A Text-based Deep Reinforcement Learning Framework for Interactive Recommendation

Chaoyang Wang[1] and **Zhiqiang Guo**[1] and **Jianjun Li**[1,2] and **Peng Pan**[1] and **Guohui Li**[1]

Abstract. Due to its nature of learning from dynamic interactions and planning for long-run performance, reinforcement learning (RL) recently has received much attention in interactive recommender systems (IRSs). IRSs usually face the large discrete action space problem, which makes most of the existing RL-based recommendation methods inefficient. Moreover, data sparsity is another challenging problem that most IRSs are confronted with. While the textual information like reviews and descriptions is less sensitive to sparsity, existing RL-based recommendation methods either neglect or are not suitable for incorporating textual information. To address these two problems, in this paper, we propose TDDPG-Rec, a Text-based Deep Deterministic Policy Gradient framework for interactive recommendation. Specifically, we leverage textual information to map items and users into a feature space, which greatly alleviates the sparsity problem. Moreover, we design an effective method to construct an action candidate set. By the policy vector dynamically learned from TDDPG-Rec that expresses the user's preference, we can select actions from the candidate set effectively. Through extensive experiments on three public datasets, we demonstrate that TDDPG-Rec achieves state-of-the-art performance over several baselines in a time-efficient manner.

1 Introduction

In the era of information explosion, recommender systems play a critical role in alleviating the information overload problem. Recently, interactive recommender system (IRS) [31], which continuously recommends items to individual users and receives their feedbacks to refine its recommendation policy, has received much attention and plays an important role in personalized services, such as Tik Tok, Pandora, and YouTube.

In the past few years, there have been some attempts to address the interactive recommendation problem by modeling the recommendation process as a multi-armed bandit (MAB) problem [16, 31, 24], but these methods are not designed for long-term planning explicitly, which makes their performance unsatisfactory [4]. It is well recognized that reinforcement learning (RL) performs excellently in finding policies on interactive long-running tasks, such as playing computer games [19] and solving simulated physics problems [17]. Therefore, it is natural to introduce RL to model the interactive recommendation process. In fact, recently there have been some works on applying RL to address the interactive recommendation problem [32, 28, 29, 10]. However, most of the existing RL-based meth-

ods, including all Deep Q-learning Network (DQN) based methods [32, 29, 5] and most Deep Deterministic Policy Gradient (DDPG) based methods [10, 28], suffer from the problem of making a decision in linear time complexity with respect to the size of the action space, i.e., the number of available items, which makes them inefficient (or unscalable) when the action space size is large.

To improve efficiency, based on DDPG, Dulac-Arnold *et al.* [9] proposed to first learn an action representation (vector) in a continuous hidden space, and then find the valid item by using a k nearest neighbor search method. However, such a method ignores the importance of each dimension in the action vector. Moreover, it still needs to find the k nearest-neighbors from the whole action space, which is time-consuming. Recently, Chen *et al.* [4] proposed a tree-structured policy gradient recommendation (TPGR) framework, within which a balanced hierarchical clustering tree is built over the items. Then, picking an item is formulated as seeking a path from the root to a certain leaf in the tree, which dramatically reduces the time complexity. But this method introduces the burden of building a clustering tree, especially when new items appear frequently, the tree needs to be reconstructed and this may cost a lot.

On the other hand, most exiting RL-based recommendation methods use the past interaction data, such as ratings, purchase logs, or viewing history, to model user preferences and item features [9, 30, 4]. A major limitation of such kind of methods is that they may suffer serious performance degradation when facing the data sparsity problem, which is very common in real-world recommendation systems. As well known, textual information like reviews by users and item descriptions provided by suppliers contains more knowledge than interaction data, and is less sensitive to data sparsity. Nowadays, textual information has been readily available in many e-commerce and review websites, such as *Amazon* and *Yelp*. Thanks to the invention of word embedding, applying textual information for recommendation is possible, and there have been some successful attempts in conventional recommender systems [33, 3, 7]. But for IRS, existing RL-based methods either neglect to leverage textual information, or are not suitable for incorporating textual information due to their unique structures for processing rating sequence.

In this paper, we propose a Text-based Deep Deterministic Policy Gradient framework for IRSs (TDDPG-Rec). Specifically, we utilize textual information and pre-trained word vectors [20] to embed items and users into a continuous feature space, which, to a great extent, alleviates the data sparsity problem. Then we classify users into several clusters by the K-means algorithm [1]. Next, based on the thought of collaborative filtering, we construct an action candidate set, which consists of positive, negative and ordinary items that are selected based on the user's historical logs and classification results. Afterwards, we use a policy vector, which is dynamically learned

---

[1] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, 430074. Email: {sunwardtree, georgeguo, jianjunli, panpeng, guohuili}@hust.edu.cn
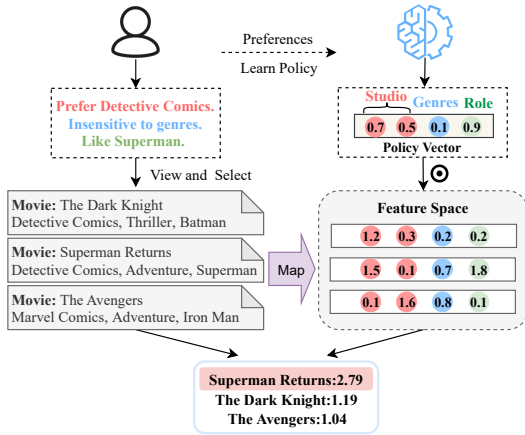
[2] Corresponding author.

Figure 1: An example for illustrating policy vector.

from the actor part of TDDPG-Rec, to express the user's preference in the feature space. Finally, we use the policy vector to select items from the candidate set to form the action for recommendation.

Figure 1 gives an example for helping understand the policy vector. Suppose a user selects a movie according to the preference that can be represented as explicit policies such as *Prefer Detective Comics*, *Insensitive to genres* and *Like Superman*. By our method, a policy vector in the feature space, e.g., $(0.7, 0, 5, 0.1, 0.9)$, can be learned, where the value of each dimension represents how much emphasis this user gives on the dimension in the feature space. By conducting a dot product between the policy vector and the item vectors, we finally can choose the movie *Superman Returns* with the highest score of 2.79 for recommendation (assume Top-1 recommendation here).

Moreover, since it is too expensive to train and test our model in an online manner, we build an environment simulator to mimic online environments with principles derived from real-world data. Through extensive experiments on several real-world datasets with different settings, we demonstrate that TDDPG-Rec achieves high efficiency and remarkable performance improvement over several state-of-the-art baselines, especially for large-scale high-sparsity datasets. To sum up, our main contributions of this work are as follows:

- By utilizing textual information and pre-trained word vectors, we embed items and users into a continuous feature space to reduce the negative influence of rating sparsity.
- We express the preferences of users by implicit policy vectors and propose a method based on DDPG to learn the policy vectors dynamically. Moreover, based on the thought of collaborative filtering, we classify users into several clusters and build the candidate set. The policy vector, combining with the candidate set, is used to select items that form an action, which reduces the scale of action space effectively.
- Extensive experiments are conducted on three benchmark datasets and the results verify the high efficiency and superior performance of TDDPG-Rec over state-of-the-art methods.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 formally defines the research problem and details the proposed TDDPG-Rec model, as well as the corresponding learning algorithm; Section 4 presents and analyzes the experimental results; Finally, Section 5 concludes the paper with some remarks.

2 Related Work

2.1 RL-based Recommendation Methods

RL-based recommendation methods usually formulate the recommendation procedure as a Markov Decision Process (MDP). They explicitly model the dynamic user's status and plan for long-run performance [22, 32, 10, 28, 29, 5, 27]. As mentioned earlier, most existing RL-based methods suffer from the large-scale discrete action space problem.

To address the large-scale discrete action space problem in IRS, there are some good attempts. Dulac-Arnold *et al.* [9] proposed to leverage prior information about the actions to embed them in a continuous space to generate a proto-action, and then find a set of discrete actions closest to the proto-action as the candidate in logarithmic time via a k nearest neighbor search. This method ignores the negative influences of the dimensions that users do not care about, which makes it fail to find proper actions sometimes. Moreover, the k nearest-neighbor search needs to be conducted on the whole action space, which still surfers a high runtime overhead. Zhao *et al.* [30] used the actor-part of the Actor-Critic network to gain k weight vectors, each of which can pick up a maximum-score item from the remaining items. But the relationship of these vectors is blurry, which causes the order of the k items cannot be explained. Based on DPG, Chen *et al.* [4] proposed a tree-structured policy gradient recommendation (TPGR) framework. In TPGR, a balanced hierarchical clustering tree is built over all the items. Then, making a decision can be formulated as seeking a path from the root to a certain leaf in the clustering tree, which reduces the time complexity significantly. But this method can only support Top-1 recommendation. Moreover, when new items appear frequently, the clustering tree needs to be reconstructed, which incurs extra cost.

2.2 Textual Information for Recommendation

Most of the recommendation models (including RL-based ones) that merely exploit interaction matrix usually face the data sparsity problem. A large amount of knowledge in the textual information can potentially alleviate the data sparsity problem [33]. The development of deep learning in natural language processing makes it possible for using textual information that human beings can understand to enhance the recommendation performance [33, 3, 7, 6]. Reviews and descriptions are the most important textual information in recommender systems. The reviews, which contain users' attitudes, and the descriptions, which contain items' advantages, along with the ratings, can show the preferences of users. There are works that use sentiment analysis [3], convolutional neural networks [33, 8] and pre-trained word vectors on large corpora [7], to get vectors from textual information. These vectors are then incorporated into the proposed model to improve recommendation performance.

IRS also suffers from the rating sparsity problem, but so far, we are not aware of any recommendation method for IRS that utilizes textual information. Most existing RL-based methods for IRS either neglect to incorporate with textual information, or have difficulty in utilizing textual information, since they use rating sequence, which has time-related structures, as the input of their model [32, 4]. Note that in the domain of conversational recommender system (CRS), Basile *et al.* [2] proposed a framework that combines deep learning and reinforcement learning and uses text-based features to provide relevant recommendations and produce meaningful dialogues. But different from CRS, in our RL-based method for IRS, the textual

information is utilized to learn the implicit long-term preferences, not the proactive immediate needs of users.

3 Proposed Method

3.1 Problem Formulation

We consider an interactive recommendation system with N users $\{u_1, u_2, \ldots, u_N\}$ and M items $\{v_1, v_2, \ldots, v_M\}$, and use $Y \in \mathbb{R}^{N \times M}$ to denote the rating matrix, where $y_{i,j}$ is the rating of user u_i on item v_j. This kind of interactive Top-k recommendation process can be modeled as a special Markov Decision Process (MDP), where the key components are defined as follows.

- **State.** Use S to denote the state space. A state $s \in \mathrm{S}$ is defined as the possible interaction between a user and the recommender system, which can be represented by n_s item vectors.
- **Action.** Use A to denote the action space. An action $a \in \mathrm{A}$ contains n_a ordered items, each of which is represented by a vector, for recommendation.
- **Reward function.** After receiving an action a at state s, our environment simulator returns a reward r, which reflects the user's feedback to the recommended items. We use $\mathcal{R}(s, a)$ to denote the reward function.
- **Transition.** In our model, since the state is a set of item vectors, once the action is determined and the user's feedback is given, the state transition is also determined.

Consider an agent that interacts with the environment E in discrete timesteps. At each timestep t, the agent can receive a state s_t by observing the current environment, then it takes an action a_t and gets a reward r_t. An agent's behavior is defined by a policy π, which maps states to a probability distribution over the action, i.e., $\pi : \mathrm{S} \to \mathcal{P}(\mathrm{A})$. Based on the above notations, we can define the instantiated MDP for our recommendation problem, $\mathcal{M} = \langle \mathrm{S}, \mathrm{A}, \mathcal{R}, \mathcal{P}, T, \gamma \rangle$, where T is the maximal decision step, and γ is the discount factor. Our objective in this work is to learn a policy π that maximizes the expected discounted cumulative reward.

3.2 Framework Overview

Figure 2 gives an overview of our framework, which contains two major steps: data preparation and training. In data preparation, we first embed items to get item vectors by leveraging textual information. Based on the derived item vectors and users' historical logs, we can embed users into the same feature space. Next, we classify the users into several clusters by K-means. In the training phase, we train a unique model for each cluster, with the objective of implementing a more personalized recommendation. Take cluster 2 for an example, we randomly select a user u_i from it. Based on the historical log of u_i and the user classification results, we sample positive, negative and ordinary items for u_i to construct a candidate set, which later will be used in the reinforcement model for action selection. Our reinforcement model is based on DDPG, which interacts with the simulator that is based on historical logs to learn the inner relationship among all possible states and actions. The training phase will stop when the model loss reaches stable.

3.3 Embedding with Textual Information

Textual information like descriptions and reviews is important for decision making, we build vectors based on them. Item vectors are cal-
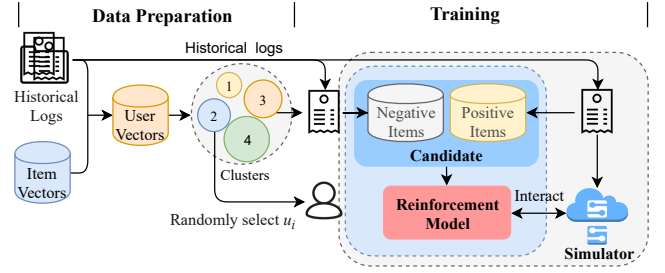

Figure 2: Framework overview.

culated by the word vectors from GloVe.6B[3] (trained on Wikipedia 2014 and Gigaword 5). Note that the descriptions and reviews contain many meaningless words, we remove them in advance by comparison with the Long Stopword List[4]. Using $\mathbf{u}_i$ and $\mathbf{v}_j$ to denote the vectors of representing user u_i and item v_j, respectively, the item vector $\mathbf{v}_j$ can be computed by,

$$\mathbf{v}_j = \frac{1}{n_d} \sum_{p=1}^{n_d} \mathbf{w}_p^D + \frac{1}{n_r} \sum_{q=1}^{n_r} \mathbf{w}_q^R \tag{1}$$

where $\mathbf{w}_p^D$ and $\mathbf{w}_q^R$ are the vectors of the words from descriptions and reviews, respectively, and n_d and n_r denote the corresponding numbers of them. The word vectors with similar semantics have closer Euclidean distance than the word vectors with large semantics differences [20], which ensures that items with similar reviews and descriptions are closer to each other.

Given a user u_i and one of its historical logs, if the corresponding rating is greater than a given bound y_b (e.g., $y_b = 2$ in a rating system with the highest rating 5), then the log is regarded as positive; Otherwise, it is negative. We use $\mathcal{V}_{u_i}^p$ and $\mathcal{V}_{u_i}^n$ to denote the set of items that are in u_i's positive and negative historical logs, respectively. After obtaining all the item vectors, we can calculate user vector $\mathbf{u}_i$ by normalizing the summation of the items' vectors that appear in $\mathcal{V}_{u_i}^p$, i.e.,

$$\mathbf{u}_i = \frac{1}{n_v} \sum_{l=1}^{n_v} \mathbf{v}_l \tag{2}$$

where n_v denotes the number of items in $\mathcal{V}_{u_i}^p$. In this way, we embed users and items in a same feature space.

3.4 Construction of the Candidate Set

In Top-k recommendation, the state is defined as a set of n_s items. So there are a total of $A_{M-n_s}^k$ (note here $A_{M-n_s}^k$ is a permutation) actions that can be chosen as an action. With the increase of the number of items (M), the scale of the action space will increase rapidly. Based on the assumption that the preferences can be obtained by a set of items that users like and dislike, we pick up the positive and negative items to build a candidate set c. Additionally, to maintain generalization, we add some ordinary items in the candidate set.

For user u_i, we sample positive items from $\mathcal{V}_{u_i}^p$, negative items from $\mathcal{V}_{u_i}^n$, and ordinary items by random. Since users usually skip the items that they do not like, the negative items in $\mathcal{V}_{u_i}^n$ are rare [18]. Based on the idea of collaborative filtering, i.e., the more differences between two users, the more possible that one user's likes are the other's dislikes, we classify users into several clusters by K-means [1] to supplement negative items. Specifically, we denote the

---

[3] http://nlp.stanford.edu/data/glove.6B.zip
[4] https://www.ranks.nl/stopwords

Algorithm 1: Candidate set construction for u_i

Input: n_c, α, $\mathcal{V}_{u_i}^p$, $\mathcal{V}_{u_i}^n$, $\mathcal{V}_{cl_l}^p$ and $\mathcal{V}_{cl_l^f}^p$.

Output: Candidate set c.

1 Initialize $c = \emptyset$, $n_{pos} = \lfloor n_c \times \alpha \rfloor$;
2 **if** $n_{pos} \leq |\mathcal{V}_{u_i}^p|$ **then**
3 $\quad$ $c \leftarrow$ randomly select n_{pos} items from $\mathcal{V}_{u_i}^p$;
4 **else**
5 $\quad$ $n_{pos} = |\mathcal{V}_{u_i}^p|$; $\quad c \leftarrow \mathcal{V}_{u_i}^p$;
6 **end**
7 $n_{neg} = \lfloor (n_c - n_{pos})/2 \rfloor$;
8 **if** $n_{neg} \leq |\mathcal{V}_{u_i}^n|$ **then**
9 $\quad$ $c \leftarrow c \cup$ randomly select n_{neg} items from $\mathcal{V}_{u_i}^n$;
10 **else**
11 $\quad$ $n_{neg} = n_{neg} - |\mathcal{V}_{u_i}^n|$; $\quad c \leftarrow c \cup \mathcal{V}_{u_i}^n$;
12 $\quad$ $\mathcal{V}_{neg} \leftarrow \mathcal{V}_{cl_l^f}^p - (\mathcal{V}_{cl_l^f}^p \cap (\mathcal{V}_{cl_l}^p \cup \mathcal{V}_{u_i}^n))$;
13 $\quad$ **if** $n_{neg} \leq |\mathcal{V}_{neg}|$ **then**
14 $\quad\quad$ $c \leftarrow c \cup$ randomly select n_{neg} items from $\mathcal{V}_{neg}$;
15 $\quad$ **else**
16 $\quad\quad$ $c \leftarrow c \cup \mathcal{V}_{neg}$;
17 $\quad$ **end**
18 **end**
19 $n_{ord} = n_c - |c|$;
20 $c \leftarrow c \cup$ randomly select n_{ord} items not in c;
21 **return** c;

---

set of items that appear in the positive historical logs of users in cluster l as $\mathcal{V}_{cl_l}^p$ (user u_i belongs to cluster cl_l), and use cl_l^f to denote the cluster that has the farthest distance from the current cluster cl_l. If the negative items in $\mathcal{V}_{u_i}^n$ are not enough, the rest negative items will be selected from $\mathcal{V}_{cl_l^f}^p - (\mathcal{V}_{cl_l^f}^p \cap (\mathcal{V}_{cl_l}^p \cup \mathcal{V}_{u_i}^n))$. In this way, we can reduce the scale of the action space from $M - n_s$ to n_c, where n_c is the number of items in the candidate set c.

Algorithm 1 shows the detail of the construction for the candidate set, in which the positive items account for no more than α percent (line 1), and the negative and ordinary items each share 50% of the remaining part of n_c (line 7). In the training phase, since constructing a candidate set only contains some simple operations, such as randomly select and merge, and the candidate set size of our model is always fixed, it is not difficult to see that it has constant time complexity.

3.5 Architecture of TDDPG-Rec

The goal of a typical reinforcement learning model is to learn a policy π that can maximize the discounted future reward, i.e., the Q-value, which is usually estimated by the state-action value function $Q^\pi(s_t, a_t)$. Combined with deep neural networks, there are many algorithms that try to approximate $Q^\pi(s_t, a_t)$. Among them, DDPG, a model-free, off-policy actor-critic algorithm, which combines the advantages of DQN [19] and DPG [21], can concurrently learn policy and $Q^\pi(s_t, a_t)$ in high-dimensional, continuous action spaces by using neural network function approximation [17]. We use DDPG in our model, and Figure 3 shows the architecture of it.

In each timestep t, the actor network takes a state s_t as input. By a multiple-layer perceptron (MLP) network, we can learn a continuous vector, which we term as the policy vector, denoted by $\mathbf{p}_t$. The critic network takes state s_t and policy vector $\mathbf{p}_t$ as input. By an MLP, it can learn the current Q-value to evaluate $\mathbf{p}_t$. As mentioned in Figure 1, $\mathbf{p}_t$ represents a user's preferences in the feature vector space, it is
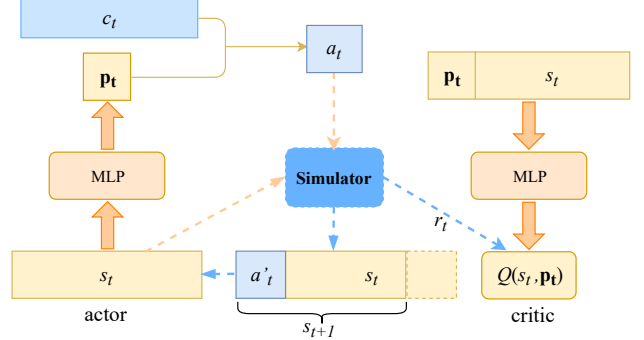

Figure 3: The structure of TDDPG-Rec.

a continuous weight vector that can measure the importance of each dimension. Combining with the candidate set c_t, we can get n_a items with the highest score, each of which is denoted by $Score(v_i)$ and,

$$Score(v_i) = \mathbf{p}_t^T \mathbf{v}_i \tag{3}$$

Moreover, to cover the action space to a large extent, the candidate set is randomly generated at each time step.

Note that the actions in IRSs are discrete. In our method, when embedding the items, we have mapped the discrete actions into a continuous feature space, where each item is represented by a feature vector. Then, by conducting the dot product between $\mathbf{p}_t$ and the item vectors in c_t, we can select the actions from a discrete space. In this way, our method can overcome the gap between discrete actions in IRSs and continuous actions in DDPG.

3.6 Environment Simulator

The same as several previous work [25, 4], we build an environment simulator to mimic online environments. It receives the present state s_t and action a_t, then returns reward r_t and the next state s_{t+1}. In our model, the reward function guides the model to capture users' preferences and evaluate the rank quality of the recommended items. For user u_i in time-step t, we give a reward r_t on a_t gained by $\mathbf{p}_t$ among the candidate set c_t. The reward r_t is determined by two values, w_k and $r_{i,j}$, specifically,

$$r_t = \mathcal{R}(s_t, a_t) = \sum_{k=1}^{n_a} w_k \times r_{i,j} \tag{4}$$

where n_a is the number of items in a_t, w_k is the ranking weight of the items in a_t, and $r_{i,j}$ is the reward of item v_j for user u_i. Inspired by DCG [11, 25], the ranking weight is calculated by,

$$w_k = 1/\log_2(k+1) \tag{5}$$

To give proper rewards for different types of items, $r_{i,j}$ is designed as follows,

$$r_{i,j} = \begin{cases} y_{i,j} - y_b, & \text{if } v_j \in \mathcal{V}_{u_i}^p; \\ y_{i,j} - y_b - 1, & \text{if } v_j \in \mathcal{V}_{u_i}^n; \\ -0.5, & \text{if } v_j \in \mathcal{V}_{cl_l^f}^p - (\mathcal{V}_{cl_l^f}^p \cap (\mathcal{V}_{cl_l}^p \cup \mathcal{V}_{u_i}^n)); \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Recall here $y_{i,j}$ is the rating of user u_i on item v_j, and y_b is the rating bound to determine whether the corresponding log is regarded as positive or negative. By this formula, positive items in a_t will get positive feedback, and negative items will get negative feedback.

Algorithm 2: Learning TDDPG-Rec

Input: The maximum step T; The number of user clusters n_{cl}; Target network update rate τ.

Output: Model parameters θ^Q, θ^μ, $\theta^{Q'}$ and $\theta^{\mu'}$.

1 **for** $l = 1$ *to* n_{cl} **do**
2   Randomly initialize critic network $Q\left(s, \mathbf{p}|\theta^Q\right)$ and actor network $\mu\left(s|\theta^\mu\right)$;
3   Initialize target network: $\theta_l^{Q'} = \theta_l^Q$, $\theta_l^{\mu'} = \theta_l^\mu$;
4   Initialize replay buffer B;
5   **repeat**
6   Randomly select u_j in cluster cl_l;
7   Initialize observation state s_1;
8   Initialize a random process $\mathcal{N}$ for exploration;
9   **for** $t = 1$ *to* T **do**
10   Construct c_t for u_j by Algorithm 1;
11             $\mathbf{p}_t = \mu\left(s_t|\theta^\mu\right) + \mathcal{N}_t$;
12   Use $\mathbf{p}_t$ to select items from c_t as a_t;
13   Interact with simulator by a_t;
14   Observe r_t and s_{t+1} from simulator;
15   Store transition $(s_t; \mathbf{p}_t; r_t; s_{t+1})$ in B;
16   Sample a random batch of N_b transitions $(s_i; \mathbf{p}_i; r_i; s_{i+1})$ from B;
17   Update critic by minimizing Equation (7);
18   Update actor by Equation (9);
19   Update the target networks:
                $\theta^{Q'} = \tau\theta^Q + (1 - \tau)\theta^{Q'}$,
                $\theta^{\mu'} = \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$;
20   **end**
21   **until** *converge*;
22 **end**
23 return θ^Q, θ^μ, $\theta^{Q'}$ and $\theta^{\mu'}$

Moreover, the supplemented negative items will get half of the minimum negative feedback, i.e., -0.5, while the other items will get a feedback of 0.

As shown in Figure 3, our method of generating s_{t+1} is in a sliding-window manner. Specifically, among ordered items in a_t, we keep the order and select the items that are not in s_t as a_t'. Then we put a_t' at the head of s_t, and select the top n_s items as s_{t+1}.

3.7   Learning TDDPG-Rec

The training phase (as shown in Algorithm 2) learns model parameters θ^Q, θ^μ, $\theta^{Q'}$, and $\theta^{\mu'}$ through maximizing the cumulated discounted rewards of all the decisions. Based on the assumption that similar users have similar preferences, our method classifies users and trains models for each cluster. At the beginning of the training phase, we randomly initialize the network parameters and the replay buffer B. In order for action exploration, we initialize a random process $\mathcal{N}$, adding some uncertainty when generating $\mathbf{p}$. The critic network focuses on minimizing the gap between the current Q-value $Q(s_i, \mathbf{p}_i|\theta^Q)$ and the expected Q-value y_i, which is achieved by minimizing the following loss,

$$L(\theta^Q) = \frac{1}{N_b} \sum_i (z_i - Q(s_i, \mathbf{p}_i|\theta^Q))^2 \qquad (7)$$

where z_i can be expressed in a recursive manner by using the Bellman equation,

$$z_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \qquad (8)$$

The objective of the actor network is to optimize the policy vector $\mathbf{p}$, through maximizing the Q-value. The actor network is trained by the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N_b} \sum_i \nabla_{\mathbf{p}} Q(s, \mathbf{p}|\theta^Q)|_{s=s_i, \mathbf{p}=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i} \quad (9)$$

Note that in our implementation, we set the minimum and maximum training step thresholds based on the size of buffer B. When the number of steps is greater than the minimum threshold and the loss remains stable, or the number of steps is greater than the maximum threshold, the training phase will stop.

4   Experiments and Results

In this section, we conduct experiments to demonstrate the effectiveness of the proposed TDDPG-Rec model versus several state-of-the-art models. We first introduce the experimental setup, and then present and discuss the experimental results from the perspective of both recommendation performance and time efficiency. Finally, we conduct the hype-parameter sensitivity analysis in the last part of this section. We have implemented our models based on Tensorflow, which can be accessed in GitHub[5].

4.1   Experimental Settings

4.1.1   Datasets

Jure Leskovec *et al.* [15] collected and categorized a variety of *Amazon* products and built several datasets[6] including ratings, descriptions, and reviews. We evaluate our models on three publicly available *Amazon* datasets: Digital Music (**Music** for short), **Beauty** and Clothing Shoes and Jewelry (**Clothing** for short), which all have at least 5 reviews for each product. Table 1 shows the statistical details of the datasets we used.

Table 1: Statistics of datasets.

DataSet	#Users	#Items	#Ratings of Pos & Neg		Sparsity	Size of Textual Info
Music	5,541	3,568	58,905	5,801	0.9967	68,096 KB
Beauty	22,363	12,101	176,520	21,982	0.9993	88,986 KB
Clothing	39,387	23,033	252,022	26,655	0.9997	84,168 KB

4.1.2   Baseline methods

We compare TDDPG-Rec with 10 methods, where ItemPop is a conventional recommendation method, LinearUCB is a MAB-based method, DMF is an MF-based method with neural network, ANR is a neural recommendation method that leverages textual information, Caser and SASRec are time-related deep learning based methods, DDPG-KNN, TPGR, TDQN-Rec and MDDPG-Rec are all RL-based methods.

- **ItemPop** recommends the most popular items (i.e., the item with the highest average rating) from currently available items to the user at each timestep. This method is non-personalized and is often used as a benchmark for recommendation tasks.
- **LinearUCB** [16] is a contextual-bandit recommendation approach that adopts a linear model to estimate the upper confidence bound for each arm.

Table 2: Overall interactive recommendation performance. Best performance is in boldface and second best is underlined.

Method	Music						Beauty						Clothing					
	HR@10	F1@10	nDCG@10	HR@20	F1@20	nDCG@20	HR@10	F1@10	nDCG@10	HR@20	F1@20	nDCG@20	HR@10	F1@10	nDCG@10	HR@20	F1@20	nDCG@20
ItemPop	0.2447	0.0454	0.1101	0.4889	0.0525	0.1716	0.2551	0.0482	0.1134	0.5278	0.0543	0.1817	0.2265	0.0413	0.1033	0.4964	0.0482	0.1706
LinerUCB	0.3318	0.0621	0.1569	0.5885	0.0626	0.2210	0.2734	0.0502	0.1249	0.5273	0.0529	0.1885	0.2393	0.0437	0.1041	0.5044	0.0488	0.1704
DMF	0.3201	0.0631	0.1462	0.5747	0.0638	0.2095	0.3219	0.0614	0.1447	0.5911	0.0613	0.2122	0.2500	0.0458	0.1130	0.5041	0.0489	0.1756
ANR	0.4980	0.1128	0.2756	0.7097	0.1084	0.3252	0.4550	0.0990	0.2252	0.6993	0.1006	0.2850	0.3421	0.0663	0.1622	0.6008	0.0659	0.2264
Caser	0.8097	0.1676	0.5351	0.9090	0.1048	0.5542	0.6125	0.1218	0.3939	0.7817	0.0826	0.4344	0.5060	0.0934	0.2900	0.7196	0.0702	0.3427
SASRec	0.8897	0.1910	0.6212	0.9635	**0.1151**	0.6325	0.6823	0.1386	0.4569	0.8330	0.0907	0.4942	0.5817	0.1084	0.3525	0.7655	0.0758	0.3968
DDPG-kNN (k=0.1M)	0.3274	0.0648	0.1527	0.5838	0.0647	0.2171	0.2585	0.0489	0.1170	0.5142	0.0529	0.1809	0.2541	0.0467	0.1131	0.5043	0.0490	0.1757
DDPG-kNN (k=M)	0.3436	0.0692	0.1617	0.6001	0.0676	0.2258	0.2734	0.0522	0.1274	0.5197	0.0539	0.1889	0.2768	0.0510	0.1242	0.5293	0.0517	0.1874
TDQN-Rec	0.8150	0.1712	0.5048	0.8977	0.1053	0.5251	0.5466	0.1053	0.3033	0.7662	0.0796	0.3573	0.3739	0.0685	0.1873	0.6252	0.0607	0.2501
MDDPG-Rec	0.8293	0.1803	0.5293	0.9074	0.1087	0.5477	0.5985	0.1202	0.3391	0.7740	0.0825	0.3830	0.3094	0.0567	0.1487	0.5595	0.0542	0.2111
TDDPG-Rec	**0.9164**	**0.2032**	**0.6630**	0.9426	0.1142	**0.6687**	**0.7648**	**0.1517**	**0.4942**	**0.8952**	**0.0948**	**0.5261**	**0.6237**	**0.1203**	**0.3553**	**0.8210**	**0.0818**	**0.4055**

- **DMF** [26] is a state-of-the-art matrix factorization model using deep neural networks. Specifically, it utilizes two distinct MLPs to map the users and items into a common low-dimensional space with non-linear projections.
- **ANR** [7] uses an attention mechanism to focus on the relevant parts of reviews and estimates aspect-level user and item importance in a joint manner.
- **Caser** [23] embeds a sequence of recent items into an image and learn sequential patterns as local features of the image by using convolutional filters.
- **SASRec** [12] is a self-attention based sequential model for next item recommendation. It models the entire user sequence and adaptively considers consumed items for prediction.
- **DDPG-kNN** [9] addresses the large discrete action space problem by combining DDPG with an approximate kNN method.
- **TPGR** [4] builds a balanced hierarchical clustering tree and formulates picking an item as seeking a path from the root to a certain leaf of the tree.
- **TDQN-Rec** is a method that replaces DDPG in TDDPG-Rec with DQN, while retains other components the same as that in TDDPG-Rec.
- **MDDPG-Rec** is a method that uses the same framework as TDDPG-Rec, but with vectors being derived by matrix factorization [13], rather than leveraging textual information.

Note that for DDPG-kNN, larger k (i.e., the number of nearest neighbors) will result in better performance but poor efficiency. For a fair comparison, we consider setting k as $0.1M$ and M (recall that M is the number of items) respectively.

4.1.3 Evaluation Metrics and Methodology

The methods that achieve their goals by Top-k recommendation take evaluation on the indexes like Hit Ratio (HR) [26], Precision [32, 23], Recall [23], F1 [4] and normalized Discounted Cumulative Gain (nDCG) [25, 30, 32, 12]. To cover as many aspects of Top-k recommendation as possible, we chose HR@k, F1@k, and nDCG@k as the evaluation metrics.

The test data was constructed in data preparation, and all the evaluated methods were tested by using this data. We now describe the test method in detail: For each user, we first classify user's history logs into positive and negative ones, and sort the items in positive history logs by time-stamp. Then, we choose the last 10% of the ordered items in the positive logs as positive items. Finally, the negative items are randomly selected from the cluster that is farthest from the one that the current user belongs to. Based on such a strategy, the recommendation methods (except TPGR, which only recommends one item in each episode) can generate a ranked Top-k list to evaluate the metrics mentioned above.

4.2 Results and Analysis

Table 2 shows the summarized results of our experiments on the three datasets in terms of six metrics including HR@10, F1@10, nDCG@10, HR@20, F1@20 and nDCG@20. Note that since TPGR is not suitable for Top-k recommendation, we did not include it as a competitor when evaluating the recommendation performance. From the results, we have the following key observations:

- The proposed model TDDPG-Rec achieves the best (or the second-best, but with a small gap to the best one) performance and obtains remarkable improvements over the state-of-the-art methods. Moreover, the performance improvement increases along with the increase of data scale and data sparsity, where the datasets are arranged in increasing order of scale and sparsity. This justifies the effectiveness of TDDPG-Rec that leverages textual information in RL-based recommendation, especially for large-scale and high-sparsity datasets.
- The structure of ANR is similar to that of DMF, while the structure of TDDPG-Rec is the same as that of MDDPG-Rec. Text-based methods ANR and TDDPG-Rec consistently outperform their counterparts, DMF and MDDPG-Rec, which only use interaction information for embedding. This demonstrates the importance of utilizing textual information to alleviate the negative effects of data sparsity for better performance.

Moreover, based on the results in Table 2, we conduct several statistical significance tests [14], with significance level $\alpha = 0.05$. For all metrics, the p-value of SASRec and TDDPG-Rec is $6.9e^{-4} < 0.01$, the p-value of TDQN and TDDPG-Rec is $2.17e^{-5} < 0.01$, and the p-value of MDDPG-Rec and TDDPG-Rec is $3.69e^{-5} < 0.01$. The results indicate that there are significant differences between the evaluated couple methods.

Table 3: Time Comparison for training and decision making.

Method	Per training step (ms)	Per decision (ms)
DDPG-kNN ($k =0.1M$)	5.26	7.86
DDPG-kNN ($k =M$)	5.51	56.42
TPGR	4.98	5.06
SASRec	205.12	14.32
TDQN-Rec	1.86	1.08
TDDPG-Rec	3.13	0.90

4.3 Time Comparison

In this section, we compare the efficiency of RL-based models on **Beauty** dataset from two aspects, the consumed time of training (updating the model) and decision making (selecting action), where the time spend is measured in millisecond. Note that since SASRec provides competitive results, we also include SASRec as a competitor.
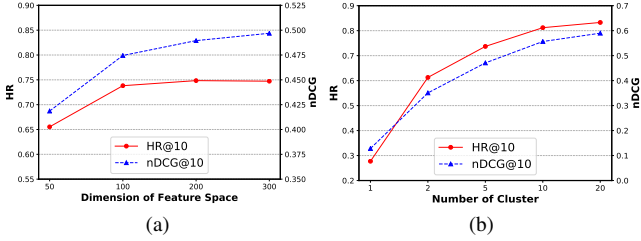
Figure 4: Performance of HR@10 and nDCG@10 *w.r.t.* (a) the dimension of feature space; (b) the number of clusters.
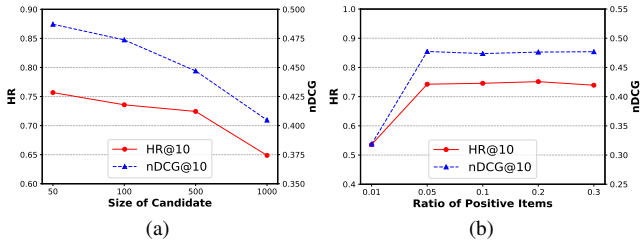

Figure 5: Performance of HR@10 and nDCG@10 *w.r.t.* (a) the size of candidate; (b) the ratio of positive items.

To make a fair comparison, both n_s and n_a are set to 1, and the experiments are conducted on the same machine with 6-core CPU (i7-6850k, 3.6GHz) and 64GB RAM.

As shown in Table 3, DDPG-kNN runs much slower than other models, because it has high time complexity. TDQN-Rec consumes the least time on training, due to its simple structure. But as shown in Table 2, it has the worst recommendation performance among all the RL-based methods. TPGR reduces the decision-making time significantly by constructing a clustering tree, but as mentioned before, it only supports Top-1 recommendation. Compared to other methods, by using policy vector and action candidate, our model TDDPG-Rec achieves significant improvement in terms of both recommendation performance and efficiency.

4.4 Parameter Sensitivity

We select several important parameters to analyze their effects on the performance of TDDPG-Rec in terms of HR@10 and nDCG@10. Note that we have conducted such experiments on all the datasets, and the results show that our approach exhibits similar performance trends on all the evaluated datasets. For simplicity, we only present the results in **Beauty** dataset. When testing one parameter, we keep the others fixed. The default settings are $D = 100$, $n_{cl} = 5$, $n_c = 100$, $\alpha = 0.5$, $n_s = 20$ and $n_a = 5$.

The Dimension of Feature Space (D)   The number of feature dimension reflects the richness of the information. As shown in Figure 4 (a), with the increase of D, as expected, TDDPG-Rec also performs better.

The Number of Clusters (n_{cl})   As shown in Figure 4 (b), the increase of n_{cl} improves the performance. This is mainly because the more clusters, the larger the difference between the current cluster and the one farthest from it, which leads to more high-quality negative items, and eventually results in better performance.

The Size of Candidate (n_c)   Figure 5 (a) shows that the performance decreases with the increase of n_c. This is mainly because the
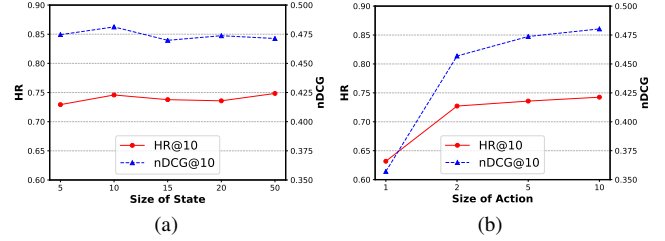

Figure 6: Performance of HR@10 and nDCG@10 *w.r.t.* (a) the size of state; (b) the size of action.

items in $\mathcal{V}_{u_i}^p$ are much less than the items in $\mathcal{V}_{cl_l^f}^p$. In other words, the increase of n_c will cause imbalance sampling, which in turn leads to worse performance.

The Ratio of Positive Items (α)   As shown in Figure 5 (b), with the increase of α, the performance first grows and then remains stable. This is because increasing α will introduce more positive items to perceive the user's interests better. But since $n_{pos} \leq |\mathcal{V}_{u_i}^p|$ (see Algorithm 1), when α is big enough, its growth may no longer affect n_{pos}.

The Size of State (n_s)   Figure 6 (a) shows that the performance stays smoothly with the increase of n_s, which means the size of state impacts TDDPG-Rec little.

The Size of Action (n_a)   Figure 6 (b) shows that with the increase of n_a, the performance also increases. This is due to that the larger n_a is, the more frequent the user state changes, which makes the positive items have more opportunities to be selected.

5 Conclusion

In this paper, we propose TDDPG-Rec, a Text-based Deep Deterministic Policy Gradient framework for Top-k interactive recommendation. By leveraging textual information and pre-trained word vectors, we embed items and users into a same feature space, which greatly alleviates the data sparsity problem. Moreover, based on the thought of collaborative filtering, we classify users into several clusters and construct an action candidate set. Combining with the policy vector dynamically learned from DDPG that expresses the user's preferences in the feature space, we select items from the candidate set to form action for recommendation, which greatly improves the efficiency of decision making. Experimental results over a carefully designed simulator on three public datasets demonstrate that compared with state-of-the-art methods, TDDPG-Rec can achieve remarkable performance improvement in a time-efficient manner.

For future work, we would like to see whether utilizing other techniques, such as the attention mechanism, can achieve better recommendation accuracy. Moreover, we intend to study if it is possible to incorporate our proposed model with transfer learning.

REFERENCES

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan, 'Automatic subspace clustering of high dimensional data for data mining applications', in *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD)*, eds., Laura M. Haas and Ashutosh Tiwary, pp. 94–105, (1998).

[2] Pierpaolo Basile, Claudio Greco, Alessandro Suglia, and Giovanni Semeraro, 'Deep learning and hierarchical reinforcement learning for modeling a conversational recommender system', *Intelligenza Artificiale*, **12**(2), 125–141, (2018).

[3] Konstantin Bauman, Bing Liu, and Alexander Tuzhilin, 'Aspect based recommendations: Recommending items with the most valuable aspects based on user reviews', in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Halifax, NS, Canada, August 13 - 17*, pp. 717–725. ACM, (2017).

[4] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu, 'Large-scale interactive recommendation with tree-structured policy gradient', in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, Hawaii, USA, January 27 - February 1*, pp. 3312–3320. AAAI Press, (2019).

[5] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song, 'Generative adversarial user model for reinforcement learning based recommendation system', in *Proceedings of the 36th International Conference on Machine Learning (ICML), 9-15 June, Long Beach, California, USA*, eds., Kamalika Chaudhuri and Ruslan Salakhutdinov, volume 97, pp. 1052–1061. PMLR, (2019).

[6] Germán Cheuque, José Guzmán, and Denis Parra, 'Recommender systems for online video game platforms: the case of STEAM', in *Proceedings of International Conference on World Wide Web (WWW), San Francisco, CA, USA, May 13-17*, eds., Sihem Amer-Yahia, Mohammad Mahdian, Ashish Goel, Geert-Jan Houben, Kristina Lerman, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, pp. 763–771. ACM, (2019).

[7] Jin Yao Chin, Kaiqi Zhao, Shafiq R. Joty, and Gao Cong, 'ANR: aspect-based neural recommender', in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM), Torino, Italy, October 22-26*, eds., Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang, pp. 147–156. ACM, (2018).

[8] Dong Deng, Liping Jing, Jian Yu, Shaolong Sun, and Haofei Zhou, 'Neural gaussian mixture model for review-based rating prediction', in *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys), Vancouver, BC, Canada, October 2-7*, eds., Sole Pera, Michael D. Ekstrand, Xavier Amatriain, and John O'Donovan, pp. 113–121. ACM, (2018).

[9] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin, 'Deep reinforcement learning in large discrete action spaces', *arXiv preprint arXiv:1512.07679*, (2015).

[10] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu, 'Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application', in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (SIGKDD)*, pp. 368–377, (2018).

[11] Kalervo Järvelin and Jaana Kekäläinen, 'Cumulated gain-based evaluation of ir techniques', *ACM Transactions on Information Systems*, **20**(4), 422–446, (October 2002).

[12] Wang-Cheng Kang and Julian J. McAuley, 'Self-attentive sequential recommendation', in *Proceedings of IEEE International Conference on Data Mining (ICDM), Singapore, November 17-20*, pp. 197–206. IEEE Computer Society, (2018).

[13] Yehuda Koren, Robert Bell, and Chris Volinsky, 'Matrix factorization techniques for recommender systems', *Computer*, (8), 30–37, (2009).

[14] Elena Kulinskaya, Stephan Morgenthaler, and Robert G. Staudte, *Significance Testing: An Overview*, 1318–1321, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[15] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[16] Lihong Li, Wei Chu, John Langford, and Robert E Schapire, 'A contextual-bandit approach to personalized news article recommendation', in *Proceedings of International Conference on World Wide Web (WWW)*, pp. 661–670, (2010).

[17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, 'Continuous control with deep reinforcement learning', in *Proceedings of the 4th International Conference on Learning Representations (ICLR Poster)*, (2016).

[18] Benjamin M. Marlin and Richard S. Zemel, 'Collaborative prediction and ranking with non-random missing data', in *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys)*, pp. 5–12, (2009).

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, et al., 'Human-level control through deep reinforcement learning', *Nature*, **518**(7540), 529, (2015).

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, 'Glove: Global vectors for word representation', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), October 25-29, Doha, Qatar*, pp. 1532–1543. ACL, (2014).

[21] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller, 'Deterministic policy gradient algorithms', in *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pp. 387–395, (2014).

[22] Haihui Tan, Ziyu Lu, and Wenjie Li, 'Neural network based reinforcement learning for real-time pushing on text stream', in *Proceedings of the 40th International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 913–916, (2017).

[23] Jiaxi Tang and Ke Wang, 'Personalized top-n sequential recommendation via convolutional sequence embedding', in *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM), Marina Del Rey, CA, USA, February 5-9*, eds., Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, pp. 565–573. ACM, (2018).

[24] Huazheng Wang, Qingyun Wu, and Hongning Wang, 'Factorization bandits for interactive recommendation', in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, San Francisco, California, USA*, pp. 2695–2702. AAAI Press, (2017).

[25] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng, 'Reinforcement learning to rank with markov decision process', in *Proceedings of the 40th International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 945–948, (2017).

[26] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen, 'Deep matrix factorization models for recommender systems', in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), Melbourne, Australia, August 19-25*, pp. 3203–3209, (2017).

[27] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin, 'Deep reinforcement learning for search, recommendation, and online advertising: A survey', *SIGWEB Newsl.*, (Spring), 4:1–4:15, (July 2019).

[28] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang, 'Deep reinforcement learning for page-wise recommendations', in *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys)*, pp. 95–103, (2018).

[29] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin, 'Recommendations with negative feedback via pairwise deep reinforcement learning', in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (SIGKDD)*, pp. 1040–1048, (2018).

[30] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang, 'Deep reinforcement learning for list-wise recommendations', *arXiv preprint arXiv:1801.00209*, (2018).

[31] Xiaoxue Zhao, Weinan Zhang, and Jun Wang, 'Interactive collaborative filtering', in *Proceedings of 22nd ACM International Conference on Information and Knowledge Management (CIKM), San Francisco, CA, USA, October 27 - November 1, 2013*, pp. 1411–1420. ACM, (2013).

[32] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li, 'Drn: A deep reinforcement learning framework for news recommendation', in *Proceedings of International Conference on World Wide Web (WWW)*, pp. 167–176, (2018).

[33] Lei Zheng, Vahid Noroozi, and Philip S. Yu, 'Joint deep modeling of users and items using reviews for recommendation', in *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM), Cambridge, United Kingdom, February 6-10*, pp. 425–434. ACM, (2017).