# Large-Scale Interactive Recommendation with Tree-Structured Policy Gradient

**Haokun Chen,**[1] **Xinyi Dai,**[1] **Han Cai,**[1] **Weinan Zhang,**[1]
**Xuejian Wang,**[1] **Ruiming Tang,**[2] **Yuzhou Zhang,**[2] **Yong Yu**[1]
[1]Shanghai Jiao Tong University
[2]Huawei Noah's Ark Lab
{chenhaokun,xydai,hcai,wnzhang,xjwang,yyu}@apex.sjtu.edu.cn
{tangruiming,zhangyuzhou3}@huawei.com

## Abstract

Reinforcement learning (RL) has recently been introduced to interactive recommender systems (IRS) because of its nature of learning from dynamic interactions and planning for long-run performance. As IRS is always with thousands of items to recommend (i.e., thousands of actions), most existing RL-based methods, however, fail to handle such a large discrete action space problem and thus become inefficient. The existing work that tries to deal with the large discrete action space problem by utilizing the deep deterministic policy gradient framework suffers from the inconsistency between the continuous action representation (the output of the actor network) and the real discrete action. To avoid such inconsistency and achieve high efficiency and recommendation effectiveness, in this paper, we propose a Tree-structured Policy Gradient Recommendation (TPGR) framework, where a balanced hierarchical clustering tree is built over the items and picking an item is formulated as seeking a path from the root to a certain leaf of the tree. Extensive experiments on carefully-designed environments based on two real-world datasets demonstrate that our model provides superior recommendation performance and significant efficiency improvement over state-of-the-art methods.

## Introduction

Interactive recommender systems (IRS) (Zhao, Zhang, and Wang 2013) play a key role in most personalized services, such as Pandora, Musical.ly and YouTube, etc. Different from the conventional recommendation settings (Mooney and Roy 2000; Koren, Bell, and Volinsky 2009), where the recommendation process is regarded as a static one, an IRS consecutively recommends items to individual users and receives their feedbacks which makes it possible to refine its recommendation policy during such interactive processes.

To handle the interactive nature, some efforts have been made by modeling the recommendation process as a multi-armed bandit (MAB) problem (Li et al. 2010; Zhao, Zhang, and Wang 2013). However, these works pre-assume that the underlying user preference remains unchanged during the recommendation process (Zhao, Zhang, and Wang 2013) and do not plan for long-run performance explicitly.

Recently, reinforcement learning (RL) (Sutton and Barto 1998), which has achieved remarkable success in various challenging scenarios that require both dynamic interaction and long-run planning such as playing games (Mnih et al. 2015; Silver et al. 2016) and regulating ad bidding (Cai et al. 2017; Jin et al. 2018), has been introduced to model the recommendation process and shows its potential to handle the interactive nature in IRS (Zheng et al. 2018; Zhao et al. 2018b; 2018a).

However, most existing RL techniques cannot handle the large discrete action space problem in IRS as the time complexity of making a decision is linear to the size of the action space. Specifically, all Deep Q-Network (DQN) based methods (Zheng et al. 2018; Zhao et al. 2018b) involve a maximization operation taken over the action space to make a decision, which becomes intractable when the size of the action space, i.e., the number of available items, is large (Dulac-Arnold et al. 2015), which is very common in IRS. Most Deep Deterministic Policy Gradient (DDPG) based methods (Zhao et al. 2018a; Hu et al. 2018) also suffer from the same problem as a specific ranking function is applied over all items to pick the one with highest score when making a decision. To reduce the time complexity, Dulac-Arnold et al. (2015) propose to select the proto-action in a continuous hidden space and then pick the valid item via a nearest-neighbor method. However, such a method suffers from the inconsistency between the learned continuous action and the actually desired discrete action, and thereby may lead to unsatisfied results (Tavakoli, Pardo, and Kormushev 2018).

In this paper, we propose a Tree-structured Policy Gradient Recommendation (TPGR) framework which achieves high efficiency and high effectiveness at the same time. In the TPGR framework, a balanced hierarchical clustering tree is built over the items and picking an item is thus formulated as seeking a path from the root to a certain leaf of the tree, which dramatically reduces the time complexity in both the training and the decision making stages. We utilize policy gradient technique (Sutton et al. 2000) to learn how to make recommendation decisions so as to maximize long-run rewards. To the best of our knowledge, this is the first work of building tree-structured stochastic policy for large-scale interactive recommendation.

Furthermore, to justify the proposed method using public available offline datasets, we construct an environment simulator to mimic online environments with principles derived from real-world data. Extensive experiments on two

real-world datasets with different settings show superior performance and significant efficiency improvement of the proposed TPGR over state-of-the-art methods.

## Related Work and Background

### Advanced Recommendation Algorithms for IRS

**MAB-based Recommendation** A group of works (Li et al. 2010; Chapelle and Li 2011; Zhao, Zhang, and Wang 2013; Zeng et al. 2016; Wang, Wu, and Wang 2016) try to model the interactive recommendation as a MAB problem. Li et al. (2010) adopt a linear model to estimate the Upper Confidence Bound (UCB) for each arm. Chapelle and Li (2011) utilize the Thompson sampling technique to address the trade-off between exploration and exploitation. Besides, some researchers try to combine MAB with matrix factorization technique (Zhao, Zhang, and Wang 2013; Kawale et al. 2015; Wang, Wu, and Wang 2017).

**RL-based Recommendation** RL-based recommendation methods (Tan, Lu, and Li 2017; Zheng et al. 2018; Zhao et al. 2018b; 2018a), which formulate the recommendation procedure as a Markov Decision Process (MDP), explicitly model the dynamic user status and plan for long-run performance. Zhao et al. (2018b) incorporate negative as well as positive feedbacks into a DQN framework (Mnih et al. 2015) and propose to maximize the difference of Q-values between the target and the competitor items. Zheng et al. (2018) combine DQN and Dueling Bandit Gradient Decent (DBGD) (Grotov and de Rijke 2016) to conduct online news recommendation. Zhao et al. (2018a) propose to utilize a DDPG framework (Lillicrap et al. 2015) with a page-display approach for page-wise recommendation.

### Large Discrete Action Space Problem in RL-based Recommendation

Most RL-based models become unacceptably inefficient for IRS with large discrete action space as the time complexity of making a decision is linear to the size of the action space.

For all DQN-based solutions (Zhao et al. 2018b; Zheng et al. 2018), a value function $Q(s, a)$, which estimates the expected discounted cumulative reward when taking the action $a$ at the state $s$, is learned and the policy's decision is:

$$\pi_Q(s) = \operatorname*{argmax}_{a \in A} Q(s, a). \tag{1}$$

As shown in Eq. (1), to make a decision, $|A|$ ($A$ denotes the item set) evaluations are required, which makes both learning and utilization intractable for tasks where the size of the action space is large, which is common for IRS.

Similar problem exists in most DDPG-based solutions (Zhao et al. 2018a; Hu et al. 2018) where some ranking parameters are learned and a specific ranking function is applied over all items to pick the one with highest ranking score. Thus, the complexity of sampling an action for these methods also grows linearly with respect to $|A|$.

Dulac-Arnold et al. (2015) attempt to address the large discrete action space problem based on the DDPG framework by mapping each discrete action to a low-dimensional continuous vector in a hidden space while maintaining an actor network to generate a continuous vector $a_v$ in the hidden space which is later mapped to a specific valid action $a$ among the $k$-nearest neighbors of $a_v$. Meanwhile, a value network $Q(s, a)$ is learned using transitions collected by executing the valid action $a$ and the actor network is updated according to $\frac{\partial Q(s,\hat{a})}{\partial \hat{a}}\big|_{\hat{a}=a_v}$ following the DDPG framework. Though such a method can reduce the time complexity of making a decision from $\mathcal{O}(|A|)$ to $\mathcal{O}(\log(|A|))$ when the value of $k$ (i.e., the number of nearest neighbors to find) is small, there is no guarantee that the actor network is learned in a correct direction as in the original DDPG. The reason is that the value network $Q(s, a)$ may behave differently on the output of the actor network $a_v$ (when training the actor network) and the actually executed action $a$ (when training the value network). Besides, the utilized approximate $k$-nearest neighbors (KNN) method may also cause trouble as the found neighbors may not be exactly the nearest ones.

In this paper, we propose a novel solution to address the large discrete action space problem. Instead of using the continuous hidden space, we build a balanced tree to represent the discrete action space where each leaf node corresponds to an action and top-down decisions are made from the root to a specific leaf node to take an action, which reduces the time complexity of making a decision from $\mathcal{O}(|A|)$ to $\mathcal{O}(d \times |A|^{1/d})$, where $d$ denotes the depth of the tree. Since such a method does not involve a mapping from the continuous space to the discrete space, it avoids the gap between the continuous vector given by the actor network and the actually executed discrete action in (Dulac-Arnold et al. 2015), which could lead to incorrect updates.

## Proposed Model

### Problem Definition

We use an MDP to model the recommendation process, where the key components are defined as follows.

- **State.** A state $s$ is defined as the historical interactions between a user and the recommender system, which can be encoded as a low-dimensional vector via a recurrent neural network (RNN) (see Figure 2).
- **Action.** An action $a$ is to pick an item for recommendation, such as a song or a video, etc.
- **Reward.** In our formulation, all users interacting with the recommender system form the environment that returns a reward $r$ after receiving an action $a$ at the state $s$, which reflects the user's feedback to the recommended item.
- **Transition.** As the state is the historical interactions, once a new item is recommended and the corresponding user's feedback is given, the state transition is determined.

An episode in the above defined MDP corresponds to one recommendation process, which is a sequence of user states, recommendation actions and user's feedbacks, e.g., $(s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_n, a_n, r_n, s_{n+1})$. In this case, the sequence starts with user state $s_1$ and then transits to $s_2$ after a recommendation action $a_1$ is carried out by the recommender system and a reward $r_1$ is given by the environment indicating the user's feedback to the recommendation action. The sequence is terminated at a specific state

$s_{n+1}$ when some pre-defined conditions are satisfied. Without loss of generality, we set the length of an episode $n$ to a fixed number (Cai et al. 2017; Zhao et al. 2018b).

## Tree-structured Policy Gradient Recommendation

**Intuition for TPGR** To handle the large discrete action space problem and achieve high recommendation effectiveness, we propose to build up a balanced hierarchical clustering tree over items (Figure 1 left) and then utilize the policy gradient technique to learn the strategy of choosing the optimal subclass at each non-leaf node of the constructed tree (Figure 1 right). Specifically, in the clustering tree, each leaf node is mapped to a certain item (Figure 1 left) and each non-leaf node is associated with a policy network (note that only three but not all policy networks are shown in the right part of Figure 1 for the ease of presentation). As such, given a state and guided by the policy networks, a top-down moving is performed from the root to a leaf node and the corresponding item is recommended to the user.

**Balanced Hierarchical Clustering over Items** Hierarchical clustering seeks to build a hierarchy of clusters, i.e., a clustering tree. One popular method is the divisive approach where the original data points are divided into several clusters, and each cluster is further divided into smaller sub-clusters. The division is repeated until each sub-cluster is associated with only one point.

In this paper, we aim to conduct *balanced* hierarchical clustering over items, where the constructed clustering tree is supposed to be balanced, i.e., for each node, the heights of its subtrees differ by at most one and the subtrees are also balanced. For the ease of presentation and implementation, it is also required that each non-leaf node has the same number of child nodes, denoted as $c$, except for parents of leaf nodes, whose numbers of child nodes are at most $c$.

We can perform balanced hierarchical clustering over items following a clustering algorithm which takes a group of vectors and an integer $c$ as input and divides the vectors into $c$ balanced clusters (i.e., the item number of each cluster differs from each other by at most one). In this paper, we consider two kinds of clustering algorithms, i.e., PCA-based and K-means-based clustering algorithms whose detailed procedures are provided in the appendices. By repeatedly applying the clustering algorithm until each sub-cluster is associated with only one item, a balanced clustering tree is constructed. As such, denoting the item set and the depth of the balanced clustering tree as $A$ and $d$ respectively, we have:

$$c^{d-1} < |A| \leq c^d. \tag{2}$$

Thus, given $A$ and $d$, we can set $c = ceil(|A|^{\frac{1}{d}})$ where $ceil(x)$ returns the smallest integer which is no less than $x$.

The balanced hierarchical clustering over items is normally performed on the (vector) representation of the items, which may largely affect the quality of the attained balanced clustering tree. In this work we consider three approaches for producing such representation:

- **Rating-based.** An item is represented as the corresponding column of the user-item rating matrix, where the value of each element $(i, j)$ is the rating of user $i$ to item $j$.

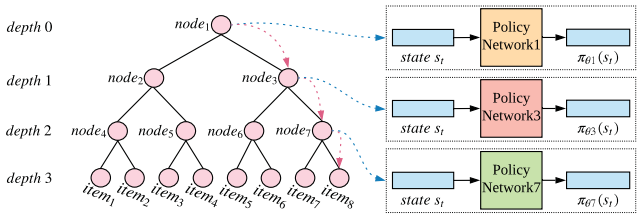


Figure 1: Architecture of TPGR.

- **VAE-based.** Low-dimensional representation of the rating vector for each item can be learned by utilizing a variational auto-encoder (VAE) (Kingma and Welling 2013).
- **MF-based.** The matrix factorization (MF) technique (Koren, Bell, and Volinsky 2009) can also be utilized to learn a representation vector for each item.

**Architecture of TPGR** The architecture of the Tree-structured Policy Gradient Recommendation (TPGR) is based on the constructed clustering tree. To ease the illustration, we assume that there is a status point to indicate which node is currently located. Thus, picking an item is to move the status point from the root to a certain leaf. Each non-leaf node of the tree is associated with a policy network which is implemented as a fully-connected neural network with a softmax activation function on the output layer. Considering node $v$ where the status point is located, the policy network associated with $v$ takes the current state as input and outputs a probability distribution over all child nodes of $v$, which indicates the probability of moving to each child node of $v$.

Using a recommendation scenario with 8 items for illustration, the constructed balanced clustering tree with the tree depth set to 3 is shown in Figure 1 (left). For a given state $s_t$, the status point is initially located at the root ($node_1$) and moves to one of its child nodes ($node_3$) according to the probability distribution given by the policy network corresponding to the root ($node_1$). And the status point keeps moving until reaching a leaf node and the corresponding item ($item_8$ in Figure 1) is recommended to the user.

We use the REINFORCE algorithm (Williams 1992) to train the model while other policy gradient algorithms can be utilized analogously. The objective is to maximize the expected discounted cumulative rewards, i.e.,

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{i=1}^{n} \gamma^{i-1} r_i\Big], \tag{3}$$

and one of its approximate gradient with respect to the parameters is:

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

where $\pi_\theta(a|s)$ is the probability of taking the action $a$ at the state $s$, and $Q^{\pi_\theta}(s,a)$ denotes the expected discounted cumulative rewards starting with $s$ and $a$, which can be estimated empirically by sampling trajectories following the policy $\pi_\theta$.

An algorithmic description of the training procedure is given in Algorithm 1 where $\mathcal{I}$ denotes the number of non-leaf nodes of the tree. When sampling an episode for TPGR

**Algorithm 1** Learning TPGR

**Require:** episode length $n$, tree depth $d$, discount factor $\gamma$, learning rate $\eta$, reward function $\mathcal{R}$, item set $A$ with representation vectors
**Ensure:** model parameters $\theta$

$c = ceil(|A|^{\frac{1}{d}})$
construct a balanced clustering tree $T$ with the number of child nodes set to $c$
$\mathcal{I} = \frac{c^d-1}{c-1}$
**for** $j = 1$ to $\mathcal{I}$ **do**
  initialize $\theta_j \leftarrow$ random values
**end for**
$\theta = (\theta_1, \theta_2, ..., \theta_{\mathcal{I}})$
**repeat**
  $\Delta\theta = 0$
  $(s_1, p_1, r_1, ..., s_n, p_n, r_n) \leftarrow$ SamplingEpisode($\theta$, $n$, $c$, $d$, $T$, $\mathcal{R}$) (see Algorithm 2)
  **for** $t = 1$ to $n$ **do**
    map $p_t$ to an item $a_t$ w.r.t. $T$ and record the trajectory nodes' indexes $(i_1, i_2, ..., i_d)$
    $\hat{Q}^{\pi_\theta}(s_t, a_t) = \sum_{i=t}^{n} \gamma^{i-t} r_i$
    $\pi_\theta(a_t|s_t) = \prod_{d'=1}^{d} \pi_{\theta_{i_{d'}}}(p_{td'}|s_t)$
    $\Delta\theta = \Delta\theta + \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{Q}^{\pi_\theta}(s_t, a_t)$
    $\theta = \theta + \eta\Delta\theta$
  **end for**
**until** converge
**return** $\theta$

(as shown in Algorithm 2), $p_t$ denotes the path from the root to a leaf at timestep $t$, which consists of $d$ choices, and each choice is represented as an integer between 1 and $c$ denoting the corresponding child node to move. Making the consecutive choices corresponding to $p_t$ from the root, we traverse the nodes along $p_t$ and finally reach a leaf node. As such, a path $p_t$ is mapped to a recommended item $a_t$, thus the probability of choosing $a_t$ given state $s_t$ is the product of the probability of making each choice (to reach $a_t$) along $p_t$.

**Time and Space Complexity Analysis** Empirically, the value of the tree depth $d$ is set to a small constant (typically set to 2 in our experiments). Thus, both the time (for making a decision) and the space complexity of each policy network is $\mathcal{O}(c)$ (see more details in the appendices).

Considering the time spent on sampling an action given a specific state in Algorithm 2, the TPGR makes $d$ choices, each of which is based on a policy network with at most $c$ output units. Therefore, the time complexity of sampling one item in the TPGR is $\mathcal{O}(d \times c) \simeq \mathcal{O}(d \times |A|^{\frac{1}{d}})$. Compared to the normal RL-based methods whose time complexity of sampling an action is $\mathcal{O}(|A|)$, our proposed TPGR can significantly reduce the time complexity.

The space complexity of each policy network is $\mathcal{O}(c)$ and the number of non-leaf nodes (i.e., the number of policy networks) of the constructed clustering tree is:

$$\mathcal{I} = 1 + c + c^2 + \cdots + c^{d-1} = \frac{c^d - 1}{c - 1}. \quad (5)$$

**Algorithm 2** Sampling Episode for TPGR

**Require:** parameters $\theta$, episode length $n$, maximum child number $c$, tree depth $d$, balanced clustering tree $T$, reward function $\mathcal{R}$
**Ensure:** an episode $E$

Initialize $s_1 \leftarrow [0]$
**for** $t = 1$ to $n$ **do**
  $node\_index = 1$
  **for** $d' = 1$ to $d$ **do**
    sample $c_{d'} \sim \pi_{\theta_{node\_index}}(s_t)$
    $node\_index = (node\_index - 1) \times c + c_{d'} + 1$
  **end for**
  $p_t = (c_1, c_2, ..., c_d)$
  map $p_t$ to an item $a_t$ w.r.t. $T$
  $r_t = \mathcal{R}(s_t, a_t)$
  **if** $t < n$ **then**
    calculate $s_{t+1}$ as described in Figure 2
  **end if**
**end for**
**return** $E = (s_1, p_1, r_1, ..., s_n, p_n, r_n)$

Therefore, the space complexity of the TPGR is $\mathcal{O}(\mathcal{I} \times c) \simeq \mathcal{O}(\frac{c^d-1}{c-1} \times c) \simeq \mathcal{O}(c^d) \simeq \mathcal{O}(|A|)$, which is the same as that of normal RL-based methods.

## State Representation

In this section, we present the state representation scheme adopted in this work, whose details are shown in Figure 2.

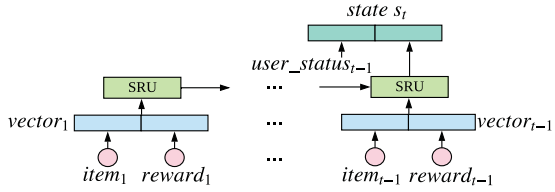


Figure 2: State representation.

In Figure 2, we assume that the recommender system is performing the $t$-th recommendation. The input is a sequence of recommended item IDs and the corresponding rewards (user's feedbacks) before timestep $t$. Each item ID is mapped to an embedding vector which can be learned together with the policy networks in an end-to-end manner, or can be pre-trained by some supervised learning models such as matrix factorization and is fixed while training. Each reward is mapped to a one-hot vector with a simple reward mapping function (see more details in the appendices).

For encoding the historical interactions, we adopt a simple recurrent unit (SRU) (Lei and Zhang 2017), an RNN model that is fast to train, to learn the hidden representation. Besides, to further integrate more feedback information, we construct a vector, denoted as $user\_status_{t-1}$ in Figure 2, containing some statistic information such as the number of positive rewards, negative rewards, consecutive positive and negative rewards before timestep $t$, which is then concate-

nated with the hidden vector generated by the SRU to gain the state representation at timestep $t$.

## Experiments and Results

### Datasets

We adopt the following two datasets in our experiments.

- **MovieLens (10M).**[1] A dataset consists of 10 million ratings from users to movies in MovieLens website.
- **Netflix.**[2] A dataset contains 100 million ratings from Netflix's competition to improve their recommender systems.

Detailed statistic information, including the number of users, items and ratings, of these datasets is given in Table 1.

Table 1: Statistic information of the datasets.

| Dataset | #users | #items | total #ratings | #ratings per user | #ratings per item |
|---|---|---|---|---|---|
| MovieLens | 69,878 | 10,677 | 10,000,054 | 143 | 936 |
| Netflix | 480,189 | 17,770 | 100,498,277 | 209 | 5,655 |

### Data Analysis

To demonstrate the existence of hidden sequential patterns in the recommendation process, we empirically analyze the aforementioned two datasets where each rating is attached with a timestamp. Each dataset comprises numerous user sessions and each session contains the ratings from one specific user to various items along timestamps.

Without loss of generality, we regard the ratings higher than 3 as positive ratings (noticed that the highest rating is 5) and the others as negative ratings. For a rating with at most $b$ consecutive positive (negative) ratings before it, we define its consecutive positive (negative) count as $b$. As such, each rating can be associated with a specific consecutive positive (negative) count and we can calculate the average rating for ratings with the same consecutive positive (negative) count.

We present the corresponding average ratings w.r.t. the consecutive positive (negative) counts in Figure 3, where we can clearly observe the sequential patterns in the user's rating behavior: a user tends to give a linearly higher rating for an item with larger consecutive positive count (green line) and vice versa (red line). The reason may be that the more satisfying (disappointing) items a user has consumed before, the more pleasure (displeasure) she gains and as a result, she tends to give a higher (lower) rating to the current item.

### Environment Simulator and Reward Function

To train and test RL-based recommendation algorithms, a straightforward way is to conduct online experiments where the recommender system can directly interact with real users, which, however, could be too expensive and commercially risky for the platform (Zhang, Paquet, and Hofmann 2016). Thus, in this paper, we focus on evaluating our proposed model on public available offline datasets by building up an environment simulator to mimic online environments.

[1]http://files.grouplens.org/datasets/movielens/ml-10m.zip

[2]https://www.kaggle.com/netflix-inc/netflix-prize-data

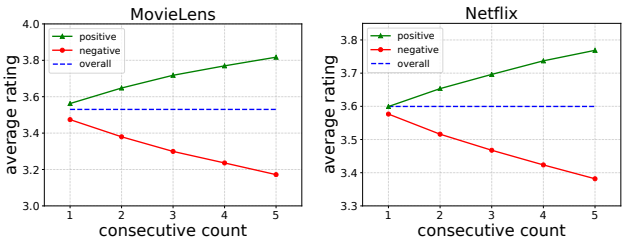


Figure 3: Average ratings for different consecutive counts.

Specifically, we normalize the ratings of a dataset into range $[-1, 1]$ and use the normalized value as the empirical reward of the corresponding recommendation. To take the sequential patterns into account, we combine a sequential reward with the empirical reward to construct the final reward function. Within each episode, the environment simulator randomly samples a user $i$ and the recommender system starts to interact with the sampled user $i$ until the end of the episode, and the reward of recommending item $j$ to user $i$, denoted as action $a$, at state $s$ is given as:

$$\mathcal{R}(s, a) = r_{ij} + \alpha \times (c_p - c_n), \tag{6}$$

where $r_{ij}$ is the corresponding normalized rating and is set to 0 if user $i$ does not rate item $j$ in the dataset, $c_p$ and $c_n$ denote the consecutive positive and negative counts respectively; $\alpha$ is a non-negative parameter to control the trade-off between the empirical reward and the sequential reward.

### Main Experiments

**Compared Methods** We compare our TPGR model with 7 methods in our experiments where Popularity and GreedySVD are conventional recommendation methods; LinearUCB and HLinearUCB are MAB-based methods; DDPG-KNN, DDPG-R and DQN-R are RL-based methods.

- **Popularity** recommends the most popular item (i.e., the item with highest average rating) from current available items to the user at each timestep.
- **GreedySVD** trains the singular value decomposition (SVD) model after each interaction and picks the item with highest rating predicted by the SVD model.
- **LinearUCB** is a contextual-bandit recommendation approach (Li et al. 2010) which adopts a linear model to estimate the upper confidence bound (UCB) for each arm.
- **HLinearUCB** is also a contextual-bandit recommendation approach (Wang, Wu, and Wang 2016) which learns extra hidden features for each arm to model the reward.
- **DDPG-KNN** denotes the method (Dulac-Arnold et al. 2015) addressing the large discrete action space problem by combining DDPG with an approximate KNN method.
- **DDPG-R** denotes the DDPG-based recommendation method (Zhao et al. 2018a) which learns a ranking vector and picks the item with highest ranking score.
- **DQN-R** denotes the DQN-based recommendation method (Zheng et al. 2018) which utilizes a DQN to estimate Q-value for each action given the current state.

Table 2: Overall interactive recommendation performance (* indicates that p-value is less than $10^{-6}$ for significance test).

| Dataset | Method | $\alpha = 0.0$ | | | | $\alpha = 0.1$ | | | | $\alpha = 0.2$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Reward | Precision@32 | Recall@32 | F1@32 | Reward | Precision@32 | Recall@32 | F1@32 | Reward | Precision@32 | Recall@32 | F1@32 |
| MovieLens | Popularity | 0.0315 | 0.0405 | 0.0264 | 0.0257 | 0.0349 | 0.0405 | 0.0264 | 0.0257 | 0.0383 | 0.0405 | 0.0264 | 0.0257 |
| | GreedySVD | 0.0561 | 0.0756 | 0.0529 | 0.0498 | 0.0655 | 0.0759 | 0.0532 | 0.0501 | 0.0751 | 0.0760 | 0.0532 | 0.0502 |
| | LinearUCB | 0.0680 | 0.0920 | 0.0627 | 0.0597 | 0.0798 | 0.0919 | 0.0627 | 0.0597 | 0.0917 | 0.0919 | 0.0627 | 0.0598 |
| | HLinearUCB | 0.0847 | 0.1160 | 0.0759 | 0.0734 | 0.1023 | 0.1162 | 0.0759 | 0.0735 | 0.1196 | 0.1165 | 0.0761 | 0.0737 |
| | DDPG-KNN($k = 1$) | 0.0116 | 0.0234 | 0.0082 | 0.0098 | 0.0143 | 0.0240 | 0.0086 | 0.0102 | 0.0159 | 0.0239 | 0.0086 | 0.0102 |
| | DDPG-KNN($k = 0.1N$) | 0.1053 | 0.1589 | 0.0823 | 0.0861 | 0.1504 | 0.1754 | 0.0918 | 0.0964 | 0.1850 | 0.1780 | 0.0922 | 0.0975 |
| | DDPG-KNN($k = N$) | 0.1764 | 0.2605 | 0.1615 | 0.1562 | 0.2379 | 0.2548 | 0.1529 | 0.1504 | 0.3029 | 0.2542 | 0.1437 | 0.1477 |
| | DDPG-R | 0.0898 | 0.1396 | 0.0647 | 0.0714 | 0.1284 | 0.1639 | 0.0798 | 0.0862 | 0.1414 | 0.1418 | 0.0656 | 0.0724 |
| | DQN-R | 0.1610 | 0.2309 | 0.1304 | 0.1326 | 0.2243 | 0.2429 | 0.1466 | 0.1450 | 0.2490 | 0.2140 | 0.1170 | 0.1204 |
| | TPGR | **0.1861*** | **0.2729*** | **0.1698*** | **0.1666*** | **0.2472*** | **0.2726*** | **0.1697*** | **0.1665*** | **0.3101** | **0.2729*** | **0.1702*** | **0.1667*** |
| Netflix | Popularity | 0.0000 | 0.0002 | 0.0001 | 0.0001 | 0.0000 | 0.0002 | 0.0001 | 0.0001 | 0.0000 | 0.0002 | 0.0001 | 0.0001 |
| | GreedySVD | 0.0255 | 0.0320 | 0.0113 | 0.0132 | 0.0289 | 0.0327 | 0.0115 | 0.0135 | 0.0310 | 0.0315 | 0.0113 | 0.0132 |
| | LinearUCB | 0.0557 | 0.0682 | 0.0212 | 0.0263 | 0.0652 | 0.0681 | 0.0212 | 0.0263 | 0.0744 | 0.0679 | 0.0211 | 0.0262 |
| | HLinearUCB | 0.0800 | 0.1005 | 0.0314 | 0.0387 | 0.0947 | 0.0999 | 0.0312 | 0.0385 | 0.1077 | 0.0995 | 0.0310 | 0.0382 |
| | DDPG-KNN($k = 1$) | 0.0195 | 0.0291 | 0.0092 | 0.0106 | 0.0252 | 0.0328 | 0.0096 | 0.0113 | 0.0272 | 0.0314 | 0.0094 | 0.0111 |
| | DDPG-KNN($k = 0.1N$) | 0.1127 | 0.1546 | 0.0452 | 0.0561 | 0.1581 | 0.1713 | 0.0546 | 0.0653 | 0.1848 | 0.1676 | 0.0517 | 0.0632 |
| | DDPG-KNN($k = N$) | 0.1355 | 0.1750 | 0.0447 | 0.0598 | 0.1770 | 0.1745 | 0.0521 | 0.0646 | 0.2519 | 0.1987 | 0.0584 | 0.0739 |
| | DDPG-R | 0.1008 | 0.1300 | 0.0343 | 0.0441 | 0.1127 | 0.1229 | 0.0327 | 0.0420 | 0.1412 | 0.1263 | 0.0351 | 0.0445 |
| | DQN-R | 0.1531 | 0.2029 | 0.0731 | 0.0824 | 0.2044 | 0.1976 | 0.0656 | 0.0757 | 0.2447 | 0.1927 | 0.0526 | 0.0677 |
| | TPGR | **0.1881*** | **0.2511*** | **0.0936*** | **0.1045*** | **0.2544*** | **0.2516*** | **0.0921*** | **0.1037*** | **0.3171*** | **0.2483*** | **0.0866*** | **0.1003*** |

**Experiment Details** For each dataset, the users are randomly divided into two parts where 80% of the users are used for training while the other 20% are used for test. In our experiments, the length of an episode is set to 32 and the trade-off factor $\alpha$ in the reward function is set to $0.0$, $0.1$ and $0.2$ respectively for both datasets. In each episode, once an item is recommended, it is removed from the set of available items, thus no repeated items occur in an episode.

For DDPG-KNN, larger $k$ (i.e., the number of nearest neighbors) leads to better performance but poorer efficiency and vice versa (Dulac-Arnold et al. 2015). For fair comparison, we consider three cases with the value of $k$ set to 1, $0.1N$ and $N$ ($N$ denotes the number of items) respectively.

For TPGR, we set the clustering tree depth $d$ to 2 and apply the PCA-based clustering algorithm with rating-based item representation when constructing the balanced tree since they give the best empirical results as shown in the following section. The implementation code[3] of the TPGR is available online.

All other hyper-parameters of all the models are carefully chosen by grid search.

**Evaluation Metrics** As the target of RL-based methods is to gain the optimal long-run rewards, we use the average reward over each recommendation for each user in test set as one evaluation metric. Furthermore, we adopt Precision@$k$, Recall@$k$ and F1@$k$ (Herlocker et al. 2004) as our evaluation metrics. Specifically, we set the value of $k$ as 32, which is the same as the episode length. For each user, all the items with a rating higher than 3.0 are regarded as the relevant items while the others are regarded as the irrelevant ones.

**Results and Analysis** In our experiments, all the models are evaluated in term of the four metrics including average reward over each recommendation, Precision@32, Recall@32, and F1@32. The summarized results are presented in Table 2 with respect to the two datasets and three different settings of trade-off factor $\alpha$ in the reward function.

From Table 2, we observe that our proposed TPGR outperforms all the compared methods in all settings with p-values less than $10^{-6}$ (indicated by a * mark in Table 2) for significance test (Ruxton 2006) in most cases, which demonstrates the performance superiority of the TPGR.

When comparing the RL-based methods with the conventional and the MAB-based methods, it is not surprising to find that the RL-based models provide superior performances in most cases, as they have the ability of long-run planning and dynamic adaptation which is lacking in other methods. Among all the RL-based methods, our proposed TPGR achieves the best performance, which can be explained by two reasons. First, the hierarchical clustering over items incorporates additional item similarity information into our model, e.g., similar items tend to be clustered into one subtree of the clustering tree. Second, different from normal RL-based methods which utilize one complicated neural network to make decisions, we propose to conduct a tree-structured decomposition and adopt a certain number of policy networks with much simpler architectures, which may ease the training process and lead to better performance.

Besides, as the value of trade-off factor $\alpha$ increases, we observe that the improvement of TPGR over HLinearUCB (i.e., the best non-RL-based method in our experiments) in terms of average reward becomes more significant, which demonstrates that the TPGR do have the capacity of capturing sequential patterns to maximize long-run rewards.

**Time Comparison** In this section, we compare the efficiency (in term of the consumed time for training and decision making stages) of RL-based models on the two datasets.

To make the time comparison fair, we remove the limitation of no repeated items in an episode to avoid involving the masking mechanism as the efficiency of the different implementations of the masking mechanism is highly different. Besides, all the models adopt the neural networks with the same architecture which consists of three fully-connected layers with the numbers of hidden units set to 32 and 16 respectively, and the experiments are conducted on the same machine with 4-core 8-thread CPU (i7-4790k, 4.0GHz) and 32GB RAM. We record the consumed time for one training step (i.e., sampling 1 thousand episodes and updating

[3]https://github.com/chenhaokun/TPGR

Table 3: Time comparison for training and decision making.

| Method | Seconds per training step | | Seconds per $10^6$ decisions | |
|---|---|---|---|---|
| | MovieLens | Netflix | MovieLens | Netflix |
| DQN-R | 13.1 | 15.3 | 19.6 | 34.6 |
| DDPG-R | 44.6 | 58.6 | 29.4 | 49.6 |
| DDPG-KNN($k = 1$) | 1.3 | 1.3 | 1.8 | 1.8 |
| DDPG-KNN($k = 0.1N$) | 24.2 | 40.3 | 200.4 | 313.0 |
| DDPG-KNN($k = N$) | 248.4 | 323.9 | 1,875.0 | 3,073.2 |
| TPGR | 3.0 | 3.1 | 3.4 | 3.9 |

the model with those episodes) and the consumed time for making 1 million decisions for each model.

As shown in Table 3, TPGR consumes much less time for both the training and the decision making stages compared to DQN-R and DDPG-R. DDPG-KNN with $k$ set to 1 gains high efficiency, which, however, is meaningless because it achieves very poor recommendation performance as shown in Table 2. In another case where $k$ is set to $N$, DDPG-KNN suffers from high time complexity which makes it even much slower than DQN-R and DDPG-R. Thus, DDPG-KNN can not achieve high effectiveness and high efficiency at the same time. Compared to the case that DDPG-KNN makes a trade-off between effectiveness and efficiency, i.e., setting $k$ as $0.1N$, our proposed TPGR achieves significant improvement in term of both effectiveness and efficiency.

## Influence of Clustering Approach

Since the architecture of the TPGR is based on the balanced hierarchical clustering tree, it is essential to choose a suitable clustering approach. In the previous section, we introduce two clustering modules, K-means-based and PCA-based modules, and three methods to represent an item, namely rating-based, MF-based and VAE-based methods. As such, there are six combinations to conduct balanced hierarchical clustering. With $\alpha$ set to 0.1, we evaluate the above six approaches in term of average reward on Netflix dataset. The results are shown in Figure 4 (left).

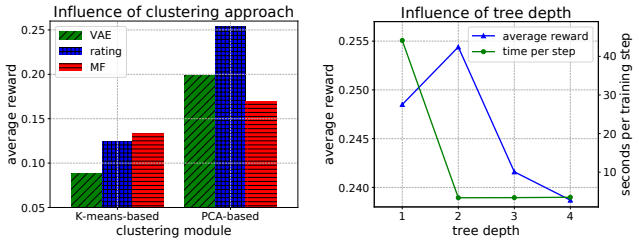


Figure 4: Influence of clustering approach and tree depth.

As shown in Figure 4 (left), applying PCA-based clustering module with rating-based item representation achieves the best performance. Two reasons may account for this result. First, the rating-based representation retains all the interaction information between the users and the items, while both the VAE-based and the MF-based representations are low-dimensional, which retain less information than rating-based representation after dimension reduction. Therefore, using rating-based representation may lead to better clustering. Second, as the number of clusters $c$ (i.e., child nodes number of non-leaf nodes) is large (134 for Netflix dataset with the tree depth set to 2), the quality of the clustering tree derived from K-means-based method would be sensitive to the choices of the initialization of center points and the distance function, etc., which may lead to worse performance than more robust methods such as PCA-based method, as observed in our experiments.

## Influence of Tree Depth

To show how the tree depth influences the performance as well as the training time of the TPGR, we vary the tree depth from 1 to 4 and record the corresponding results.

As shown in Figure 4 (right), the green curve shows the consumed time per training step with respect to different tree depths, where each training step consists of sampling 1 thousand episodes and updating the model with those episodes. It should be noticed that the model with tree depth set to 1 is actually without a tree structure but with only one policy network taking a state as input and giving the policy possibility distribution over all items. Thus, the tree-structured models (i.e., models with tree depth set to 2, 3 and 4) do significantly improve the efficiency. The blue curve in Figure 4 (right) presents the performance of the TPGR over different tree depths, from which we can see that the model with tree depth set to 2 achieves the best performance while other tree depths lead to a slight discount on performance. Therefore, setting the depth of the clustering tree to 2 is a good starting point to explore suitable tree depth when using the TPGR, which can significantly reduce the time complexity and provide great or even the best performance.

## Conclusion

In this paper, we propose a Tree-structured Policy Gradient Recommendation (TPGR) framework to conduct large-scale interactive recommendation. TPGR performs balanced hierarchical clustering over the discrete action space to reduce the time complexity of RL-based recommendation methods, which is crucial for scenarios with a large number of items. Besides, it explicitly models the long-run rewards and captures the sequential patterns so as to achieve higher rewards in the long run. Thus, TPGR has the capacity of achieving high efficiency and high effectiveness at the same time. Extensive experiments over a carefully-designed simulator based on two public datasets demonstrate that the proposed TPGR, compared to the state-of-the-art models, can lead to better performance with higher efficiency. For future work, we plan to deploy TPGR onto an online commercial recommender system. We also plan to explore more clustering tree construction schemes based on the current recommendation policy, which is also a fundamental problem for large-scale discrete action clustering in reinforcement learning.

## Acknowledgements

The work is sponsored by Huawei Innovation Research Program. The corresponding author Weinan Zhang thanks the support of National Natural Science Foundation of China (61632017, 61702327, 61772333), Shanghai Sailing Program (17YF1428200).

**Algorithm 3** K-means-based Balanced Clustering

**Require:** a group of vectors $v_1, v_2, ..., v_m$ and the number of clusters $c$
**Ensure:** clustering result
**for** $j = 1$ to $c$ **do**
  initialize the $j^{th}$ cluster $\leftarrow \emptyset$
**end for**
**if** $m \leq c$ **then**
  **for** $j = 1$ to $m$ **do**
    assign $v_j$ to the $j^{th}$ cluster
  **end for**
  **return** first $m$ clusters
**end if**
use the normal k-means algorithm to find $c$ centroids: $p_1, p_2, ..., p_c$
mark all input vectors as unassigned
$i = 1$
**while** not all vectors are marked as assigned **do**
  find the vector $v'$ among unassigned vectors which is with the shortest Euclid distance to $p_i$
  assign $v'$ to the $i^{th}$ cluster
  mark $v'$ as assigned
  $i = i \ \ mod\ c + 1$
**end while**
**return** all $c$ clusters

**Algorithm 4** PCA-based Balanced Clustering

**Require:** a group of vectors $v_1, v_2, ..., v_m$ and the number of clusters $c$
**Ensure:** clustering result
**for** $j = 1$ to $c$ **do**
  initialize the $j^{th}$ cluster $\leftarrow \emptyset$
**end for**
**if** $m \leq c$ **then**
  **for** $j = 1$ to $m$ **do**
    assign $v_j$ to the $j^{th}$ cluster
  **end for**
  **return** first $m$ clusters
**end if**
use PCA to find the principal component $u$ with the largest possible variance
sort the input vectors according to the value of projections on $u$ and gain $v_{i_1}, v_{i_2}, ..., v_{i_m}$
$thredhold = (m-1) mod\ c + 1$
$max\_length = ceil(m/c)$
**for** $j = 1$ to $thredhold$ **do**
  $start = (j-1) \times max\_length + 1$
  assign $v_{start}, v_{start+1}, ..., v_{start+max\_length-1}$ to the $j^{th}$ cluster
**end for**
**for** $j = thredhold + 1$ to $c$ **do**
  $start = thredhold \times max\_length + (j - 1 - thredhold) \times (max\_length - 1) + 1$
  assign $v_{start}, v_{start+1}, ..., v_{start+max\_length-2}$ to the $j^{th}$ cluster
**end for**
**return** all $c$ clusters

# Appendices

## Clustering Modules

We introduce two balanced clustering modules in this paper, namely, K-means-based and PCA-based modules, whose algorithmic details are shown in Algorithm 3 and Algorithm 4 respectively.

## Time and Space Complexity for Each Policy Network of TPGR

As the value of the tree depth $d$ is empirically set to a small constant (typically set to 2 in our experiments) and $c$ equals to $ceil(|A|^{\frac{1}{d}})$, we have:

$$\mathcal{O}(c + m) \simeq \mathcal{O}(|A|^{\frac{1}{d}} + m) \simeq \mathcal{O}(|A|^{\frac{1}{d}}) \simeq \mathcal{O}(c) \qquad (7)$$

and

$$\mathcal{O}(m \times c) \simeq \mathcal{O}(m \times |A|^{\frac{1}{d}}) \simeq \mathcal{O}(|A|^{\frac{1}{d}}) \simeq \mathcal{O}(c) \qquad (8)$$

where $m$ is a constant.

As described in the paper, each policy network is implemented as a fully-connected neural network. Thus, the time complexity of making a decision for each policy network is $\mathcal{O}(a + b \times c)$, where $a$ is a constant indicating the time consuming before the output layer while $b$ is also a constant indicating the number of hidden units of the hidden layer before the output layer. According to Eq. 7 and Eq. 8, we have $\mathcal{O}(a + b \times c) \simeq \mathcal{O}(c)$.

A similar analysis can be applied to derive the space complexity for each policy network. Assuming that the space occupation for each policy network except the parameters of the output layer is $a'$ and the number of hidden units of the hidden layer before the output layer is $b'$, we can derive that the space complexity for each policy network is $\mathcal{O}(a' + b' \times c) \simeq \mathcal{O}(c)$.

Thus, both the time (for making a decision) and the space complexity of each policy network is linear to the size of its output units, i.e., $\mathcal{O}(c)$.

## Reward Mapping Function

Assuming that the range of reward values is $(a, b]$ and the desired dimension of the one-hot vector is $l$, we define the reward mapping function as:

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

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

# References

Cai, H.; Ren, K.; Zhang, W.; Malialis, K.; Wang, J.; Yu, Y.; and Guo, D. 2017. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 661–670. ACM.

Chapelle, O., and Li, L. 2011. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, 2249–2257.

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

Grotov, A., and de Rijke, M. 2016. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 1215–1218. ACM.

Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; John; and Riedl, T. 2004. Evaluating collaborative filtering recommender systems. j acm trans inform syst. *Acm Transactions on Information Systems* 22(1):5–53.

Hu, Y.; Da, Q.; Zeng, A.; Yu, Y.; and Xu, Y. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application.

Jin, J.; Song, C.; Li, H.; Gai, K.; Wang, J.; and Zhang, W. 2018. Real-time bidding with multi-agent reinforcement learning in display advertising. *arXiv preprint arXiv:1802.09756*.

Kawale, J.; Bui, H.; Kveton, B.; Long, T. T.; and Chawla, S. 2015. Efficient thompson sampling for online matrix-factorization recommendation. 28.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42(8).

Lei, T., and Zhang, Y. 2017. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*.

Li, L.; Chu, W.; Langford, J.; and Schapire, R. E. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, 661–670. ACM.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

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

Mooney, R. J., and Roy, L. 2000. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, 195–204. ACM.

Ruxton, G. D. 2006. The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test. *Behavioral Ecology* 17(4):688–690.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Tan, H.; Lu, Z.; and Li, W. 2017. Neural network based reinforcement learning for real-time pushing on text stream. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 913–916. ACM.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action branching architectures for deep reinforcement learning. *AAAI*.

Wang, H.; Wu, Q.; and Wang, H. 2016. Learning hidden features for contextual bandits. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 1633–1642. ACM.

Wang, H.; Wu, Q.; and Wang, H. 2017. Factorization bandits for interactive recommendation. In *AAAI*, 2695–2702.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer. 5–32.

Zeng, C.; Wang, Q.; Mokhtari, S.; and Li, T. 2016. Online context-aware recommendation with time varying multi-armed bandit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2025–2034. ACM.

Zhang, W.; Paquet, U.; and Hofmann, K. 2016. Collective noise contrastive estimation for policy transfer learning. In *AAAI*, 1408–1414.

Zhao, X.; Xia, L.; Zhang, L.; Ding, Z.; Yin, D.; and Tang, J. 2018a. Deep reinforcement learning for page-wise recommendations. *arXiv preprint arXiv:1805.02343*.

Zhao, X.; Zhang, L.; Ding, Z.; Xia, L.; Tang, J.; and Yin, D. 2018b. Recommendations with negative feedback via pairwise deep reinforcement learning. *arXiv preprint arXiv:1802.06501*.

Zhao, X.; Zhang, W.; and Wang, J. 2013. Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, 1411–1420. ACM.

Zheng, G.; Zhang, F.; Zheng, Z.; Xiang, Y.; Yuan, N. J.; Xie, X.; and Li, Z. 2018. Drn: A deep reinforcement learning framework for news recommendation. *WWW*.