World Wide Web (2023) 26:4153–4172
https://doi.org/10.1007/s11280-023-01215-6

Click is not equal to purchase: multi-task reinforcement learning for multi-behavior recommendation

Huiwang Zhang[1] · Pengpeng Zhao[1] · Xuefeng Xian[2] · Victor S. Sheng[3] · Yongjing Hao[1] · Zhiming Cui[4]

Received: 31 March 2023 / Revised: 21 September 2023 / Accepted: 25 September 2023 /
Published online: 20 December 2023

Abstract
Reinforcement learning (RL) has achieved ideal performance in recommendation systems (RSs) by taking care of both immediate and future rewards from users. However, the existing RL-based recommendation methods assume that only a single type of interaction behavior (e.g., clicking) exists between user and item, whereas practical recommendation scenarios involve multiple types of user interaction behaviors (e.g., adding to cart, purchasing). In this paper, we propose a Multi-Task Reinforcement Learning model for multi-behavior Recommendation (MTRL4Rec), which gives different actions for users' different behaviors with a single agent. Specifically, we first introduce a modular network in which modules can be shared or isolated to capture the commonalities and differences across users' behaviors. Then a task routing network is used to generate routes in the modular network for each behavior task. We adopt a hierarchical reinforcement learning architecture to improve the efficiency of MTRL4Rec. Finally, a training algorithm and a further improved training algorithm are proposed for our model training. Experiments on two public datasets validated the effectiveness of MTRL4Rec.

Keywords Recommendation system · Multi-behavior modeling · Reinforcement learning · Multi-task learning

1 Introduction

With the increasingly serious problem of information overload, it is difficult for users to find the content they are interested in when faced with vast and complex information. To this

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2022*
Guest Editor: Richard Chbeir, Helen Huang, Yannis Manolopoulos, Fabrizio Silvestri.

✉ Pengpeng Zhao
ppzhao@suda.edu.cn

✉ Xuefeng Xian
xianxuefeng@jssvc.edu.cn

Extended author information available on the last page of the article

end, recommendation systems assist users by suggesting personalized items that best fit their needs and preferences. It is widely used in many fields, such as e-commerce [1, 2], news [3], music [4, 5]. The deep learning (DL)-based methods have become the current mainstream due to their capability of modeling complex user-item interactions [6–9]. For example, the work of Hidasi et al. [10] adopts recurrent neural networks (RNNs) to model the sequence information of user interactions. BERT4Rec [11] and S^3-Rec [12] model sequence information through self-supervised learning (SSL), for better user and item representation. Since these DL-based methods cannot continuously update the strategies during the interactions and maximize the expected long-term cumulative reward from users, RL-based methods were proposed. For example, Shani et al. [13] modeled RS as a Markov decision process (MDP) and estimated the transition probability and the Q-value table. Zheng et al. [3] proposed a deep Q-network (DQN) based method which can take care of both immediate and future rewards. Zhao et al. [14] argued that some recommended items that users skipped can influence the recommendation performance and proposed a model considering both positive feedback and negative feedback. Zhao et al. [15] designed a deep hierarchical reinforcement learning model to overcome the problem of sparse feedback signals in RSs.

However, existing RL-based recommendation methods ignore the interest gap in the different behaviors of users. The interest gap means that users have different preferences on different behaviors. In simple terms, a person may be interested in an item and click on it, but never buy it. It pays off to value this difference and use several recommendation strategies. The recommendation on the homepage should use a click-targeted strategy, and when placing an order, it should be a purchase-targeted strategy. Suppose we treat each behavior as an individual task, and build several independent models with existing methods for multi-behavior recommendation. It costs too much and ignores the commonality between tasks since the models are independent.

To this end, we propose a **M**ulti-**T**ask **R**einforcement **L**earning **for** multi-behavior **Rec**ommendation (MTRL4Rec), which gives different actions for users' different behaviors with a single policy. Firstly, we use a modular network to model the commonalities and differences between behavior tasks. The modular network consists of several Modular Units (MoUs). Next, a task routing network is used to select routes automatically for different behavior tasks in the modular network. It reweighs the outputs of each MoU in the modular network for different behavior tasks. In this way, the MoUs that model commonalities of tasks can be reused, and the MoUs that model differences of tasks can be isolated. Then, to improve the model's efficiency, we adopt a hierarchical reinforcement learning structure consisting of a high-level agent and a low-level agent. The high-level agent is a category selector which generates the category that should be recommended. In contrast, the low-level agent recommends a specific item to users in the category selected by the high-level agent. Additionally, we provide an off-policy training algorithm for the MTRL4Rec to train our model. We then extend MTRL4Rec to a double deep Q-network (DDQN) reinforcement learning algorithm for higher performance and stability. Finally, we evaluated our method on pre-trained environment simulators, as [14]. Experiments on two datasets validate the effectiveness of our MTRL4Rec and MTRL4Rec-DDQN.

We summarize our major contributions as follows:

- To the best of our knowledge, this is the first work to treat the multi-behaviors as multi-tasks in an RL-based recommendation.
- We propose an MTRL4Rec model, which uses a modular network and a task routing network to solve the multi-task problem in multi-behavior recommendation with reinforcement learning.

- We further extend our MTRL4Rec method to a DDQN reinforcement learning algorithm, namely MTRL4Rec-DDQN, which improves the performance and stability of the model.
- Experimental results demonstrate that users do have different preferences in different behaviors, and our MTRL4Rec model outperforms state-of-the-art models in multi-behavior recommendation. The experiment also verified that MTRL4Rec-DDQN has better performance and stability.

2 Related work

In this section, we briefly review related work, including recommendation system, reinforcement learning and multi-task learning.

2.1 Recommendation system

Recommendation systems help users find items that match their preferences and needs by giving personalized recommendations. Currently, deep learning based methods have become the mainstream of RS research. In past research on recommendation systems, He et al. [16] proposed a neural collaborative filtering (NCF) method, which represents users and items into low-dimensional vectors, and combines the matrix factorization (MF) model and the multi-layer perceptron model in the hidden layer. Self-supervised graph learning (SGL) [17] is proposed by Wu et al., in which a self-supervised method is used to alleviate the problem of data sparsity. Some works model the user's interaction history sequence to capture the user's dynamic preference and then predict the next item the user may be interested in. The work of Hidasi et al. [10] adopts recurrent neural networks (RNNs) to model the sequence information of user interactions. BERT4Rec [11] and S^3-Rec [12] model sequence information through self-supervised learning (SSL), for better user and item representation.

However, these methods cannot continuously update the strategy during the interaction process and maximize the user's long-term cumulative reward. Therefore, some reinforcement learning based research has been proposed to solve this problem.

2.2 Reinforcement Learning

Reinforcement learning is a type of machine learning that focuses on training agents to make decisions based on feedback from their environment. In [18], deep reinforcement learning (DRL) is employed in making Q-learning and deep neural networks (DNNs) merge together. The methods of past DRL can be summarized into three categories: value-based methods (such as DQN [18], DDQN [19]), policy gradient methods (such as REINFORCE [20], REINFORCE-wb [21]), and actor-critic methods (such as DDPG [22], SAC [23]).

Reinforcement learning based recommendation systems (RLRSs) use reinforcement learning to continuously update the strategy during the interaction and maximize the expected long-term cumulative reward from users [24]. Shani et al. [13] modeled the RS as an MDP process and estimated the transition probability and the Q-value table. Zheng et al. [3] proposed a DQN based method that can take care of both immediate and future reward. Zhao et al. [14] argued that some recommended items that users skipped can influence the recommendation performance and proposed a model considering both positive feedback and negative feedback. In the course recommendation scenario, users may be interested in various courses. For this reason, Zhang et al. [25] proposed a hierarchical reinforcement learning algorithm to

revise the user profiles, and tune the course recommendation model on the revised profiles. Per et al. [26] built an RL-based value-aware recommendation to maximize the profit of an RS directly. Bai et al. [27] proposed a solution to more effectively utilize log data with model-based RL algorithms to avoid the high interaction cost. Wang et al. [28] proposed a KERL model for fusing knowledge graph information into an RL-based sequential recommendation system.Hierarchical reinforcement learning (HRL) was proposed to solve more complex and challenging RL tasks in the work of Barto and Mahadevan [29]. This is because it allows the agent to learn different sub-policies hierarchically in a specific task to make step-by-step decisions. Zhao et al. [15] designed a multi-goals abstraction based HRL algorithm to overcome the problem of sparse feedback signals in recommendation systems. In the hierarchical reinforcement learning recommendation method (HRL-Rec) designed by Xie et al. [30], the recommendation is divided into two steps. One is to recommend a channel, and one is to recommend specific items. However, these are all single-task methods and cannot perform well in multi-behavior recommendation systems.

In RLRS research, building a suitable environment to evaluate methods is challenging. There are mainly three methods for evaluation in previous work: online test, offline test, and simulation. In the online method, the methods are evaluated while interacting with real users and in real-time, such as [30, 31]. Obviously, this is the best way to evaluate RLRS methods. But it may bring terrible experiences to real users. In the offline methods, the environment is a static dataset from user interaction logs, as [32–34] did. This method can make full use of the log data in history. However, it cannot fine-tune the recommendation strategy according to the dynamic changes of the environment. In the simulation method, a user model is established, known as a simulator. In the researches of [15, 35, 36], the proposed RLRS methods interact with simulators for training and evaluation.

2.3 Multi-task learning

Multi-task learning (MTL) is one of the core topics of machine learning [37]. Many researches [38, 39] have shown that multiple objectives make different tasks benefit each other. Strezoski et al. [40] use a task routing layer to generate a task-specific binary mask, which is applied to the neural network. Liu et al. [41] adopt a shared Bidirectional Encoder Representations from Transformers (BERT) embedding layer in multi-task scenarios. Multi-task Learning is also a challenging problem in RL. Singh [42] proposed a method of selecting different Q-functions for different tasks using a gate mechanism. Wilson et al. [43] proposed a hierarchical Bayesian-based multi-task RL framework. The work of Pinto and Gupta [44] uses a shared trunk architecture to jointly learn robot moving. Instead of directly selecting routes for each task, a soft modularization method is proposed in the work of Yang et al. [45] to softly combine possible routes. Some works adopt MTL to improve the recommendation performance. MOE [46] is proposed to share some experts at the bottom and combine experts through a gating network. Ma et al. [47] extended MOE to utilize different gates for each task to obtain different fusing weights in MTL. Tang et al. [48] found seesaw phenomena in MTL and proposed a progressive layered extraction method.

However, no previous work has attempted to improve multi-behavior recommendation using multi-task reinforcement learning. In this work, we use a multi-task reinforcement learning framework with modular networks and task routing networks to implement the multi-behavior recommendation.

3 Method

We model recommendation tasks on different behaviors as several Markov decision processes (MDPs) and leverage reinforcement learning to automatically learn the optimal recommendation strategy. The user is the environment, while the recommendation system that recommends items to users is the agent. The user's information and interaction history is the state of the environment. According to the current state, the agent selects an action (recommends an item to the user). The feedback from the users is seen as a reward to the agent. Then the agent is updated according to the reward, and the following interaction starts.

We define our problem as follows When a user u interacts with the agent (i.e., the recommendation system) and has a request on task (behavior) t, the agent is going to select an action a from the item set for this user. Then, the agent observes the user's feedback r and adjusts the policy for the next time recommendation.

3.1 Basic RL model for multi-behavior recommendation

In this subsection, we discuss two methods to implement multi-behavior recommendation using existing RL-based recommendation methods. As shown in Figure 1(a), each task has its corresponding agent. Recommendation agents for different behaviors are completely independent of each other. Figure 1(b) depicts another method. In this method, we allow the agent to observe the state, action, and reward of other tasks to improve its performance.

Algorithm 1 Off-policy Training of Single-task RL Model for Multi-behavior Recommendation

```
1: Initialize the parameters of single-task agents A = {A_1, A_2, ...} for each tasks
2: Initialize the replay memories D = {D_1, D_2, ...} for each tasks
3: for session = 1, 2, ··· , M do
4:     Initialize state s_0 from previous sessions
5:     for step i = 1, 2, ··· , T do
6:         for task t in task_list do
7:             generate a_i^t with A_t
8:         end for
9:         Observe user's behavior as t
10:        Recommend item a_i^t to the user
11:        Observe reward r_i and next state s_{t+1} from user
12:        Store transition (s_i, a_i^t, r_i, s_{i+1}) in D_t
13:        Sample a mini-batch of transitions from D_t
14:        Update parameters of current task agent A_t
15:        if s_{i+1} is terminal state then
16:            break
17:        end if
18:    end for
19: end for
```

The Algorithm 1 describes the method shown in Figure 1(a) formally. The agents and replay memories are initialized in Line 1-2. At the beginning of each session, we use the previous session as the current state (Line 4). If a user does not have a previous session, it will be an empty sequence. Before each interaction step, we call all single-task agents to generate

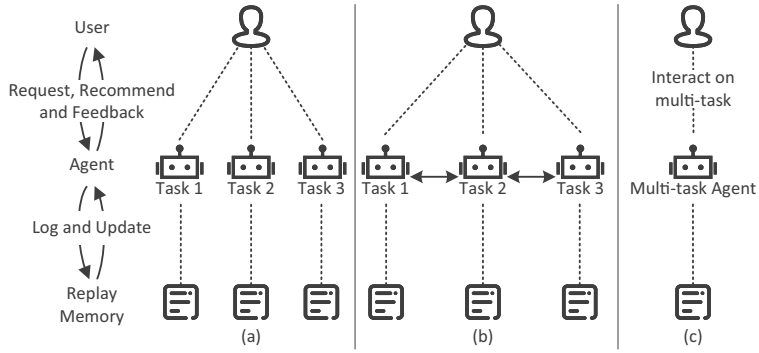

Figure 1 The RL Model for Multi-behavior Recommendation. (a) Recommendation agents for different behavior tasks are completely independent of each other. Each task has its corresponding agent. (b) The tasks are still independent of each other, but agents can observe each other. (c) One agent is responsible for all tasks. An MTRL4Rec model, which is in this framework, is proposed in Section 3.2

recommended items for each task in $task\_list$, which includes all recommendation targets such as clicks, purchases, Etc. (lines 6-8). When a user requests, we give a corresponding recommendation a_i^t based on the user's behavior task t. For example, when a user visits the homepage, a click-targeted recommendation will be given (Line 9-10). And we observe the reward r_i from the user, and store this transition to the replay memory D_t (Line 11-12). Then we sample some transitions from D_t and update the current agent A_t as defined in each RL method (Line 13-14).

3.2 The proposed MTRL4Rec model

We have proposed two methods to implement multi-behavior recommendation using existing RL-based methods in the previous subsection. However, as mentioned in the introduction, existing RL-based methods cannot capture the commonalities between tasks, and tasks cannot benefit from each other. To overcome these problems, we propose a Multi-Task Reinforcement Learning for multi-behavior Recommendation (MTRL4Rec). In our method, a single agent serves multi-behavior recommendation as shown in Figure 1(c). Next, we will describe our MTRL4Rec in detail.

Hierarchical RL Structure For higher exploration and exploitation efficiency in RL, a hierarchical structure is adopted in our model, which consists of a high-level agent (HRA) and a low-level agent (LRA), as shown in Figure 2.

The high-level agent is a category selector. It limits the categories in which items should be recommended. The category information is an attribute of the item and comes from the dataset. The input of the HRA network is a state-category-task tuple (s^h, c, t). And it outputs a score of the category $Q_H(s^h, c, t; \theta^H)$. Then HRA generates the category that should be recommended following an ε-greedy strategy according to:

$$c_i^t = \arg\max_c Q_H(s_i^h, c, t; \theta^H), \tag{1}$$

where θ^H is the learnable parameter set of the HRA network. In other words, the agent chooses the category with the highest score with a probability of $1 - \varepsilon$, or randomly chooses a category with the probability ε to explore. Similarly, the LRA network outputs a score Q_L

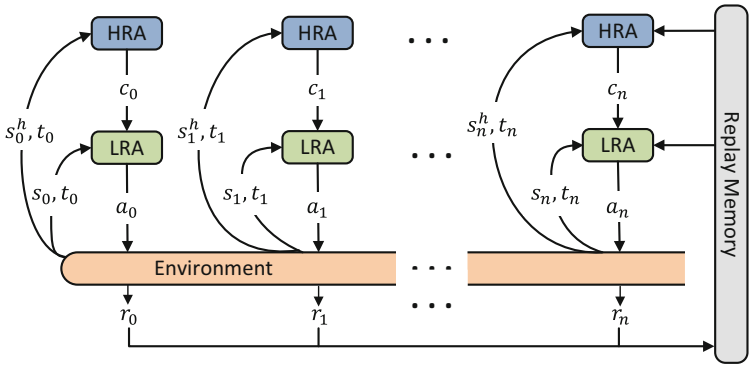

Figure 2 The hierarchical RL framework of our model. In each step of the interaction, the HRA chooses a category, and the LRA recommends an item in the category HRA selected

for each item with parameters θ^L. And the LRA recommends specific items to users, within the limit of the output of HRA, according to:

$$a_i^t = \underset{cate(a)=c_i^t}{\arg\max}\ Q_L(s_i, a, t; \theta^L), \tag{2}$$

where the $cate()$ denotes the category of an item.

LRA Network The structure of the LRA network is shown in Figure 3. It has three important components (embedding, task routing network, and modular network). The HRA network has a similar structure whose action space and the state space are at the category level compared with LRA. The illustration of the HRA network structure will be omitted to avoid repetition.

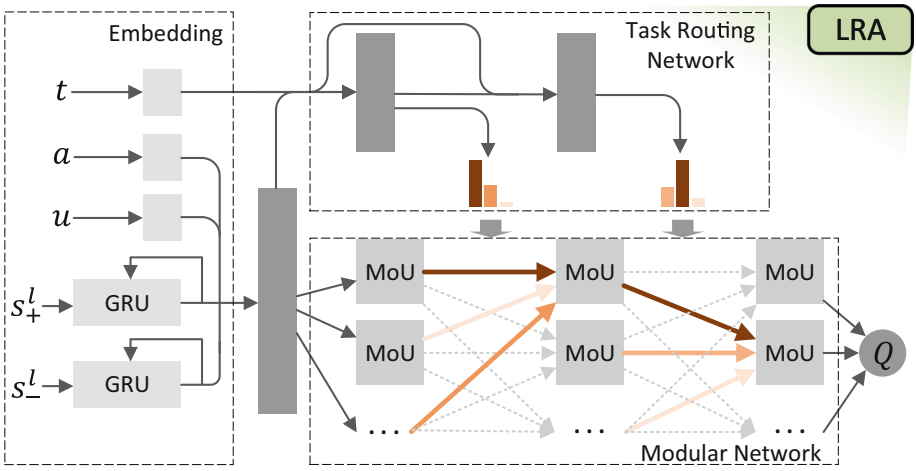

Figure 3 LRA network. It has three critical components (embedding, modular network, and task routing network). The modular network captures the commonalities and differences between tasks, and the task routing network selects a route for each task

Embedding The role of this component is to process the input of the network. The one-hot inputs (action, user, and task tag) are mapped as vectors. Two RNN models with gate recurrent unit (GRU) are utilized. One RNN receives the user's latest interacted item sequence s_+ as inputs and outputs the final hidden state of GRU cells. And another receives the negative item sequence s_- (items that the user skipped) as inputs and outputs the final hidden state of GRU cells. Finally, a fully connected layer merges the action, user, and item sequences (positive and negative) information. Task information is only used as a part of the input of the Task Routing Network.

Modular Network To improve the performance of the model on multiple tasks, we build a Modular Network to capture the commonalities and differences of tasks. The Modular Network has multiple layers, and each layer has M Modular Units (MoUs), as shown in Figure 3. In this way, MoUs that model commonalities can be reused, and MoUs that model differences can be isolated from each other in different tasks. Each MoU has the same multi-layer fully connected structure, as shown in Figure 4.

Task Routing Network The role of the task routing network is to control the route of different tasks in the modular network. It chooses different paths for different tasks in the Modular Network by controlling which MoUs are activated. In addition to the current state, the input of the task routing network also contains the current task, which is not considered in the modular network. It is also a multi-layer structure with one layer less than the modular network. Figure 4 depicts how the task routing network controls the weight of each MoU. Each layer of the task routing network outputs a weight matrix W_l and a hidden state H_l. We formally define this process as (3).

$$(W_l, H_l) = TR_l(s, t, H_{l-1}), \tag{3}$$

where s is the current state embedding, t is the current task embedding, and H_{l-1} is the hidden state of the last layer. We set H_0 as a zero vector. The weight matrix reweighs the outputs of each MoU in the modular network for different tasks. The input of an MoU in the modular network can be calculated according to

$$X_{l+1,k} = softmax(W_{l,k})Y_l, 1 \leq k \leq M, \tag{4}$$

where $W_{l,k}$ is a column vector in W_l for the k-th MoU in layer l, and Y_l is the output of MoUs in the l-th layer of the modular network.

Off-policy Training Algorithm In the MTRL4Rec method, both HRA and LRA adopt the DQN architecture. Here we will illustrate how to utilize the DQN-based algorithm to train our MTRL4Rec model.

First, we randomly initialize the HRA network parameters θ^H and the LRA network parameters θ^L and create an empty replay memory D. At the beginning of a session, we set the initial state s_0 as the user's information. Before each interaction step i, we call the HRA to generate different item categories $\{c_i^{t_1}, c_i^{t_2}, \ldots\}$ according to (1) for different behavior tasks $\{t_1, t_2, \ldots\}$. Then we call the LRA to generate different items $\{a_i^{t_1}, a_i^{t_2}, \ldots\}$ for each task according to (2). At the time-stamp t, we give the corresponding recommendation a_i^t and observe the feedback from the user. If the user interacted with the item, we set the current reward $r_i = 1$, add a_i^t to the positive item sequence s_+ and generate the next state s_{i+1}. Otherwise, we set the current reward $r_i = 0$ and add a_i^t to the negative item sequence s_-.

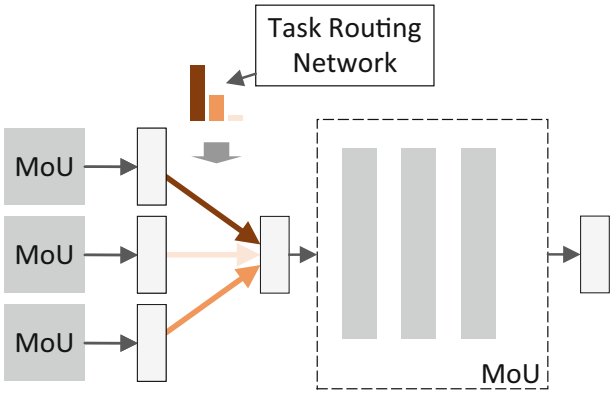

Figure 4 The structure of Modular Unit(MoU). Each MoU has the same multi-layer fully connected structure. The weight of MoU is controlled by the task routing network

Then the transition $\langle s_i, t, a_i^t, r_i, s_{i+1} \rangle$ will be stored to the replay memory D. We sample a mini-batch of transitions $\langle s, t, a, r, s' \rangle$ from D , and update the HRA network parameters θ^H and the LRA network parameters θ^L. The LRA network has the loss function:

$$Loss^L(\theta^L) = \mathbb{E}_{s,a,r,s',t}(y_L - Q_L(s, a, t; \theta^L))^2, \tag{5}$$

where y_L is the target for the current iteration, as in (6):

$$y_L = \begin{cases} r & \text{terminal } s' \\ r + \gamma \max\limits_{a'} Q_L(s', a', t; \theta^L) & \text{otherwise} \end{cases} \tag{6}$$

The hyper-parameter γ is a discount factor that controls the weight of future reward. Equation 5 can be minimized according to:

$$\begin{aligned} \nabla_{\theta^L} Loss^L(\theta^L) = \mathbb{E}_{s,a,r,s',t}[(y_L - Q_L(s, a, t; \theta^L)) \\ \nabla_{\theta^L} Q_L(s, a, t; \theta^L)]. \end{aligned} \tag{7}$$

Similarly, the HRA ignores specific items and generates candidate categories for each behavior task. The state of HRA, s^h, is generated from s. It ignores the specific item information and keeps the category information. The HRA network has the loss function:

$$Loss^H(\theta^H) = \mathbb{E}_{s^h, cate(a), r, s'^h, t}(y_H - Q_H(s^h, cate(a), t; \theta^H))^2, \tag{8}$$

where y_H is calculated from (9):

$$y_H = \begin{cases} r & \text{terminal } s' \\ r + \gamma \max\limits_{c} Q_L(s'^h, c, t; \theta^H) & \text{otherwise.} \end{cases} \tag{9}$$

The HRA network parameters θ^H can be updated according to:

$$\begin{aligned} \nabla_{\theta^H} Loss^H(\theta^H) = \mathbb{E}_{s^h, cate(a), r, s'^h, t}[(y_H - Q_H(s^h, cate(a), t; \theta^H)) \\ \nabla_{\theta^H} Q_H(s^h, cate(a), t; \theta^H)]. \end{aligned} \tag{10}$$

Formally, Algorithm 2 describes the process of off-policy training for our MTRL4Rec model.

Algorithm 2 Off-policy Training of Multi-task Reinforcement Learning Model for Multi-behavior Recommendation

Initialize the HRA network parameters θ^H and the LRA network parameters θ^L
Initialize the replay memory D
for session = 1, 2, ..., M **do**
  Initialize state s_0 from previous sessions
  for step i = 1, 2, ..., T **do**
    for task t in $task\_list$ **do**
      generate c_i^t according to (1)
      generate item a_i^t according to (2)
    end for
    Observe user's behavior as t
    Recommend item a_i^t to the user
    Observe reward r_i and next state s_{i+1} from user
    Store transition $\langle s_i, t, a_i^t, r_i, s_{i+1} \rangle$ in D
    Sample a mini-batch of transitions $\langle s, t, a, r, s' \rangle$ from D
    Update the LRA network parameters θ^L based of (7)
    Update the HRA network parameters θ^H based of (10)
    if s_{i+1} is terminal state **then**
      break
    end if
  end for
end for

A Further Improved Algorithm: MTRL4Rec-DDQN The value function update target contains maximization operation (6), (9). In this process, a behavior is selected through a maximization operation and evaluated maximally. This makes the output of the value function greater than the real value. And this error will be enlarged as the number of behaviors increases. The result is that DQN-based reinforcement learning algorithms have the defect of overestimation.

To this end, we further improve and propose an off-policy update algorithm based on DDQN [19]. Compared with DQN, DDQN is more stable because it adopts a dual network structure and delayed update, effectively solving the problem of overestimating Q-value. DDQN adopts two different value functions to realize the selection and evaluation of actions. The neural network structures corresponding to these two value functions are consistent. It can also be understood that there are two sets of network parameters, θ and θ^-.

First, the online network is used for action selection with θ, consistent with the previous DQN-based method.

$$a^* = \arg\max_a Q(s', a; \theta) \tag{11}$$

Then use the target network with parameters θ^- to evaluate the Q-value of this action, for the update target calculation:

$$y = r + \gamma Q(s', a^*; \theta^-). \tag{12}$$

Then we have:

$$y = r + \gamma Q(s', \arg\max_a Q(s', a; \theta); \theta^-). \tag{13}$$

Specific to our multi-task reinforcement learning approach, randomly initialize the parameters θ^H and θ^L of the HRA and LRA networks, and create an empty replay memory D. Create target network parameters $\theta^{H-} = \theta^H$ and $\theta^{L-} = \theta^L$. At the beginning of each session, set the initial state s_0 to the user profile and the user's previous session. Before each interaction step i, we still call the HRA to generate different item categories $c_i^{t_1}, c_i^{t_2}, \ldots$ for different

behavior tasks $t_1, t_2, \ldots$, according to (1). Then we call the LRA to generate specific items $a_i^{t_1}, a_i^{t_2}, \ldots$ for each task according to the (2). When a user initiates a request on an action t, we provide the corresponding recommendation a_i^t and observe the user feedback. If the user interacts with the item, we set the current reward $r_i = 1$, add a_i^t to the positive feedback item sequence s_+ and generate the next state s_{i+1}. Otherwise, we set the current reward $r_i = 0$ and add a_i^t to the negative feedback sequence s_-. Then, the transition $\langle s_i, t, a_i^t, r_i, s_{i+1} \rangle$ will be stored into replay memory D. So far, it is similar to the DQN-based method. Then, we sample a batch of transitions $\langle s, t, a, r, s' \rangle$ from the replay memory D randomly. The loss function still has the formulas 5 and 8. The difference is the update manner of targets y_L and y_L in the loss functions. Similar to (11), the HRA selects an action (a category) according with θ^H:

$$c^* = \arg\max_c Q_H(s'^h, c, t; \theta^H). \tag{14}$$

Use θ^{H-} to evaluate the action c^* and calculate its update target:

$$y_H = \begin{cases} r & \text{terminal } s' \\ r + \gamma Q_H(s'^h, c^*, t; \theta^{H-}) & \text{otherwise.} \end{cases} \tag{15}$$

We can calculate the update target of LRA parameters according to the following:

$$y_L = \begin{cases} r & \text{terminal } s' \\ r + \gamma Q_L(s', a^*, t; \theta^{L-}) & \text{otherwise,} \end{cases} \tag{16}$$

where $a^* = \arg\max_{cate(a')=c^*} Q_L(s', a', t; \theta^L)$. Then, update θ^H and θ^L according to (8) and (5). After each C times of updating operations on θ^H and θ^L, update θ^{H-} and θ^{L-} once, set $\theta^{H-} = \theta^H$, $\theta^{L-} = \theta^L$.

The off-policy training process of the DDQN-based MTRL4Rec model (MTRL4Rec-DDQN) is formally described in the Algorithm 3.

4 Experiment

In this section, we conduct a series of experiments with the two public datasets to evaluate the effectiveness of our proposed MTRL4Rec. We mainly focus on four research questions (RQs):

- **RQ1:** How do the proposed MTRL4Rec and MTRL4Rec-DDQN perform compared to baselines?
- **RQ2:** How do the components in MTRL4Rec contribute to performance?
- **RQ3:** Can MTRL4Rec and MTRL4Rec-DDQN achieve better performance in the interaction with the online environment?
- **RQ4:** How do the key hyper-parameters affect the model performance?

4.1 Datasets

We select two public datasets for experiments to evaluate our MTRL4Rec and MTRL4Rec-DDQN recommendation performance.

Algorithm 3 Off-policy Training of Multi-task Reinforcement Learning Model for Multi-behavior Recommendation (MTRL4Rec-DDQN)

Initialize the HRA network parameters θ^H and the LRA network parameters θ^L
Initialize the HRA target network parameters $\theta^{H-} = \theta^H$ and the LRA target network parameters $\theta^{L-} = \theta^L$
Initialize the replay memory D
for session = 1, 2, ..., M **do**
  Initialize state s_0 from previous sessions
  for step i = 1, 2, ..., T **do**
    for task t in $task\_list$ **do**
      generate c_i^t according to (1)
      generate item a_i^t according to (2)
    end for
    Observe user's behavior as t
    Recommend item a_i^t to the user
    Observe reward r_i and next state s_{i+1} from user
    Store transition $\langle s_i, t, a_i^t, r_i, s_{i+1} \rangle$ in D
    Sample a mini-batch of transitions $\langle s, t, a, r, s' \rangle$ from D
    Calculate update target y_H and y_L according to (15) and (16)
    Update the LRA network parameters θ^L based of (7)
    Update the HRA network parameters θ^H based of (10)
    Every C steps, set $\theta^{L-} = \theta^L$, $\theta^{H-} = \theta^H$
    if s_{i+1} is terminal state **then**
      break
    end if
  end for
end for

Tmall [1] is a dataset of user behaviors from Tmall for recommendation problems with implicit feedback offered by Alibaba. This dataset randomly selects users' behavior logs, including clicking, purchasing, adding items into the shopping cart, and item favoring, during November 25 to December 03, 2017. It has a total of 100,150,807 interaction logs. Its organization is very similar to MovieLens-20M.

Another dataset is **Tianchi**[2] from the Tianchi competition. It is sampled from the Taobao application and has a similar structure as the Tmall dataset. Compared with Tmall, Tianchi contains side information about users and items, and is smaller, with 12,256,906 interaction logs.

Since the data on item favouring is too sparse, in the following experiment, we choose clicking, purchasing, and adding items into the shopping cart as three different behavior tasks and discard the data on item favoring.

4.2 Baselines

We compare our MTRL4Rec and MTRL4Rec-DDQN with two types of baselines. The first is the state-of-the-art RL-based recommendation methods to demonstrate the overall performance of our MTRL4Rec. We use the method in Algorithm 1 to implement the multi-behavior recommendation for comparison. The second type is the variants of our MTRL4Rec

[1] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

[2] https://tianchi.aliyun.com/dataset/dataDetail?dataId=46

model to show its ablation performance. In detail, we compare our MTRL4Rec method with the following baselines:

- **DQN** [49]: This is a fundamental reinforcement learning method with a deep Q-network. We use an independent deep Q-network for each task. This baseline helps us establish a reference point to verify the performance of the multi-task models.
- **DEERS** [14]: This baseline adopts a classical DQN framework and considers both positive feedback and negative feedback. And a pairwise regularization term is used to maximize the difference of Q-values between positive items and negative items.
- **HRL**: This is a baseline with a hierarchical reinforcement learning structure like [15, 30]. For fairness, both the high-level agent and the low-level agent adopt the DQN framework. This is a representative RL-based recommendation method. This baseline also helps us evaluate the performance improvement achieved by introducing a hierarchical structure to the recommendation.
- **X+MT**: We add task tags to the state to adapt existing methods to the multi-behavior recommendation. The baseline HRL+MT (MTRL4Rec w/o M) is also a variant of our model whose modular network is replaced by fully connected layers. It allows us to evaluate the impact of our proposed modular network.
- **MTRL4Rec w/o R**: In this variant, we removed the routing network and set the MoUs' weights in the modular network equal. This enables us to evaluate the influence of task-specific routing and understand the significance of the routing network in our proposed MTRL4Rec model.
- **MTRL4Rec w/o H**: This variant is designed to evaluate the importance of the hierarchical reinforcement learning (HRL) structure in our model. In this case, the HRA is removed, and the users interact with the LRA directly.

4.3 Simulator

Due to the difficulty and huge cost of A/B tests, we train our method and RL-based baselines on a simulated environment as [14, 15]. The simulated environment is trained on users' logs. The simulator is a deep neural network. It outputs a one-hot vector that predicts the immediate feedback for a state-action pair $\langle s, a \rangle$ for every behavior task t. We tested the simulator on users' logs (not the data for training), and experimental results suggest that the simulated online environment can predict immediate feedback accurately and simulate the real online environment. This enables us to train and test our model on it. To get closer to reality, we let the simulator randomly requests on different behavior tasks according to the distribution of different behaviors in the dataset.

Moreover, the user randomly terminates the session with a probability of $1 - p_i$ in each step. If the user interacted with the last recommended item a_{i-1}, we set p_i as 1. Otherwise, we set p_i as $p_{i-1}d_p$, where d_p is a patience discount factor, a hyper-parameter between $0 - 1$. If the agent cannot recommend satisfactory items to users, the users may terminate the session earlier.

4.4 Offline Test (RQ1 and RQ2)

In the offline test, we re-rank the items from users' logs in the test set, according to items' Q-values output by the models. The target items will be ranked at the top of the ranked new list if a method works well. For the hierarchical RL algorithm, we use their LRA to re-rank these

Table 1 Performance comparisons among different models

Methods	Tmall				Tianchi			
	HR @5	NDCG	HR @10	NDCG	HR @5	NDCG	HR @10	NDCG
DQN	0.2718	0.1630	0.5363	0.2448	0.3150	0.1863	0.5965	0.2706
	0.2812	0.1666	0.5286	0.2458	0.3279	0.1990	0.6257	0.2830
	0.2947	0.1783	0.5376	0.2549	0.3302	0.2026	0.6032	0.2822
DQN+MT	0.2718	0.1609	0.5238	0.2384	0.3268	0.1940	0.5787	0.2711
	0.2746	0.1642	0.5286	0.2460	0.3279	0.1954	0.6175	0.2808
	0.2871	0.1740	0.5384	0.2463	0.3333	0.2070	0.6063	0.2887
DEERS	0.2714	0.1623	0.5318	0.2419	0.3248	0.1913	0.5768	0.2667
	0.2840	0.1692	0.5239	0.2447	0.3251	0.2044	0.5929	0.2937
	0.2796	0.1646	0.5225	0.2416	0.3238	0.1971	0.6063	0.2856
HRL	0.2784	0.1648	0.5290	0.2428	0.3248	0.2025	0.5906	0.2859
	0.2812	0.1663	0.5361	0.2458	0.3497	0.2118	0.6093	0.2925
	0.2913	0.1763	0.5451	0.2565	0.3556	0.2067	0.6222	0.2909
HRL+MT (MTRL4Rec w/o M)	0.2746	0.1629	0.5297	0.2436	0.3287	0.2036	0.6004	0.2813
	0.2826	0.1690	0.5380	0.2464	0.3361	0.2077	0.6230	0.2909
	0.2963	0.1768	0.5668	0.2625	0.3492	0.2183	0.6349	0.2923
MTRL4Rec w/o R	0.2812	0.1699	0.5349	0.2510	0.3425	0.2052	0.6201	0.2896
	0.2793	0.1678	0.5239	0.2450	0.3770	0.2270	0.6230	0.3015
	0.2796	0.1682	0.5384	0.2496	0.4000	0.2518	0.6667	0.3240
MTRL4Rec w/o H	0.2714	0.1590	0.5189	0.2372	0.3150	0.1870	0.5965	0.2670
	0.2666	0.1546	0.5206	0.2352	0.3087	0.1850	0.5656	0.2630
	0.3097	0.1859	0.5434	0.2603	0.3111	0.1907	0.5810	0.2740
MTRL4Rec	**0.3034**	**0.1824**	**0.5554**	**0.2623**	**0.3543**	**0.2230**	**0.6201**	**0.3011**
	0.3079	**0.1886**	**0.5515**	**0.2659**	**0.4153**	**0.2622**	**0.6803**	**0.3466**
	0.3222	**0.2028**	**0.5701**	**0.2811**	**0.4159**	**0.2559**	**0.6857**	**0.3331**
MTRL4Rec - DDQN	0.3179	0.1894	0.5738	0.2717	0.3609	0.2178	0.6204	0.2956
	0.3177	0.1930	0.5656	0.2706	0.3855	0.2361	0.6529	0.3081
	0.3289	0.2066	0.5676	0.2837	0.4254	0.2671	0.7048	0.3575

We evaluate our model and baselines on two public datasets. There are three lines for each method, corresponding to performance on three behavior tasks (clicking, purchasing, and adding items to shopping cart). Our experimental results show that our model outperforms most baseline methods on different tasks. The underlines indicate the performance metrics in which MTRL4Rec-DDQN is better than MTRL4Rec

items. We select Hit Ratio @K (HR@K) and Normalized Discounted Cumulative Gain @K (NDCG@K) [50] as metrics to measure the performance of methods, where $K \in \{5, 10\}$. The modular network has 6 layers in total, and each layer consists of 4 MoUs ($M = 4$). The learning rate in reinforcement learning is set to 0.01. Additionally, for the simulator we set the patience discount factor p_d to an appropriate 0.98. The experimental results are shown in Table 1.

We can find the following five points:

- Comparing the performance difference between X and X+MT, we find that X+MT shows better performances on some behaviors. This proves that recommendation tasks with different behaviors can benefit each other. However, the improvement is not significant

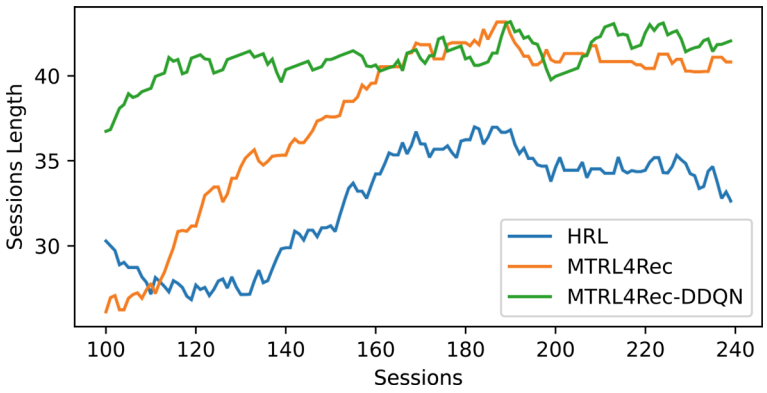

Figure 5 Online Test Comparison - Session Length. Data is smoothed by a sliding window

since the X+MTs cannot well model the commonalities and differences of different behavior tasks of users.

- Our MTRL4Rec method outperforms all the baselines on all tasks. It can better capture the similarities and differences between users' different behaviors. Our MTRL4Rec demonstrates outstanding capability in a multi-behavior recommendation scenario.
- The MTRL4Rec-DDQN method we further proposed can achieve excellent performance. The underlines in Table 1 indicate the greater performance metrics of MTRL4Rec-DDQN than MTRL4Rec. The overall performance of MTRL4Rec-DDQN is better than MTRL4Rec. The reason is that the DDQN-based algorithm can effectively avoid the overestimation of Q-values. It makes the model converge faster and perform better.
- The ablation tests (experiment on variants of MTRL4Rec) demonstrate that our model's hierarchical structure, modular network, and routing network are all meaningful for multi-behavior recommendation.
- On sparse behaviors, our MTRL4Rec method achieves larger performance gains. The reason is that the task on sparse behavior can benefit from other tasks, and our MTRL4Rec, with a modular network and a task routing network, has a better ability to capture the features of sparse behaviors.

4.5 Online Test (RQ3)

We do the online test in the simulated online environment. Due to limited space, we compare our methods, MTRL4Rec and MTRL4Rec-DDQN, with the best baseline in the previous offline test, HRL.

In the online test, we choose SessionLength and AverageReward as metrics to measure the performance of methods. SessionLength refers to the number of interactions between the agent and the environment in a session. Longer session length demonstrates that the model works better. This is because when the agent cannot recommend satisfactory items to the user, the user may have a greater probability of terminating the current session due to the patience mechanism. The second metric, AverageReward, is the average reward per step in a session, which is calculated according to TotalReward/SessionLength. TotalReward is the total accumulated reward obtained by the agent in the current session. It depicts the proportion of positive feedback the agent received after each action.

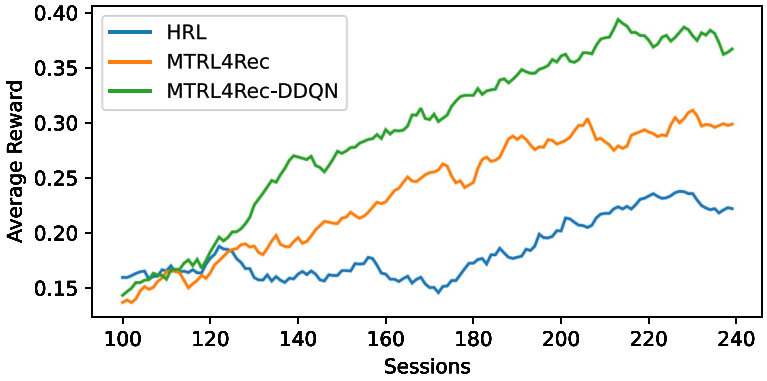

Figure 6 Online Test Comparison - Average Reward. Data is smoothed by a sliding window

Figures 5 and 6 show the online test results. Due to the instability of reinforcement learning at the beginning of a training, we make comparisons starting from session 100. For more intuition, the data in Figures 5 and 6 is smoothed by a sliding window as:

$$data_i' = avg([data_{i-k+1}, \ldots, data_i]). \tag{17}$$

Here we set $k = 30$.

We can find that the SessionLength of MTRL4Rec and MTRL4Rec-DDQN is greater than HRL, indirectly showing that they bring better customer satisfaction. In addition, our new MTRL4Rec-DDQN reaches a higher level earlier than MTRL4Rec. This proves that MTRL4Rec-DDQN has higher training efficiency and training stability. After several training sessions, MTRL4Rec and MTRL4Rec-DDQN reach the same high level of SessionLength.

The AverageRewards obtained by MTRL4Rec and MTRL4Rec-DDQN are greater than the baselines. Furthermore, they increase faster in the training. At the same time, MTRL4Rec-DDQN gets a better average reward than MTRL4Rec. This means that the addition of DDQN can indeed help the model to achieve greater improvement.

4.6 Parameter Analysis (RQ4)

Our method has two key parameters: the discount factor γ controls the weight of future reward, and M is the number of MoU in a layer of modular networks. To study the impact

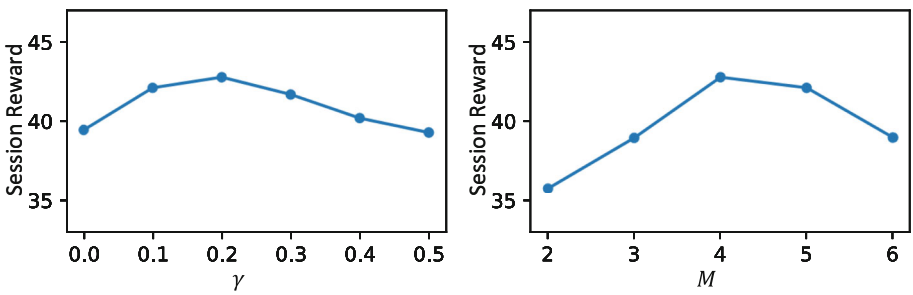

Figure 7 Results of parameter analysis

of these parameters, we investigate how the proposed framework works with the changes of one parameter while fixing other parameters.

Figure 7 shows the parameter sensitivity of our model to γ and M. The metric to measure the performance is the average cumulative reward in a number of consecutive sessions. The performance of MTRL4Rec achieves the peak when $\gamma = 0.2$. In other words, considering future reward indeed improves the performance of the model. However, it is less important than the current reward. A small γ also appears in other RL-based RS studies [3, 30].

In terms of the parametric analysis of M, MTRL4Rec has the best performance when $M = 4$. We analyze that the modular network with a small M does not have enough MoUs to capture the commonalities and differences between users' behaviors. When M is too large, there may be redundant and similar MoUs, which reduce the model performance. In future work, we may build a mechanism to force MoUs in the modular network to have low similarity to further improve the performance.

5 Conclusion

In this paper, we proposed a Multi-Task Reinforcement Learning model for multi-behavior Recommendation (MTRL4Rec), which gives different actions for users' different behaviors with a single policy. We treated the recommendation for different behaviors to users as different tasks and built a multi-task reinforcement learning based recommendation model. We use a modular network to capture the commonalities and differences between users' behaviors, and use a task routing network to generate routes in the modular network for each behavior task. Our MTRL4Rec model adopts a hierarchical structure for higher exploration and exploitation efficiency. An off-policy training algorithm and its further improved algorithm (MTRL4Rec-DDQN) are proposed. We demonstrate the effectiveness of the proposed MTRL4Rec and MTRL4Rec-DDQN in real-world e-commerce datasets and validate the importance of multi-task reinforcement learning for multi-behavior recommendation.

In future work, we can combine more RL frameworks (such as DDPG, SAC) with MTRL4Rec, and build a mechanism that forces MoUs to have low similarity to improve performance further. Moreover, self-supervised methods may improve the quality of embeddings in the model for better performance.

Acknowledgements This research was partially supported by the NSFC (62376180, 62176175), the major project of natural science research in Universities of Jiangsu Province (21KJA520004), Suzhou Science and Technology Development Program (SYC2022139), the Priority Academic Program Development of Jiangsu Higher Education Institutions and the Exploratory Self-selected Project of the State Key Laboratory of Software Development Environment.

Author Contributions Huiwang Zhang: Conceptualization, Methodology, Writing - Original Draft Pengpeng Zhao: Conceptualization, Methodology, Writing - Review & Editing Xuefeng Xian: Writing - Review & Editing Victor S. Sheng: Writing - Review & Editing Yongjing Hao: Conceptualization, Review & Editing Zhiming Cui: Writing - Review & Editing

Funding This research was partially supported by the NSFC (62376180, 62176175), the major project of natural science research in Universities of Jiangsu Province (21KJA520004), Suzhou Science and Technology Development Program (SYC2022139), the Priority Academic Program Development of Jiangsu Higher Education Institutions and the Exploratory Self-selected Project of the State Key Laboratory of Software Development Environment.

Availability of data and materials All the datasets are public available, and the links are attached in the footnotes.

Declarations

Ethical Approval Not Applicable

Competing Interests 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.

References

1. Cai, Q., Filos-Ratsikas, A., Tang, P., Zhang, Y.: Reinforcement mechanism design for e-commerce. In: WWW, pp. 1339–1348 (2018)
2. Takanobu, R., Zhuang, T., Huang, M., Feng, J., Tang, H., Zheng, B.: Aggregating e-commerce search results from heterogeneous sources via hierarchical reinforcement learning. In: WWW, pp. 1771–1781 (2019)
3. 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: WWW, pp. 167–176 (2018)
4. van den Oord, A., Dieleman, S., Schrauwen, B.: Deep content-based music recommendation. In: NIPS, pp. 2643–2651 (2013)
5. Hong, D., Li, Y., Dong, Q.: Nonintrusive-sensing and reinforcementlearning based adaptive personalized music recommendation. In: SIGIR, pp. 1721–1724 (2020)
6. Zhao, J., Zhao, P., Zhao, L., Liu, Y., Sheng, V.S., Zhou, X.: Variational self-attention network for sequential recommendation. In: ICDE, pp. 1559–1570 (2021)
7. Xu, C., Feng, J., Zhao, P., Zhuang, F., Wang, D., Liu, Y., Sheng, V.S.: Long- and short-term self-attention network for sequential recommendation. Neurocomputing **423**, 580–589 (2021)
8. Liu, J., Zhao, P., Zhuang, F., Liu, Y., Sheng, V.S., Xu, J., Zhou, X., Xiong, H.: Exploiting aesthetic preference in deep cross networks for crossdomain recommendation. In: WWW, pp. 2768–2774 (2020)
9. Xiao, K., Ye, Z., Zhang, L., Zhou, W., Ge, Y., Deng, Y.: Multi-user mobile sequential recommendation for route optimization. TKDD **14**(5), 52–15228 (2020)
10. Hidasi, B., Karatzoglou, A.: Recurrent neural networks with top-k gains for session-based recommendations. In: CIKM, pp. 843–852 (2018)
11. Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., Jiang, P.: Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In: CIKM, pp. 1441–1450 (2019)
12. Zhou, K., Wang, H., Zhao, W.X., Zhu, Y., Wang, S., Zhang, F., Wang, Z., Wen, J.: S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In: CIKM, pp. 1893–1902 (2020)
13. Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. J. Mach. Learn. Res. **6**, 1265–1295 (2005)
14. Zhao, X., Zhang, L., Ding, Z., Xia, L., Tang, J., Yin, D.: Recommendations with negative feedback via pairwise deep reinforcement learning. In: KDD, pp. 1040–1048 (2018)
15. Zhao, D., Zhang, L., Zhang, B., Zheng, L., Bao, Y., Yan, W.: Mahrl: Multi-goals abstraction based deep hierarchical reinforcement learning for recommendations. In: SIGIR, pp. 871–880 (2020)
16. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.: Neural collaborative filtering. In: WWW, pp. 173–182 (2017)
17. Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J., Xie, X.: Selfsupervised graph learning for recommendation. In: SIGIR, pp. 726–735 (2021)
18. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)
19. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI, pp. 2094–2100 (2016)
20. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)
21. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. IEEE Trans. Neural Networks **9**(5), 1054–1054 (1998)
22. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: ICLR (Poster) (2016)

23. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: ICML. Proceedings of machine learning research, vol. 80, pp. 1856–1865 (2018)
24. Afsar, M.M., Crump, T., Far, B.H.: Reinforcement learning based recommender systems: A survey. ACM Comput. Surv. **55**(7), 145–114538 (2023)
25. Zhang, J., Hao, B., Chen, B., Li, C., Chen, H., Sun, J.: Hierarchical reinforcement learning for course recommendation in moocs. In: AAAI, pp. 435–442 (2019)
26. Pei, C., Yang, X., Cui, Q., Lin, X., Sun, F., Jiang, P., Ou, W., Zhang, Y.: Value-aware recommendation based on reinforcement profit maximization. In: WWW, pp. 3123–3129 (2019)
27. Bai, X., Guan, J., Wang, H.: A model-based reinforcement learning with adversarial training for online recommendation. NeurIPS, 10734–10745 (2019)
28. Wang, P., Fan, Y., Xia, L., Zhao, W.X., Niu, S., Huang, J.: KERL: A knowledge-guided reinforcement learning model for sequential recommendation. In: SIGIR, pp. 209–218 (2020)
29. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discret. Event Dyn. Syst. **13**(1–2), 41–77 (2003)
30. Xie, R., Zhang, S., Wang, R., Xia, F., Lin, L.: Hierarchical reinforcement learning for integrated recommendation. In: AAAI, pp. 4521–4528 (2021)
31. Chen, M., Chang, B., Xu, C., Chi, E.H.: User response models to improve a REINFORCE recommender system. In: WSDM, pp. 121–129 (2021)
32. Xin, X., Karatzoglou, A., Arapakis, I., Jose, J.M.: Self-supervised reinforcement learning for recommender systems. In: SIGIR, pp. 931–940 (2020)
33. Wang, P., Fan, Y., Xia, L., Zhao, W.X., Niu, S., Huang, J.X.: KERL: A knowledge-guided reinforcement learning model for sequential recommendation. In: SIGIR, pp. 209–218 (2020)
34. Gao, C., Xu, K., Zhou, K., Li, L., Wang, X., Yuan, B., Zhao, P.: Value penalized q-learning for recommender systems. In: SIGIR, pp. 2008–2012 (2022)
35. Deng, Y., Li, Y., Sun, F., Ding, B., Lam, W.: Unified conversational recommendation policy learning via graph-based reinforcement learning. In: SIGIR, pp. 1431–1441 (2021)
36. Zhao, X., Xia, L., Zou, L., Liu, H., Yin, D., Tang, J.: Whole-chain recommendations. In: CIKM, pp. 1883–1891 (2020)
37. Caruana, R.: Multitask learning. Machine learning **28**(1), 41–75 (1997)
38. Pinto, L., Gupta, A.: Learning to push by grasping: Using multiple tasks for effective learning. In: ICRA, pp. 2161–2168 (2017)
39. Crawshaw, M.: Multi-task learning with deep neural networks: A survey. CoRR abs/2009.09796 (2020)
40. Strezoski, G., van Noord, N., Worring, M.: Many task learning with task routing. In: ICCV, pp. 1375–1384 (2019)
41. Liu, X., He, P., Chen, W., Gao, J.: Multi-task deep neural networks for natural language understanding. In: ACL (1), pp. 4487–4496 (2019)
42. Singh, S.P.: Transfer of learning by composing solutions of elemental sequential tasks. Mach. Learn. **8**, 323–339 (1992)
43. Wilson, A., Fern, A., Ray, S., Tadepalli, P.: Multi-task reinforcement learning: a hierarchical bayesian approach. In: ICML, vol. 227, pp. 1015–1022 (2007)
44. Pinto, L., Gupta, A.: Learning to push by grasping: Using multiple tasks for effective learning. In: ICRA, pp. 2161–2168 (2017)
45. Yang, R., Xu, H., Wu, Y., Wang, X.: Multi-task reinforcement learning with soft modularization. NeurIPS (2020)
46. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Comput. **3**(1), 79–87 (1991)
47. Ma, J., Zhao, Z., Yi, X., Chen, J., Hong, L., Chi, E.H.: Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In: KDD, pp. 1930–1939 (2018)
48. Tang, H., Liu, J., Zhao, M., Gong, X.: Progressive layered extraction (PLE): A novel multi-task learning (MTL) model for personalized recommendations. In: RecSys, pp. 269–278 (2020)
49. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)
50. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)

Authors and Affiliations

Huiwang Zhang[1] · Pengpeng Zhao[1] · Xuefeng Xian[2] · Victor S. Sheng[3] · Yongjing Hao[1] · Zhiming Cui[4]

Huiwang Zhang
hwzhangcs@stu.suda.edu.cn

Victor S. Sheng
Victor.Sheng@ttu.edu

Yongjing Hao
yjhaozb@stu.suda.edu.cn

Zhiming Cui
zmcui@mail.usts.edu.cn

1 School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu, China

2 Suzhou Vocational University, Suzhou, Jiangsu, China

3 Texas Tech University, Lubbock, Texas, USA

4 SuZhou University of Science and Technology, Suzhou, Jiangsu, China