based on the hidden representation learned by the sequential models. However, the real-world interaction data do not often follow the left-to-right rigid order, as indicated in a sequence [26,31]. Therefore, the aforementioned methods may have limitations in dealing with real-world data containing long and noisy dependencies [32].

The adoption of attention-based models for IRS also can be found in various studies, including [4,22]. In [22], they used attentive recurrent neural networks to capture the order over the historical items. Moreover, in [33], a multi-head attention network with GRU models the state representations. Recently, some advanced models have been proposed for modeling the states of RL-based interactive recommendation. Embracing the knowledge graph (KG) with sequential models evolves knowledge-based state representation models [34]. Such works can be found in [14,35–38]. Unfortunately, knowledge-based state representation needs extra knowledge to interpret the states, which is not always available in some interactive recommendation scenarios.

2.2. Dealing with Large Discrete Action Space in Reinforcement Learning

Most RL methods become less effective when facing tasks with large discrete action spaces, as the search space of finding the optimal policies grows exponentially with the increase of actions. To address this problem, one common strategy is to reduce the large action spaces into a considerable one. For example, Sallans et al. [39] proposed a method

that can first factorize the action space as negative free energies and then adopt an ensemble method to learn the policy. Similarly, Pazis et al. [40] proposed a method that represented each action in binary format and optimized a value function associated with each bit. Bellemare et al. [41] showed that the performance of RL method can be improved when the action spaces are pre-categorized. Another popular method is to first map the large discrete action space to a smaller continuous action space and then select the discrete actions based on the similarity between continuous action spaces, such as DDPG-KNN [23] and Calca [16]. However, those methods are inefficient when mapping the actions, as the nearest neighbor search may be time costly. More recently, Chandak et al. [42] proposed a method that can learn the action representations during RL process, which can factorize the action space into a solvable level. Although this method is effective and scalable for solving tasks with large discrete action spaces, training such a model is inefficient, as it involves integral calculating over the probability distribution of actions. Fu et al. [43] proposed DHCRS, a hierarchical RL method that adopted two DQN agents for addressing the large action space problem. DHCRS divided the items into categories and the high-level DQN agent selects a category and the low-level DQN agent recommends a certain item within this category. However, in practice, the amount of items in each category is not always equal, and adopting the same DQN agent for different categories may affect the overall efficiency in learning.

Alternatively, designing specific neural networks to facilitate learning the tasks in large action space is another popular strategy. For instance, DRN [24] and DEERS [6] adopt a refined DQN [25] network to learn the policies with large action spaces. However, learning for DQN-based solutions involves a maximum operation among the actions, whose time complexity grows linearly with the number of actions. Moreover, Zhao et al. [16] adopted a deconvolution layer that maps the action space into a matrix. Chen et al. [5] proposed TPGR, which uses a balanced tree-structured policy to simplify the policies with large action space, and thus, improves the efficiency in both learning and decision making. However, TPGR is less efficient when learning online. The parameters in TPGR policy networks grow exponentially with the depth of the tree. Backpropagating such a number of parameters is computationally costly when the depth of the tree-structured policy is large [44].

In addition, when making decisions, TPGR needs to calculate the product of the probability of making each choice via each layer. This mechanism makes it inefficient when getting the top-K ranking actions, which is very common in IRS.

3. Method

In this section, we first introduce the problem statement of RL-based interactive recommendation, and then demonstrate our proposed ATRec method in detail.

3.1. Problem Statement

In this work, we consider the recommendation task in which a recommender agent interacts with an environment (i.e., users). During the interaction process, the environment sequentially selects the items recommended by the recommender so as to maximize the cumulative reward of the recommender agent. Usually, the reward of the environment is set to the score that related to users' feedbacks, such as the rating for the corresponding item. Consequently, we model the recommendation process as an Markov decision process (MDP), whose key components are $< S, A, P, R, \gamma >$. The details of each component are defined as follows:

- State S: A state $s \in S$ is defined as the historical interactions between the user and the recommender agent.
- Action A: An action $a \in A$ is the suggested items provided by the recommender agent.
- Transition probability function P: $P(s'|s, a)$ is the transition function that determines the new state s' after the recommender agent suggests item a under observation s, which models the dynamics of user preference.

- Reward function R: $R(s, a)$ is a function that calculates the immediate reward received by the recommender agent after the user provides feedback when given the recommendation items a under state s.
- Discount factor γ: $\gamma \in [0, 1]$ defines the measurement of present value of long-term rewards.

In the recommendation process, the recommender agent interacts with a user by providing recommended items based on the user's state and then receives an immediate reward that indicates the user's feedback. Our goal is to find a recommendation policy $\pi : S \rightarrow A$ that can maximize the cumulative reward for the recommender agent.

3.2. Attention-Based State Representation Model

In this section, we describe the proposed attention-based state representation model in detail.

The proposed state representation model consists of an input layer, a feature embedding layer, an attention layer, and an output layer, as shown in Figure 2. The input layer receives the feature vectors from items and the users' feedback, and then preprocessed them by a fully connected layer. The attention layer calculates the weights of each corresponding item. Finally, the output layer concatenates the user feature and the output context embedding vector of the attention layer. In the input layer, the historical N interactions, consisting of the recommended items and their corresponding rewards along with the user ID are collected. Assuming that the recommender agent is performing t-th recommendation, then N previous item–reward pairs (from $t - N - 1$ to $t - 1$) are used to encode the state. The item ID and user ID are mapped to a latent feature vector via matrix factorization. The feature vectors are denoted as the item feature and user feature, respectively, which are fixed during learning. The user feedback on the corresponding recommended items, such as ratings, is also added to the item feature. In the feature embedding layer, the input user features and item features are encoded by a fully connected layer to obtain the feature embeddings.
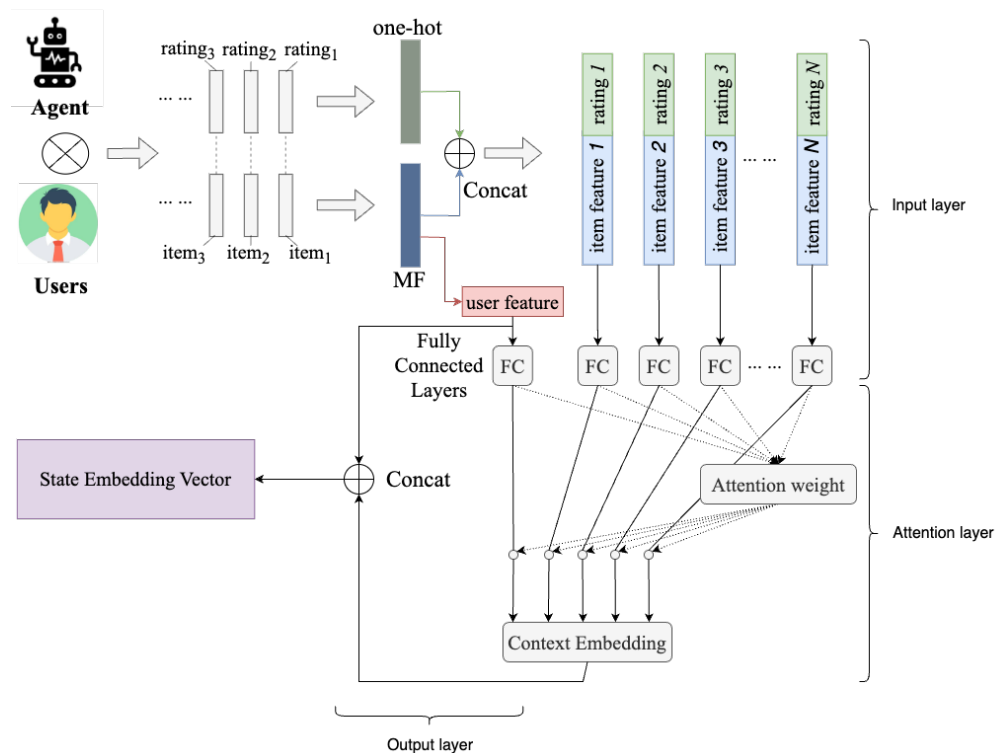

Figure 2. Attention-based state representation model.

In order to capture the dynamics for different items in long and noisy sequential data, we adopt an attention layer to discriminatively capture the contributions of each item together with the user features. The attention layer aims at learning the integration weights for both item and user features. Denoting h_i as the ith preprocessed item feature and h_{user} as the preprocessed user feature, the attentive context embedding is calculated as follows:

$$e = \sum_{i=1}^{N} (\alpha_{ti} h_i) + \alpha_{user} h_{user},$$
$$s.t. \sum_{i=1}^{N} (\alpha_{ti}) + \alpha_{user} = 1 \tag{1}$$

where α_{ti} is the integration weight of the ith item embedding vector with respect to the tth target item and α_{user} is the integration weight for a specific user.

The attention weights are calculated through a Softmax layer as below:

$$\alpha_{ti} = \frac{\exp(f(h_i))}{\sum_{i=1}^{N} \exp(f(h_i^F)) + \exp(f(h_{user}))}$$
$$\alpha_{user} = \frac{\exp(f(h_{user}))}{\sum_{i=1}^{N} \exp(f(h_i^F)) + \exp(f(h_{user}))} \tag{2}$$
$$f(h_i) = ReLU(W^\alpha h_j^T + b^\alpha) \ \forall h_j \in \{h_i, h_{user}\}$$

where W^α and b^α are the weight and bias parameters for the attention layer, respectively.

Finally, in the output layer, we concatenate the input user feature and the attentive context embedding as the state embedding vector. The concatenate symbol is represented as $\oplus$ in Figure 2.

Although the state representation model DRR-att in [4] looks similar to our proposed model, compared to DRR-att, our proposed state representation model involves users' feedback as the input of items. In addition, we also preprocess the input features by a fully connected layer and simplify the integration of user feature and item features. Our experimental results show that our proposed method outperforms DRR-att in our benchmark datasets.

3.3. Efficient Tree-Structured Policy

Most of the value-based RL methods are not efficient, since they involve maximization over the action spaces, which are often very large. In contrast, policy gradient methods are more efficient in solving IRS tasks. The reason is that, in policy gradient methods, policies are represented with the state as the input and the action probabilities as the output. Unfortunately, learning such a policy network with a large number of output dimensions is always time costly since the softmax output layer requires explicit normalization over the actions [5]. Representing the output layer of the policy network with the tree-structured network can substantially reduce the computation cost in learning and evaluation, such as hierarchical softmax [44] and TPGR [5].

For TPGR, the policy network is represented by a tree-structured neural network, with each leaf node representing the output probability of the corresponding item and each none-leaf node outputs a probability for selecting the node in the below layer. The output probability of each item is calculated by multiplying the probabilities on the traversing path. One major drawback is that the computation of the probability to recommend each candidate item requires specifying the path from the root node to the corresponding leaf in the tree. Traversing the tree always needs a loop to determine the nodes in each layer, which is quite inefficient in practice. Specifically, let us first take a look at the tree structure of policy in TPGR. Denoting M as the total item number and c as the child number for each node, then based on the balanced clustering algorithm, TPGR obtains a tree-structured policy with items arranged into $floor(M/c)$ clusters. Here, $floor(x)$ returns the largest

integer which is no less than x. Among those clusters, we have (M mod c) clusters that contains $c + 1$ items, and ($floor(M/c) - M$ mod c) clusters that contains c items. Since the item number for each cluster denotes the output dimension for each node in the final layer of the tree policy, such distribution of items in the leaf node requires layer-to-layer traversing of the tree. As a consequence, indexing each item needs $O(\log M)$ decision time. For IRS, calculating the top-K ranked items for both evaluation and recommendation is very common in practice. In this scenario, TPGR needs $O(M \log M)$ decision time to obtain all the action probabilities and, finally, obtain the top-K ranked items. Such a computation method for calculating recommendation probability for candidate items lacks efficiency and is hard to parallelize. Therefore, deploying TPGR to real-world IRS is challenging.

To address this challenging issue, we propose a novel complete tree to better represent the structure of the policy, as illustrated in Figure 3. In this framework, each non-leaf node of the policy tree receives the item feature list as input and contains two parts: the state representation model and a node policy network. The output of the leaf node denotes the recommendation probabilities of the corresponding items and is computed by traversing the tree from the root to this leaf. For each leaf node, clustering over items can also be used to assign similar items to one node, and thus, simplify learning. We will describe the clustering algorithm for the complete tree policy in the Appendix A.The property of a complete tree makes it easy to index each root node of the decision path. If we store each node of the tree policy in an array, given an item index I, the indices of the tree node from bottom to top can be obtained as $< I/c, I/c^2, I/c^3, \ldots, 1 >$. Therefore, we can then compute all the item recommendation probabilities based on a few simple matrix computations. Algorithm 1 illustrates the computation of item recommendation probability based on the complete tree policy. Based on this algorithm, we can then obtain all the item recommendation probabilities in $O(\log M)$ decision time. In addition, we can also construct the computational graphs for the complete tree policy with popular deep learning toolkits such as Tensorflow (http:www.tensorflow.org, accessed on 10 April 2023) and Pytorch (http://pytorch.org, accessed on 10 April 2023), which makes it easy for parallel computing.
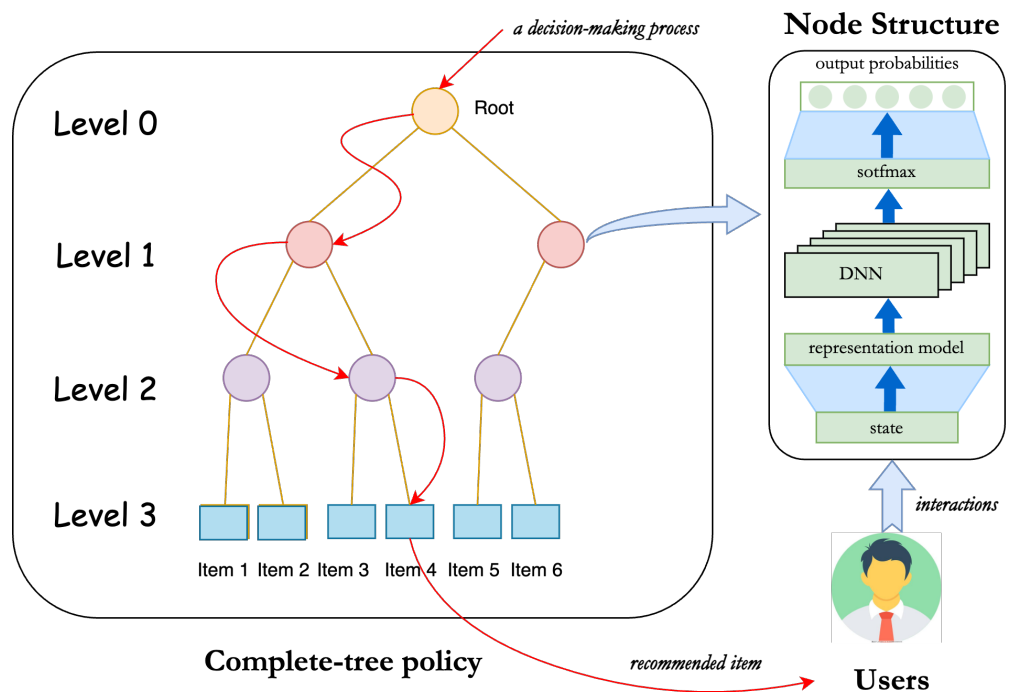

Figure 3. Illustration of the complete tree policy.

Moreover, during the computation of recommendation probability, we obtain a cumulative product of the tree node, which contains a series of state representation models. The backpropagation operation in such a network is time consuming, as it involves too many parameters. Hence, we design a parameter-sharing strategy to reduce the scale of the number of parameters and further improve the learning efficiency of the tree policy. For each node, the parameters of the state representation model can be shared through two levels:

- All-shared: The parameters of the state representation model are shared across all the tree nodes.
- Layer-shared: The parameters of the state representation model are shared across different layers; that is, for one layer of the tree, the nodes hold the same state representation model.

Note that the parameter-sharing strategy is not applied to each policy network, in order to keep the representation probabilities of each node. In the next section, we will illustrate the learning of the complete tree policy.

Algorithm 1 Decision making based on the complete tree policy

---

Input: Input item feature list $< u_1, u_2, \ldots, u_N >$, node of the tree policy $\pi_0, \pi_1, \ldots, \pi_n$, tree depth d, child number c.
Output: The probabilities of the top-M corresponding items.
1: $root\_output = \pi_0(u_1, u_2, \ldots, u_N)$
2: $k = 1$
3: **for** $i = 1$ to d **do**
4:    $output\_list = \varnothing$
5:    **for** $j = 1$ to c^i **do**
6:       $node\_output = root\_output \circ \pi_k(u_1, u_2, \ldots, u_N)$ {Compute the element-wise product between the two nodes' outputs}
7:       Add $node\_output$ to $output\_list$
8:       $k = k + 1$
9:    **end for**
10:   $root\_output = concat(output\_list)$ {Concatenate all the outputs for each layer}
11: **end for**
12: **return** The first M elements of $root\_output$

3.4. Learning Process

The learning process of the proposed method is illustrated in Figure 4. Firstly, we use the historical user–item matrix to extract the features for each item and users. Specifically, we use Funk SVD [45] to decompose the original user-rating matrix into an item matrix and a user matrix. The item matrix and user matrix are used as item features and user features, respectively. Then clustering is performed to separate the items into subclasses, in order to simplify the learning of the tree policy. As we use the complete tree to represent the recommender policy, the clustering tree can be constructed by applying a hierarchical clustering algorithm that can associate the leaf node with one item. We adopt a K-means-based clustering algorithm to build the clustering tree, which is depicted in Algorithm 2. Based on the clustering result, the complete tree policy is constructed, with each leaf node corresponding to a certain item id. Recommendations are provided to the user by mapping the output action of the policy to the items. During the interaction with the user, the tree-structured policy is trained through a policy gradient method.
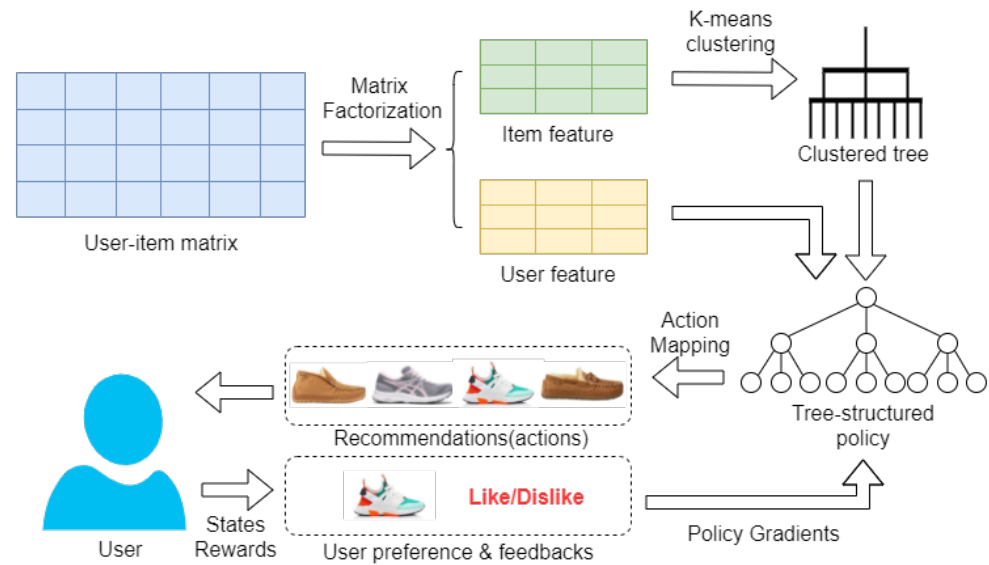
Figure 4. Learning process of the tree policy.

Algorithm 2 K-means clustering algorithm for building the complete tree policy

---

Input: a group of vectors $v_1, v_2, \ldots, v_m$ and the child number of the complete tree c.
Output: The clusters for each leaf node of the tree.
1: **Initialize:** mark all the input vectors as unassigned.
2: Use K-means algorithm to find c centroids: $p_1, p_2, \ldots, p_c$.
3: **for** $i = 1$ to $c - 1$ **do**
4:    Find c nearest vectors $t_1, t_2, \ldots, t_c$ to p_i among unassigned vectors based on Euclid distance.
5:    Assign $t_1, t_2, \ldots, t_c$ to the ith cluster.
6:    Mark $t_1, t_2, \ldots, t_c$ as assigned.
7: **end for**
8: Assign the unmarked vector to the cth cluster.
9: **Return:** All c clusters.

The learning of the tree policy can utilize any policy gradient method; here, we use the REINFORCE [46] algorithm to illustrate the learning process. Denoting the overall tree policy as π_θ, since the learning objective is to maximize the expected discounted total rewards, the loss function can be written as:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{3}$$

Based on policy gradient theorem, the gradient with respect to parameter θ can be written as:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)\right] \tag{4}$$

where $\pi(a|s)$ is the probability of taking action a at state s and Q^{π_θ} is the expected discount rewards after taking action a under state s. During learning, Q^{π_θ} can be obtained by sampling the trajectories under policy π_θ from either historical user data or online data. The whole learning algorithm is depicted in Algorithm 3.

Algorithm 3 Learning the complete tree policy

Input: Complete tree policy π_θ, learning rate $\alpha > 0$, discount factor γ.
Output: The learned policy π_θ.
  1: **Initialize:** Policy parameter θ.
  2: **repeat**
  3:    Sample an episode $\{(s_1, a_1, r_1, s_2), \ldots, (s_n, a_n, t_n, r_{n+1})\}$ from historical user data or online data.
  4:    **for** $t = 1$ to n **do**
  5:        $Q^{\pi_\theta}(s_t, s_t) = \sum_{i=t}^{n} \gamma^{i-t} r_i$
  6:        Calculate $\pi_\theta(a_t|s_t)$ based on Algorithm 1.
  7:        $\theta \to \theta + \alpha \gamma^t Q^{\pi_\theta}(s_t, s_t) \nabla_\theta \log \pi_\theta(a|s)$
  8:    **end for**
  9: **until** Converge
10: **Return:** π_θ

---

4. Results and Discussion

In this section, the empirical study of the proposed method is given. Specifically, we first describe the setup of the experiments by preparing the experimental datasets and introducing the baseline methods and then evaluating the performance and efficiency of the proposed method. We intend to answer the following four research questions (RQ) through experiments:

- RQ1: How does the proposed method perform compared with the state-of-the-art interactive recommendation methods?
- RQ2: Does our method improve learning efficiency?
- RQ3: Does the proposed state representation method improve the performance over the state-of-the-art methods?
- RQ4: How do the different parameter-sharing strategies affect the performance of our model?

4.1. Experimental Setting

4.1.1. Datasets

We conduct experiments on three representative real-world benchmark datasets: Instant Video, Baby, and Musical Instruments, which are commonly used for testing the performance of IRSs. These datasets contain product reviews from Amazon (https://jmcauley.ucsd.edu/data/amazon/, accessed on 10 April 2023) [47,48]. The ratings for each dataset are ranging from 0 to 5. Specifically, we use a quarter of each dataset for evaluation. Table 1 lists the statistics of the three datasets. For each dataset, we use 80% of the data for training and the other 20% of data for testing.

Table 1. Summary of the statistics of the datasets.

Dataset	Instant Video	Baby	Musical Instruments
Number of Users	122,609	175,826	98,959
Number of Items	8229	8256	17,380
Number of Ratings	145,983	228,861	125,044
Number of Users in training set	98,084	140,660	79,167
Number of Users in testing set	24,522	35,166	19,792

Due to the interactive nature of IRS, an ideal way to conduct experiments is to directly interact with real users. However, online experiments may be too expensive and vulnerable to commercial risks for the IRS itself [5,14]. Following some of the existing works [1,4], we use an offline environment simulator based on offline datasets to conduct the experiments. At each timestep, the environment simulator provides the historical items and ratings of a user and provides feedback after the recommender system suggests items. The reward

function of the environment is set to normalize the ratings of the user, which linearly normalizes the rating to $[-1, 1]$:

$$R(s, a) = -1 + 2 * \frac{r_{ij}}{5} \tag{5}$$

where r_{ij} is the rating for user *i* for item *j*.

4.1.2. Evaluation Metrics

We report three commonly used evaluation metrics [49,50] in the experiments:

- Average Reward: Since IRS aims to maximize the total reward of the episode, the average reward is a straightforward performance measurement. We adopt the reward over the top-K suggested items. If the top-K suggested items contain the item that the user selects, then the reward is set to the ratings commented on by the user. Otherwise, the reward over top-K suggested items is set to 0.
- Hit Ratio *HR@K*: HR measures the fraction of items that the user favors in the recommendation list and is calculated as below:

$$HR@K = \frac{1}{\# \, Users * K} \sum_{\# \, Users} \sum_{i=1}^{K} \theta_{hit} \tag{6}$$

where we define $\theta_{hit} = 1$ if the item user selects and favors ($r_{ij} > 3.5$) is in the top-K suggested items.

- Mean Reciprocal Rank *MRR@K*: *MRR@K* measures the average reciprocal rank of the first relevant item. Denoting k_i as the rank of the first relevant recommendation item, *MRR@K* is calculated as below:

$$MRR@K = \frac{1}{\# \, Users * K} \sum_{\# \, Users} \sum_{i=1}^{K} \frac{1}{k_i} \tag{7}$$

4.1.3. Compared Methods

We compare our method with state-of-the-art IRS methods of different types, as listed below:

- Popularity: Ranks the top k frequent items according to their popularity measured by a number of ratings, a simple but widely adopted baseline method.
- SVD: Suggests recommendations based on singular value decomposition (SVD). For the IRS setting, the model is trained after each user interaction and gives recommendations with the predicted highest rating.
- DDPG-KNN: A DDPG-based method that maps the discrete action space to a continuous one, then selects *K* nearest items in the continuous space with the max Q-value obtained by the critic network [23]. In our experiment, the *K* value is set to $\{1, 0.1N\}$.
- DQN-R: A DQN-based method that adopts a refined DQN to evaluate the Q-values of the items and chooses the item with the max Q-value [24].
- TPGR: Adopts a tree-structured policy and uses the policy gradient to optimize the tree-structured policy [5]. This is the state-of-the-art IRS approach and is similar to our proposed method.

For the deep RL methods DDPG-KNN, DQN-R, and TPGR, we use Tensorflow version 1.4 for implementation. We will open-source our code after acceptance. The experimental details can also be found in Appendix A.

4.2. Performance Evaluation (RQ1)

We investigate the recommendation performance of ATRec against the state-of-the-art baselines with respect to average reward@K, *HR@K*, and *MRR@K*. In this part, we fixed the length of each episode to 32. The experimental results are summarized in Table 2, where the best result in each row is highlighted in bold. The proposed method performs clearly better than the comparison baselines on the three Amazon datasets, especially in the instant

video dataset. The average improvement of *HR@30* is 42.3% and the average improvement of *MRR@30* is 21.4%.

Table 2. Overall performance comparison.

Dataset	Metric	Popularity	SVD	DDPG-KNN (k = 1)	DDPG-KNN (k = 0.1N)	DQN-R	TPGR	ATRec	Improv.
Instant Video	Reward@10	0.00004	0.00041	0.00264	0.01619	0.05082	0.11041	**0.15478**	40.2%
	HR@10	0.00004	0.00119	0.00336	0.01807	0.05376	0.11782	**0.17416**	57.7%
	MRR@10	0.00001	0.00041	0.00165	0.00679	0.02117	0.04569	**0.08836**	93.4%
	Reward@30	0.00012	0.003	0.00567	0.04598	0.11228	0.16860	**0.24399**	44.4%
	HR@30	0.00016	0.00507	0.00713	0.05137	0.12150	0.18169	**0.28041**	54.3%
	MRR@30	0.00002	0.00075	0.00190	0.00884	0.02497	0.05023	**0.09431**	87.8%
Musical Instruments	Reward@10	0.00008	0.00056	0	0.00004	0.03542	0.05270	**0.05575**	4.8%
	HR@10	0.00010	0.00054	0	0.00005	0.03965	0.05998	**0.06669**	11.1%
	MRR@10	0.00003	0.00011	0	0.00002	0.00506	**0.03754**	0.02262	−39.7%
	Reward@30	0.00013	0.00332	0.00048	0.00004	0.06618	0.07350	**0.09943**	35.3%
	HR@30	0.00015	0.00471	0.00060	0.00005	0.07196	0.08056	**0.12180**	51.1%
	MRR@30	0.00004	0.00034	0.00004	0.00002	0.00747	**0.03859**	0.02590	−32.9%
Baby	Reward@10	0	0.00373	0.00019	0.00208	0.04811	**0.06859**	0.06842	−0.00%
	HR@10	0	0.00054	0.00019	0.00216	0.05405	0.07514	**0.08587**	14.3%
	MRR@10	0	0.00086	0.00003	0.00057	0.01791	0.03523	**0.03628**	3.0%
	Reward@30	0.00001	0.00965	0.00019	0.00370	0.09720	0.12384	**0.12885**	4.0%
	HR@30	0.00003	0.01871	0.00019	0.00381	0.11064	0.13759	**0.16719**	21.5%
	MRR@30	0	0.00220	0.00003	0.00066	0.02142	0.03973	**0.04311**	9.3%

The conventional methods such as popularity and matrix factorization obtain bad performance under the three datasets. DDPG-KNN method performs worse in $K = 1$, and improves at $K = 0.1N$ in both Instant Video and Baby datasets, where N denotes the total number of items. However, in the Musical Instruments dataset, which contains more items than the other two datasets, DDPG-KNN performs even worse at $K = 0.1N$. The DQN-R performs much better than the DDPG-KNN method. TPGR is better than the other baseline methods in the three datasets.

Compared with TPGR, the proposed ATRec method outperforms TPGR in all three datasets, which can be explained for two reasons. First, the state representation model in ATRec can capture more context information in sequence than TPGR, which only uses the final states of RNNs. The attentive state representation model is more robust in dealing with the long and noisy dependencies lying in the data. Second, the state representation model in ATRec is fully learned with an online reinforcement learning method, while the state representation model in TPGR is trained using offline supervised learning. The online learning method of the state representation model can update the model with the growth of the data it encountered, making it able to adjust the dynamic change of user preference and also lead to better performance in both reward and hit rate.

4.3. Efficiency Evaluation (RQ2)

In this subsection, we evaluate the learning and decision efficiency of the proposed methods with the baseline methods. The length of each episode is also set to 32. We run each method on an AMD Ryzen 3600 6-core CPU with NVIDIA GeForce RTX2060 GPU. For the learning efficiency, we compare the average learning time per step for ATRec with three RL-based recommendation methods, i.e., DDPG-KNN, DQN-R, and ATRec, as shown in Table 3. Among the baseline methods, TPGR performs best as it can improve the learning efficiency at $O(\log M)$. With the help of parameter-sharing strategies, ATRec can greatly reduce the parameters in learning and thus improving learning efficiencies at an average improvement of 35.5% on the three benchmark datasets. Specifically, the full-online

algorithm ATRec needs less time in learning than TPGR, of which the state representation model is off-line trained.

Table 3. Learning efficiency comparison (seconds).

Method	Instant Video	Musical Instruments	Baby
DDPG-KNN ($k = 1$)	0.56430	0.23432	1.42474
DDPG-KNN ($k = 0.1N$)	0.88891	1.15585	1.72415
DQN-R	0.64808	5.64286	1.75619
TPGR	0.07419	0.07971	0.06205
ATRec	**0.05439**	**0.06201**	**0.04382**
Improv.	36.4%	28.5%	41.6%

For decision efficiency, we specifically compare the two tree-based methods, i.e., ATRec and TPGR, and evaluate the average decision time per item and average decision time for top-K items. The results are shown in Table 4. For TPGR, the decision time for each item is faster than ATRec, as it just needs a transverse from root to leaf and only needs to calculate the nodes over the path, while ATRec needs to calculate all the output of the tree nodes. However, the decision time for calculating top-K items needs more time in TPGR, as ATRec can obtain all the recommendation probabilities for each item in a single computation, while TPGR needs to calculate the probabilities for each item iteratively. Therefore, in practice, ATRec is more efficient and applicable in real-world recommendation systems than TPGR.

Table 4. Efficiency evaluation for decision making (seconds).

Method	Average Decision Time per Item			Average Decision Time for Top-K Items		
	Instant Video	Musical Instruments	Baby	Instant Video	Musical Instruments	Baby
TPGR	0.00073	0.00072	0.00071	6.03045	12.66505	5.91112
ATRec	0.00939	0.01135	0.000814	0.00939	0.01135	0.000814

4.4. Influence of the Attention-Based State Representation (RQ3)

We also conduct experiments to show the influence of the attention-based state representation in ATRec comparing with four state representation models:

- Caser [51]: A popular CNN-based model for sequential recommendation by embedding the sequence with multiple convolutional filters to capture the user dynamics.
- ATEM [26]: An attention-based model for sequential recommendation. Compared to the proposed state representation model, ATEM ignores the user feature and feedback.
- TPGR's state representation model [5]: An RNN-based state representation model that encodes the state with the final output of RNN.
- DRR-att [4]: An attention-based state representation model that uses an attention mechanism and average pool to obtain the user's feature. Compared to DRR-att, our method introduces the user's feedback and preprocessed the user and item feature with a fully connected layer.

To make a fair comparison, for each state representation model, we use the same tree policy network as ATRec. The only difference between the baseline methods and ATRec is the state representation model. The method is completely trained online. We report the hit rate@30 of the five state representation models under five different episode lengths: 8, 16, 32, 64, and 128. The experimental result is shown in Figure 5.
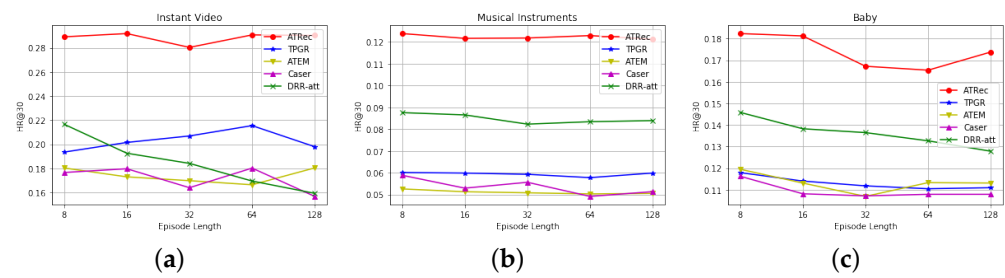
Figure 5. Hit rate@30 of different state modeling methods over different episode lengths. (**a**) Instant Video; (**b**) Musical Instruments; (**c**) Baby.

For the two sequential recommendation models Caser and ATEM, poor performances were obtained with the IRS settings. The proposed ATRec clearly outperforms all the baseline methods in the three benchmark datasets. Compared to DRR-att, the preprocessing of the input user and item features improves the effectiveness of the state representation model. In addition, with the increase in episode length, our method can obtain better performances in Instant Video and Musical Instruments datasets. The performance of DRR-att falls as the length of the episode grows, especially in the Instant Video dataset.

We also notice that the ATRec method with TPGR's state representation outperforms the original TPGR, which implies that the complete tree policy with a parameter-shared online learning framework is more effective than the offline one.

4.5. Effect of Different Parameter Sharing Strategies (RQ4)

In this subsection, we evaluate the effect of different parameter-sharing strategies: layer-shared and all-shared. We report the performance and efficiency of the two strategies, as depicted in Table 5. To make a fair comparison, ATRec methods with two sharing strategies only differ in sharing strategy. We notice that in most cases, all-shared strategy is better than layer-shared strategy, both in performance and efficiency, except in the baby dataset. Therefore, we can conclude that in most cases, the layer-shared strategy with ATRec is enough for IRS policy representation. The unshared tree structure in TPGR is not efficient in learning, as it contains redundant parameters.

Table 5. Performance comparison of different feature extraction methods.

Dataset	Metric	All-Shared	Layer-Shared
Instant Video	Reward@10	**0.15478**	0.11970
	HR@10	**0.15151**	0.17416
	MRR@10	**0.08836**	0.05442
	Reward@30	**0.24399**	0.22471
	HR@30	**0.28041**	0.25890
	MRR@30	**0.09431**	0.06133
	Average learning time per-step	**0.05439**	0.07010
	Average decision time per-step	**0.00939**	0.01216
Musical Instruments	Reward@10	**0.05500**	0.05191
	HR@10	**0.06776**	0.06305
	MRR@10	**0.02333**	0.02208
	Reward@30	**0.09998**	0.10186
	HR@30	**0.12205**	0.12486
	MRR@30	**0.02647**	0.02556
	Average learning time per-step	**0.06201**	0.08463
	Average decision time per-step	**0.01135**	0.01593

Table 5. *Cont.*

Dataset	Metric	All-Shared	Layer-Shared
	Reward@10	**0.07272**	0.06842
	HR@10	**0.08944**	0.08587
	MRR@10	**0.03883**	0.03628
Baby	Reward@30	0.12885	**0.13162**
	HR@30	0.16719	**0.17207**
	MRR@30	**0.04311**	0.04106
	Average learning time per-step	**0.04382**	0.05972
	Average decision time per-step	**0.00814**	0.01149

5. Conclusions

In this paper, we propose ATRec, an attention-based tree policy for large-scale interactive recommendation. ATRec first adopts an attentive state representation model to capture the user dynamics under the long and noisy user–item interaction sequences and then uses a complete tree to present the policy to improve the decision efficiency. The parameter-sharing strategy is also introduced to ATRec to improve learning efficiency. Extensive experiments on three real-world benchmark datasets demonstrate that ATRec provides better recommendation performance and significant learning and decision efficiency over the state-of-the-art methods. One limitation for ATRec is that it needs to know all the available items in the recommendation scenario, which is not always applicable to real-world settings, as the number of items may increase. In such a setting, the policy network is difficult to construct with the unknown action numbers. In the future, we will study this problem to improve the applicability of ATRec in real-world applications.

Author Contributions: L.S. developed the method, conducted the experiment and wrote the manuscript. Q.Z. and Z.Z. validated the method. S.W., B.Z., M.W. and S.L. supervised the research work and the writing of the manuscript. All authors have read and agreed to the published version of the manuscript.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The research work in this paper uses a publicity available dataset, which can be accessed here: https://jmcauley.ucsd.edu/data/amazon/, accessed on 10 April 2023.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Experiment Details

For the three datasets, the length of each episode is set to 32 and the discount factor γ in RL is searched over $\{0.6, 0.9, 0.99, 1\}$, and we found that $\gamma = 1$ obtains the best performance for most methods.

For ATRec, we use a single layer neural network with 64 neurons as the node of the tree policy network. In the attention-based state representation model, the hidden size for the fully connected layer is also set to 64. The learning rate is searched over $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. For the matrix factorization method, we decompose the user–item matrix to obtain 128-dimensional user and item feature vectors. Additionally, The user rating (with range $[a, b]$) is mapped to a one-hot vector with a mapping function as below:

$$onehot\_mapping(rating) = onehot(l - floor(\frac{l \times (b - rating)}{b - a}), l)$$

where $floor(x)$ returns the largest integer no greater than x and $one\_hot(i,l)$ returns an l-dimensional vector with the ith elements is set to 1 while others are set to 0.

For DDPG-KNN, we use a three-layer neural network to represent the actor network and critic network. The hidden size of the neural network is set to 64 for each layer. The learning rates for both actor and critic is searched over $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ to obtain the best performance. The size of experience replay is set to 1,000,000 and the batch size of each learning step is set to 64. The soft-update parameter τ for DDPG is set to 0.05. To improve the search efficiency of KNN, we use FLANN [52] for implementation.

For DQN-R, we also use a three-layer neural network to represent the Q-network. The hidden size of the neural network is set to 64 for each layer. The learning rate for Q-network is searched over $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ to obtain the best performance. The soft-update period is set to 500 steps. In addition, the size of experience replay is set to 1,000,000 and the batch size of each learning step is set to 64.

The implementation of TPGR is similar to [5], except some hyperparameters are well tuned. The learning rate for TPGR is searched over $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. The state representation model for TPGR is learned offline by supervised learning.

References

1. Zou, L.; Xia, L.; Gu, Y.; Zhao, X.; Liu, W.; Huang, J.X.; Yin, D. Neural Interactive Collaborative Filtering. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 25–30 June 2020; pp. 749–758.
2. Zhao, X.; Zhang, W.; Wang, J. Interactive collaborative filtering. In Proceedings of the 22nd ACM international conference on Information and Knowledge Management, San Francisco, CA, USA, 27 October–1 November 2013; pp. 1411–1420.
3. Zhang, Q.; Cao, L.; Shi, C.; Niu, Z. Neural Time-Aware Sequential Recommendation by Jointly Modeling Preference Dynamics and Explicit Feature Couplings. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 5125–5137. [CrossRef] [PubMed]
4. Liu, F.; Tang, R.; Li, X.; Zhang, W.; Ye, Y.; Chen, H.; Guo, H.; Zhang, Y.; He, X. State representation modeling for deep reinforcement learning based recommendation. *Knowl. Based Syst.* **2020**, *205*, 106170. [CrossRef]
5. Chen, H.; Dai, X.; Cai, H.; Zhang, W.; Wang, X.; Tang, R.; Zhang, Y.; Yu, Y. Large-scale Interactive Recommendation with Tree-structured Policy Gradient. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 3312–3320.
6. Zhao, X.; Zhang, L.; Ding, Z.; Xia, L.; Tang, J.; Yin, D. Recommendations with negative feedback via pairwise deep reinforcement learning. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 1040–1048.
7. Wang, S.; Hu, L.; Wang, Y.; Sheng, Q.Z.; Orgun, M.; Cao, L. Modeling multi-purpose sessions for next-item recommendations via mixture-channel purpose routing networks. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; AAAI Press: Palo Alto, CA, USA, 2019; pp. 3771–3777.
8. Wang, S.; Pasi, G.; Hu, L.; Cao, L. The era of intelligent recommendation: Editorial on intelligent recommendation with advanced AI and learning. *IEEE Intell. Syst.* **2020**, *35*, 3–6. [CrossRef]
9. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
10. Yang, L.; Zheng, Q.; Pan, G. Sample complexity of policy gradient finding second-order stationary points. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; Volume 35, pp. 10630–10638.
11. Yang, L.; Zheng, G.; Zhang, Y.; Zheng, Q.; Li, P.; Pan, G. On convergence of gradient expected sarsa (λ). In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; Volume 35, pp. 10621–10629.
12. Shi, J.C.; Yu, Y.; Da, Q.; Chen, S.Y.; Zeng, A.X. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4902–4909.
13. Zou, L.; Xia, L.; Ding, Z.; Song, J.; Liu, W.; Yin, D. Reinforcement learning to optimize long-term user engagement in recommender systems. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2810–2818.
14. Zhou, S.; Dai, X.; Chen, H.; Zhang, W.; Ren, K.; Tang, R.; He, X.; Yu, Y. Interactive recommender system via knowledge graph-enhanced reinforcement learning. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 25–30 July 2020; pp. 179–188.
15. Cai, X.; Han, J.; Li, W.; Zhang, R.; Pan, S.; Yang, L. A Three-Layered Mutually Reinforced Model for Personalized Citation Recommendation. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 6026–6037. [CrossRef] [PubMed]
16. Zhao, X.; Xia, L.; Zhang, L.; Ding, Z.; Yin, D.; Tang, J. Deep reinforcement learning for page-wise recommendations. In Proceedings of the 12th ACM Conference on Recommender Systems, Vancouver, BC, USA, 2 October 2018; pp. 95–103.

17. Afsar, M.M.; Crump, T.; Far, B. Reinforcement learning based recommender systems: A survey. *ACM Comput. Surv.* **2022**, *55*, 1–38. [CrossRef]

18. Wang, S.; Hu, L.; Cao, L.; Huang, X.; Lian, D.; Liu, W. Attention-based transactional context embedding for next-item recommendation. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI 2018. Association for the Advancement of Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 2532–2539.

19. Wang, S.; Cao, L.; Hu, L.; Berkovsky, S.; Huang, X.; Xiao, L.; Lu, W. Hierarchical attentive transaction embedding with intra-and inter-transaction dependencies for next-item recommendation. *IEEE Intell. Syst.* **2021**, *36*, 56–64. [CrossRef]

20. Wang, N.; Wang, S.; Wang, Y.; Sheng, Q.Z.; Orgun, M. Modelling local and global dependencies for next-item recommendations. In Proceedings of the International Conference on Web Information Systems Engineering, Amsterdam, The Netherlands, 20–24 October 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 285–300.

21. Song, W.; Wang, S.; Wang, Y.; Wang, S. Next-item recommendations in short sessions. In Proceedings of the 15th ACM Conference on Recommender Systems, Amsterdam, The Netherlands, 27 September–1 October 2021; pp. 282–291.

22. Zhang, J.; Hao, B.; Chen, B.; Li, C.; Chen, H.; Sun, J. Hierarchical reinforcement learning for course recommendation in MOOCs. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 435–442.

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

24. Zheng, G.; Zhang, F.; Zheng, Z.; Xiang, Y.; Yuan, N.J.; Xie, X.; Li, Z. DRN: A deep reinforcement learning framework for news recommendation. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 167–176.

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

26. Wang, S.; Hu, L.; Wang, Y.; Cao, L.; Sheng, Q.Z.; Orgun, M. Sequential recommender systems: Challenges, progress and prospects. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 10–16 August 2019; pp. 6332–6338.

27. Chen, X.; Li, S.; Li, H.; Jiang, S.; Qi, Y.; Song, L. Generative adversarial user model for reinforcement learning based recommendation system. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 1052–1061.

28. Zou, L.; Xia, L.; Du, P.; Zhang, Z.; Bai, T.; Liu, W.; Nie, J.Y.; Yin, D. Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020; pp. 816–824.

29. Lei, T.; Zhang, Y.; Artzi, Y. Training rnns as fast as cnns. In Proceedings of the ICLR 2018, Vancouver, BC, Canada, 30 August–3 May 2018.

30. Gao, R.; Xia, H.; Li, J.; Liu, D.; Chen, S.; Chun, G. DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation. In Proceedings of the 2019 IEEE International Conference on Data Mining (ICDM), Beijing, China, 8–11 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1048–1053.

31. Sun, F.; Liu, J.; Wu, J.; Pei, C.; Lin, X.; Ou, W.; Jiang, P. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In Proceedings of the CIKM '19: 28th ACM International Conference on Information and Knowledge Management, New York, NY, USA, 3–7 November 2019; pp. 1441–1450. [CrossRef]

32. Wang, S.; Cao, L.; Wang, Y.; Sheng, Q.Z.; Orgun, M.A.; Lian, D. A survey on session-based recommender systems. *Acm Comput. Surv. (CSUR)* **2021**, *54*, 1–38. [CrossRef]

33. Liu, H.; Cai, K.; Li, P.; Qian, C.; Zhao, P.; Wu, X. REDRL: A review-enhanced deep reinforcement learning model for interactive recommendation. *Expert Syst. Appl.* **2022**, *213*, 118926. [CrossRef]

34. Wang, S.; Hu, L.; Wang, Y.; He, X.; Sheng, Q.Z.; Orgun, M.A.; Cao, L.; Ricci, F.; Yu, P.S. Graph learning based recommender systems: A review. In Proceedings of the 30th International Joint Conference on Artificial Intelligence, Virtual, 19–26 August 2021; AAAI Press: Palto, CA, USA 2021; pp. 4644–4652.

35. Wang, X.; Xu, Y.; He, X.; Cao, Y.; Wang, M.; Chua, T.S. Reinforced negative sampling over knowledge graph for recommendation. In Proceedings of the Web Conference 2020, Taipei, Taiwan, 20–24 April 2020; pp. 99–109.

36. Wang, P.; Fan, Y.; Xia, L.; Zhao, W.X.; Niu, S.; Huang, J. KERL: A knowledge-guided reinforcement learning model for sequential recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 25–30 July 2020; pp. 209–218.

37. Chen, X.; Huang, C.; Yao, L.; Wang, X.; Zhang, W. Knowledge-guided deep reinforcement learning for interactive recommendation. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–8.

38. Xian, Y.; Fu, Z.; Muthukrishnan, S.; De Melo, G.; Zhang, Y. Reinforcement knowledge graph reasoning for explainable recommendation. In Proceedings of the 42nd international ACM SIGIR Conference on Research and Development in Information Retrieval, Paris, France, 21–25 July 2019; pp. 285–294.

39. Sallans, B.; Hinton, G.E. Reinforcement learning with factored states and actions. *J. Mach. Learn. Res.* **2004**, *5*, 1063–1088.

40. Pazis, J.; Parr, R. Generalized value functions for large action sets. In Proceedings of the ICML, Washington, DC, USA, 28 June–2 July 2011.

41. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]

42. Chandak, Y.; Theocharous, G.; Kostas, J.; Jordan, S.; Thomas, P. Learning action representations for reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 941–950.

43. Fu, M.; Agrawal, A.; Irissappane, A.A.; Zhang, J.; Huang, L.; Qu, H. Deep reinforcement learning framework for category-based item recommendation. *IEEE Trans. Cybern.* **2021**, *52*, 12028–12041. [CrossRef] [PubMed]

44. Mnih, A.; Hinton, G.E. A scalable hierarchical distributed language model. *Adv. Neural Inf. Process. Syst.* **2008**, *21*, 1081–1088.

45. Funk, S. Netflix Update: Try This at Home. Available online: http://sifter.org/simon/journal/20061211.html (accessed on 9 June 2023).

46. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]

47. He, R.; McAuley, J. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In Proceedings of the 25th International Conference on World Wide Web, Montreal, QC, Canada, 11–15 April 2016; pp. 507–517.

48. McAuley, J.; Targett, C.; Shi, Q.; Van Den Hengel, A. Image-based recommendations on styles and substitutes. In Proceedings of the 38th international ACM SIGIR Conference on Research and Development in Information Retrieval, Santiego, Chila, 9–13 August 2015; pp. 43–52.

49. Wang, S.; Xu, X.; Zhang, X.; Wang, Y.; Song, W. Veracity-aware and event-driven personalized news recommendation for fake news mitigation. In Proceedings of the ACM Web Conference 2022, Lyon, France, 25–29 April 2022; pp. 3673–3684.

50. Wang, S.; Zhang, X.; Wang, Y.; Liu, H.; Ricci, F. Trustworthy recommender systems. *arXiv* **2022**, arXiv:2208.06265.

51. Tang, J.; Wang, K. Personalized top-n sequential recommendation via convolutional sequence embedding. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, Marina Del Rey, CA, USA, 5–9 February 2018; pp. 565–573.

52. Muja, M.; Lowe, D. Fast Library for Approximate Nearest Neighbors (FLANN). Available online: git://github.com/mariusmuja/flann.git or http://www.cs.ubc.ca/research/flann (accessed on 9 June 2023).