

# Toward Simulating Environments in Reinforcement Learning Based Recommendations

Xiangyu Zhao  
Michigan State University  
zhaoxi35@msu.edu

Long Xia  
JD.com  
xialong@jd.com

Zhuoye Ding  
JD.com  
dingzhuoye@jd.com

Dawei Yin  
JD.com  
yindawei@acm.org

Jiliang Tang  
Michigan State University  
tangjili@msu.edu

## ABSTRACT

With the recent advances in Reinforcement Learning (RL), there have been tremendous interests in employing RL for recommender systems. RL-based recommender systems have two key advantages: (i) they can continuously update their recommendation strategies according to users' real-time feedback, and (ii) the optimal strategy maximizes the long-term reward from users, such as the total revenue of a recommendation session. However, directly training and evaluating a new RL-based recommendation algorithm needs to collect users' real-time feedback in the real system, which is time and efforts consuming and could negatively impact on users' experiences. Thus, it calls for a user simulator that can mimic real users' behaviors where we can pre-train and evaluate new recommendation algorithms. Simulating users' behaviors in a dynamic system faces immense challenges – (i) the underlining item distribution is complex, and (ii) historical logs for each user are limited. In this paper, we develop a user simulator base on Generative Adversarial Network (GAN). To be specific, we design the generator to capture the underlining distribution of users' historical logs and generate realistic logs that can be considered as augmentations of real logs; while the discriminator is developed to not only distinguish real and fake logs but also predict users' behaviors. The experimental results based on real-world e-commerce data demonstrate the effectiveness of the proposed simulator. Further experiments have been conducted to understand the importance of each component in the simulator.

## KEYWORDS

User Simulator, Generative Adversarial Network, Reinforcement Learning, Recommender System

### ACM Reference Format:

Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Toward Simulating Environments in Reinforcement Learning Based Recommendations. In *Woodstock '18: ACM Symposium on Neural Gaze Detection*,

June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages.  
<https://doi.org/10.1145/1122445.1122456>

## 1 INTRODUCTION

With the explosive growth of the world-wide web, huge amounts of data have been generated, which results in the increasingly severe information overload problem that users are overwhelmed by massive information [6]. Recommender system can mitigate the information overload problem through suggesting personalized items (products, services, or information) that best match users' preferences [2, 14, 20, 26, 38]. The majority of existing recommender systems, e.g., content-based, learning-to-rank and collaborative filtering, often face several common challenges. First, most traditional recommender systems consider the recommendation task as a static procedure and generate recommendations via a fixed greedy strategy. However, these approaches may fail to capture the evolution of users' preferences over time. Second, most current recommender systems have been developed to maximize the short-term reward (e.g. immediate revenue) of recommendations, i.e., immediately purchasing the recommended items, but completely neglect whether these recommendations will result in more profitable rewards in the long run [29].

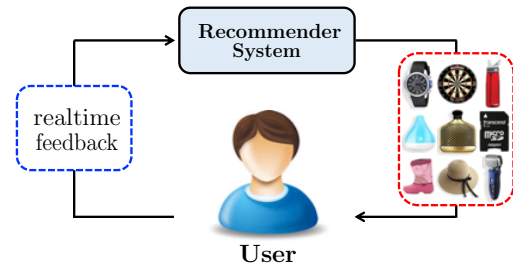


Figure 1: An example of system-user interactions.

Recently, with the immense development of Reinforcement Learning techniques, a wide range of complex tasks such as the game of Go [30, 31], video games [24, 25] and robotics [16] have been unprecedentedly advanced. In the reinforcement learning framework, an intelligent agent learns to solve a complex task by acquiring experiences from try-and-error interactions with a dynamic environment. The goal is to automatically learn an optimal policy (solution) for the complex task without any specific instructions [15]. Applying reinforcement learning in recommendation tasks can naturally

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

Woodstock '18, June 03–05, 2018, Woodstock, NY

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9999-9/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

solve the aforementioned challenges, where recommendation procedures are considered as sequential interactions between users and a recommender system [35–37, 39, 40] as shown in Figure 1. In each iteration, the recommender system suggests a set of items to the user; then the user browses the recommended items and provides her/his real-time feedback; and next the system will update its recommendation strategy according to user's feedback. First, considering the recommendation task as sequential interactions between an agent (recommender system) and environment (users), the agent can continuously update its strategies according to users' real-time feedback (reward function) during the interactions, until the system generates items that best fit users' preferences (the optimal policy). Second, the RL mechanism is developed to maximize the cumulative (long-term) reward from users, e.g., the total revenue of a recommendation session. Therefore, the agent is able to identify items with smaller immediate rewards but benefit the cumulative rewards in the long run. Given the advantages of reinforcement learning, very recently, there have been tremendous interests in developing RL-based recommender systems [10, 33, 36, 39–41].

RL-based recommendation algorithms are desired to be trained and evaluated based on users' real-time feedback (reward function). The most practical way is online A/B test, where a new recommendation algorithm is trained based on the feedback from real users and the performance is compared against that of the previous algorithm via randomized experiments. However, online A/B tests are expensive and inefficient: (1) online A/B tests usually take several weeks to collect sufficient data for sake of statistical sufficiency, and (2) numerous engineering efforts are required to deploy the new algorithm in the real system [12, 19, 34]. Furthermore, online A/B tests often lead to bad user experience in the initial stage when the new recommendation algorithms have not been well trained [18]. These reasons prevent us from quickly training and testing new RL-based recommendation algorithms. One successful solution to handle these challenges in the RL community is to first build a simulator to approximate the environment (e.g. OpenAI Gym for video games), and then use it to train and evaluate the RL algorithms [11]. Thus, following the best practice, we aim to build a user simulator based on users' historical logs in this work, which can be utilized to pre-train and evaluate new recommendation algorithms before launching them online.

However, simulating users' behaviors (preferences) in a dynamic recommendation environment is very challenging. There are millions of items in practical recommender systems. Thus the underlining distribution of recommended item sequences are widely spanned and extremely complex in historical logs. In order to learn a robust simulator, it typically requires large-scale historical logs as training data from each user. Though massive historical logs are often available, data available to each user is rather limited. An attempt to tackle the two aforementioned challenges, we propose a simulator (RecSimu) for reinforcement learning based recommendations based on Generative Adversarial Network (GAN) [13]. We summarize our major contributions as follows:

- We introduce a principled approach to capture the underlining distribution of recommended item sequences in historical logs, and generate realistic item sequences;

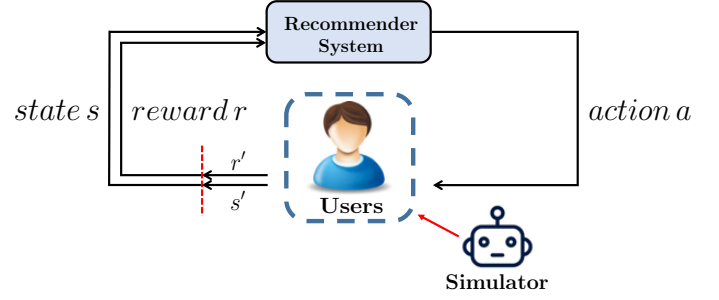


Figure 2: A common setting for RL-based recommendations.

- We propose a user behavior simulator RecSimu, which can be utilized to simulate environments to pre-train and evaluate reinforcement learning based recommender systems; and
- We conduct experiments based on real-world data to demonstrate the effectiveness of the proposed simulator and validate the effectiveness of its components.

## 2 PROBLEM STATEMENT

Following one common setting as shown in Figure 2, we first formally define reinforcement learning based recommendations [36, 39] and then present the problem we aim to solve based on this setting. In this setting, we treat the recommendation task as sequential interactions between a recommender system (agent) and users (environment  $\mathcal{E}$ ), and use Markov Decision Process (MDP) to model it. It consists of a sequence of states, actions and rewards. Typically, MDP involves four elements  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ , and below we introduce how to set them. Note that there are other settings [10, 40, 41] and we leave further investigations on them as one future work.

- **State space  $\mathcal{S}$ :** We define the state  $s = \{i_1, \dots, i_N\} \in \mathcal{S}$  as a sequence of  $N$  items that a user browsed and user's corresponding feedback for each item. The items in  $s$  are chronologically sorted.
- **Action space  $\mathcal{A}$ :** An action  $a \in \mathcal{A}$  from the recommender system perspective is defined as recommending a set of items to a user. Without loss of generality, we suppose that each time the recommender system suggests one item to the user, but it is straightforward to extend this setting to recommending more items.
- **Reward  $\mathcal{R}$ :** When the system takes an action  $a$  based on the state  $s$ , the user will browse the recommended item and provide her feedback of the item. In this paper, we assume a user could skip, click and purchase the recommended items. Then the recommender system will receive a reward  $r(s, a)$  solely according to the type of feedback.
- **State transition probability  $\mathcal{P}$ :** State transition probability  $p(s'|s, a)$  is defined as the probability that the state transits from  $s$  to  $s'$  when action  $a$  is executed. We assume that the MDP satisfies  $p(s'|s_t, a_t, \dots, s_1, a_1) = p(s'|s_t, a_t)$ , and the state transition is deterministic: we always remove the first item  $i_1$  from  $s$  and add the action  $a$  at the bottom of  $s$ , i.e.,  $s' = \{i_2, \dots, i_N, a\}$ .

With the aforementioned definitions and notations, in this paper, we aim to build a simulator to imitate users' feedback (behavior) on a recommended item according to user's preference learned from

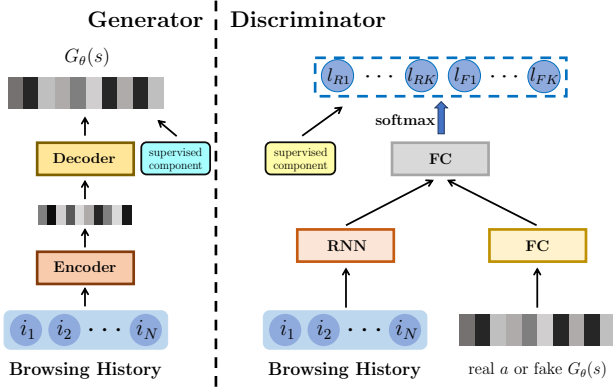


Figure 3: An overview of the proposed simulator.

the user’s browsing history as demonstrated in Figure 2. In other words, the simulator aims to mimic the reward function  $r(s, a)$ . More formally, the goal of a simulator can be formally defined as follows: *Given a state-action pair  $(s, a)$ , the goal is to find a reward function  $r(s, a)$ , which can imitate user’s behaviors as much as possible.*

### 3 THE PROPOSED SIMULATOR

In this section, we will propose a simulator framework that imitates users’ feedback (behavior) on a recommended item according to the user’s current preference learned from her browsing history. As discussed in Section 1, building a user simulator is challenging, since (1) the underlining distribution of recommended item sequences in users’ historical logs is complex, and (2) historical data for each user is usually limited.

Recent efforts have demonstrated that Generative Adversarial Network (GAN) and its variants are able to generate fake but realistic images [13], which implies their potential in modeling complex distributions. Furthermore, the generated images can be considered as augmentations of real-world images to enlarge the data space. Driven by these advantages, we propose to build the simulator based on GAN to capture the complex distribution of users’ browsing logs and generate realistic logs to enrich the training dataset. Another challenge with GAN-based simulator is that the discriminator should not only be able to distinguish real logs and generated logs, but also can predict user’s feedback of a recommended item. To address these challenges, we propose a recommendation simulator as shown in Figure 3, where the generator with a supervised component is designed to learn the data distribution and generate indistinguishable logs, and the discriminator with a supervised component can simultaneously distinguish real/generated logs and predict user’s feedback of a recommended item. In the following, we will first introduce the architectures of generator and discriminator separately, and then discuss the objective functions with the optimization algorithm.

#### 3.1 The Generator Architecture

The goal of the generator is to learn the data distribution and then generate indistinguishable logs (action) based on users’ browsing history (state), i.e., to imitate the recommendation policy of the

recommender system that generates the historical logs. Figure 4 illustrates the generator with the Encoder-Decoder architecture. The Encoder component aims to learn user’s preference according to the items browsed by the user and the user’s feedback. The input is the state  $s = \{i_1, \dots, i_N\}$  that is observed in the historical logs, i.e., the sequence of  $N$  items that a user browsed and user’s corresponding feedback for each item. The output is a low-dimensional representation of user’s current preference, referred as to  $p^E$ . Each item  $i_n \in s$  involves two types of information:

$$i_n = (e_n, f_n), \quad (1)$$

where  $e_n$  is a low-dimensional and dense item-embedding of the recommended item<sup>1</sup>, and  $f_n$  is a one-hot vector representation to denote user’s feedback on the recommended item. The intuition of selecting these two types of information is that, we not only want to learn the information of each item in the sequence, but also want to capture user’s interests (feedback) on each item. We use an embedding layer to transform  $f_n$  into a low-dimensional and dense vector:  $F_n = \tanh(W_F f_n + b_F) \in \mathbb{R}^{|F|}$ . Note that we use “tanh” activate function since  $e_n \in (-1, +1)$ . Then, we concatenate  $e_n$  and  $F_n$ , and get a low-dimensional and dense vector  $I_n \in \mathbb{R}^{|I|}$  ( $|I| = |E| + |F|$ ) as:

$$\begin{aligned} I_n &= \text{concat}(e_n, F_n) \\ &= \text{concat}(e_n, \tanh(W_F f_n + b_F)). \end{aligned} \quad (2)$$

Note that all embedding layers share the same parameters  $W_F$  and  $b_F$ , which can reduce the amount of parameters and have better generalization. We introduce a Recurrent Neural Network (RNN) with Gated Recurrent Units (GRU) to capture the sequential patterns of items in the logs. We choose GRU rather than Long Short-Term Memory (LSTM) because of GRU’s fewer parameters and simpler architecture. Unlike LSTM utilizing an input gate and a forget gate to yield a new state, GRU uses an update gate  $z_n$ :

$$z_n = \sigma(W_z I_n + U_z h_{n-1}). \quad (3)$$

GRU employs a reset gate  $r_n$  to control the former state  $h_{n-1}$ :

$$r_n = \sigma(W_r I_n + U_r h_{n-1}). \quad (4)$$

The candidate activation function  $\hat{h}_n$  is computed as:

$$\hat{h}_n = \tanh[W I_n + U(r_n \cdot h_{n-1})]. \quad (5)$$

Finally, the activation of GRU is a linear interpolation between the the candidate activation  $\hat{h}_n$  and the previous activation  $h_{n-1}$ :

$$h_n = z_n \hat{h}_n + (1 - z_n) h_{n-1}. \quad (6)$$

We consider the final hidden state  $h_n$  as the output of Encoder component, i.e., the lower dimensional representation of user’s current preference:

$$p^E = h_n. \quad (7)$$

The goal of the Decoder component is to predict the item that will be recommended according to the user’s current preference. Therefore, the input is user’s preference representation  $p^E$ , while the output is the item-embedding of the item that is predicted to

<sup>1</sup>The item-embeddings are pre-trained by an e-commerce company via word embedding [17] based on users’ historical browsing logs, where each item is considered as a word and the item sequence of a recommendation session as a sentence. The effectiveness of these item representations is demonstrated in their business like searching, recommendation and advertisement.

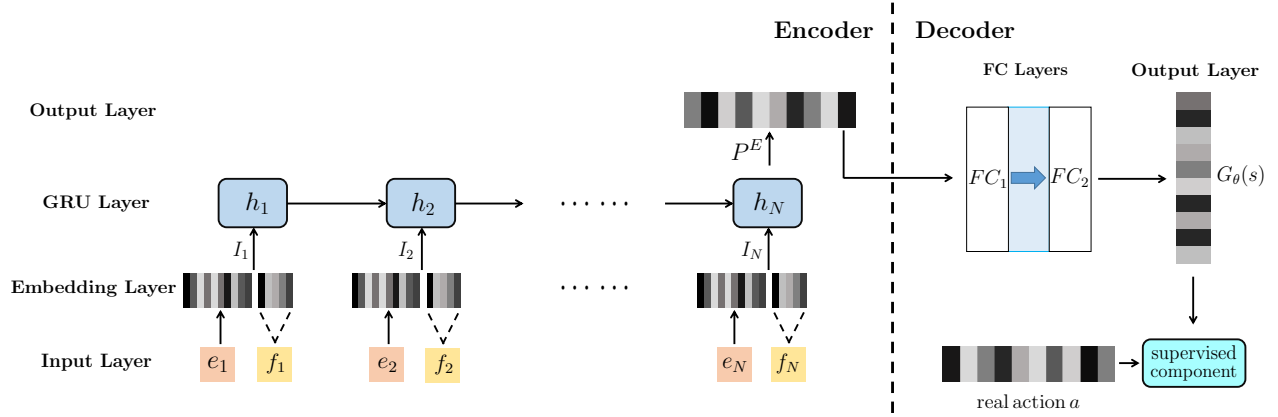


Figure 4: The proposed generator with Encoder-Decoder architecture.

appear at next position in the log, referred as to  $G_\theta(s)$ . For simplification, we leverage several fully-connected layers as the Decoder to directly transform  $p^E$  to  $G_\theta(s)$ . Note that it is straightforward to leverage other methods to generate the next item, such as using a *softmax* layer to compute relevance scores of all items, and selecting the item with the highest score as the next item. So far, we have delineated the architecture of the Generator, which aims to imitate the recommendation policy of the existing recommender system, and generate realistic logs to augment the historical data. In addition, we add a supervised component to encourage the generator to yield items that are close to the ground truth items, which will be discussed in Section 3.3. Next, we will discuss the architecture of discriminator.

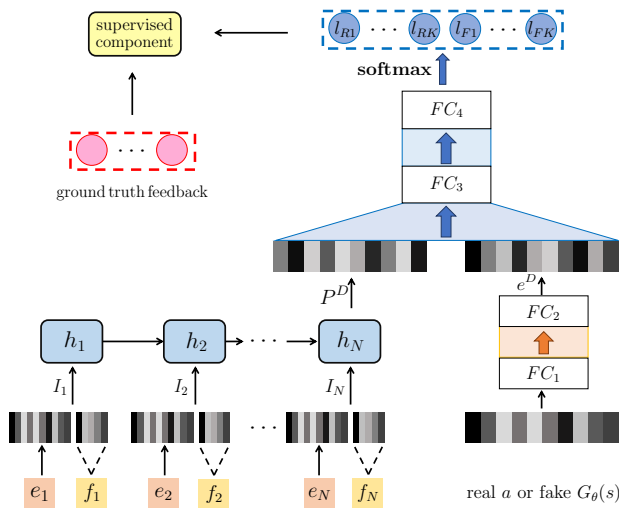


Figure 5: The discriminator architecture.

### 3.2 The Discriminator Architecture

The discriminator aims to not only distinguish real historical logs and generated logs, but also predict the class of user's feedback

of a recommended item according to her browsing history. Thus we consider the problem as a classification problem with  $2 \times K$  classes, i.e.,  $K$  classes of *real* feedback for the recommended items observed from historical logs, and  $K$  classes of *fake* feedback for the recommended items yielded by the generator.

Figure 5 illustrates the architecture of the discriminator. Similar with the generator, we introduce a RNN with GRU to capture user's dynamic preference. Note that the architecture is the same with the RNN in generator, but they have different parameters. The input of the RNN is the state  $s = \{i_1, \dots, i_N\}$  observed in the historical logs, where  $i_n = (e_n, f_n)$ , and the output is the dense representation of user's current preference, referred as to  $p^D$ . Meanwhile, we feed the item-embedding of the recommended item (real  $a$  or fake  $G_\theta(s)$ ) into fully-connected layers, which encode the recommended items to low-dimensional representations, referred as to  $e^D$ . Then we concatenate  $p^D$  and  $e^D$ , and feed the concatenation ( $p^D, e^D$ ) into fully-connected layers, whose goal is to (1) judge whether the recommended items are real or fake, and (2) predict users' feedback on these items. Therefore, the final fully-connected layer outputs a  $2 \times K$  dimensional vector of logits, which represent  $K$  classes of *real* feedback and  $K$  classes of *fake* feedback respectively:

$$\text{output} = [l_{R1}, \dots, l_{RK}, l_{F1}, \dots, l_{FK}], \quad (8)$$

where we include  $K$  classes of *fake* feedback in output layer rather than only one *fake* class, since fine-grained distinction on fake samples can increase the power of discriminator (more details in following subsections). These logits can be transformed to class probabilities through a softmax layer, and the probability corresponding to the  $j^{\text{th}}$  class is:

$$p_{\text{model}}(r = l_j | s, a) = \frac{\exp(l_j)}{\sum_{k=1}^{2 \times K} \exp(l_k)}, \quad (9)$$

where  $r$  is the result of classification. The objective function is based on these class probabilities. In addition, a supervised component is introduced to enhance the user's feedback prediction and more details about this component will be discussed in Section 3.3.

### 3.3 The Objective Function

In this subsection, we will introduce the objective functions of the proposed simulator. The discriminator has two goals: (1) distinguishing real-world historical logs and generated logs, and (2) predicting the class of user's feedback of a recommended item according to the browsing history. The first goal corresponds to an unsupervised problem just like standard GAN that distinguishes real and fake images, while the second goal is a supervised problem that minimizes the class difference between users' ground truth feedback and the predicted feedback. Therefore, the loss function  $L_D$  of discriminator consists of two components.

For the unsupervised component that distinguishes real-world historical logs and generated logs, we need calculate the probability that a state-action pair is *real* or *fake*. From Eq (9), we know the probability that a state-action pair observed from historical logs is classified as *real*, referred as to  $D_\phi(s, a)$ , is the summation of the probabilities of  $K$  *real* feedback:

$$D_\phi(s, a) = \sum_{k=1}^K p_{model}(r = l_k | s, a) \quad (10)$$

while the probability of a *fake* state-action pair where  $G_\theta(s)$  action is produced by the generator, say  $D_\phi(s, G_\theta(s))$ , is the summation of the probabilities of  $K$  *real* feedback:

$$D_\phi(s, G_\theta(s)) = \sum_{k=K+1}^{2 \times K} p_{model}(r = l_k | s, G_\theta(s)) \quad (11)$$

Then, the unsupervised component of the loss function  $L_D$  is defined as follows:

$$L_D^{unsup} = -\{\mathbb{E}_{s,a \sim p_{data}}[\log D_\phi(s, a)] + \mathbb{E}_{s \sim p_{data}}[\log D_\phi(s, G_\theta(s))]\}, \quad (12)$$

where both  $s$  and  $a$  are sampled from historical logs distribution  $p_{data}$  in the first term; in the second term, only  $s$  is sampled from historical logs distribution  $p_{data}$ , while the action  $G_\theta(s)$  is yielded by generator policy  $G_\theta$ .

The supervised component targets to predict the class of user's feedback, which is formulated as a supervised problem to minimize the class difference (i.e. the cross-entropy loss) between users' ground truth feedback and the predicted feedback. Thus it also has two terms – the first term is the cross-entropy loss of ground truth class and predicted class for a real state-action pair sampled from real historical data distribution  $p_{data}$ ; while the second term is the cross-entropy loss of ground truth class and predicted class for a fake state-action pair, where the action is yielded by the generator. Thus the supervised component of the loss function  $L_D$  is defined as follows:

$$L_D^{sup} = -\{\mathbb{E}_{s,a,r \sim p_{data}}[\log p_{model}(r = l_k | s, a, k \leq K)] + \lambda \cdot \mathbb{E}_{s,r \sim p_{data}}[\log p_{model}(r = l_k | s, G_\theta(s), K < k \leq 2K)], \quad (13)$$

where  $\lambda$  controls the contribution of the second term. The first term is a standard cross entropy loss of a supervised problem. The intuition we introduce the second term of Eq (13) is – in order to tackle the data limitation challenge mentioned in Section 1, we consider fake state-action pairs as augmentations of real state-action pairs, then fine-grained distinction on fake state-action pairs

will increase the power of discriminator, which also in turn forces the generator to output more indistinguishable actions. The overall loss function of the discriminator  $L_D$  is defined as follows:

$$L_D = L_D^{unsup} + \alpha \cdot L_D^{sup}, \quad (14)$$

where parameter  $\alpha$  is introduced to control the contribution of the supervised component.

The target of the generator is to output realistic recommended items  $G_\theta(s)$  that can fool the discriminator, which tackles the complex data distribution problem as mentioned in Section 1. To achieve this goal, we design two components for the loss function  $L_G$  of the generator. The first component aims to maximize  $L_D^{unsup}$  in Eq (12) with respect to  $G_\theta$ . In other words, the first component minimizes that probabilities that fake state-action pairs are classified as *fake*, thus we have:

$$L_G^{unsup} = \mathbb{E}_{s \sim p_{data}}[\log D_\phi(s, G_\theta(s))], \quad (15)$$

where  $s$  is sampled from real historical logs distribution  $p_{data}$  and the action  $G_\theta(s)$  is yielded by generator policy  $G_\theta$ . Inspired by a supervised version of GAN [22], we introduce a supervised loss  $L_G^{sup}$  as the second component of  $L_G$ , which is the  $\ell_2$  distance between the ground truth item  $a$  and the generated item  $G_\theta(s)$ :

$$L_G^{sup} = \mathbb{E}_{s,a \sim p_{data}} \|a - G_\theta(s)\|_2^2. \quad (16)$$

where  $s$  and  $a$  are sampled from historical logs distribution  $p_{data}$ . This supervised component encourages the generator to yield items that are close to the ground truth items. The overall loss function of the discriminator  $L_D$  is defined as follows:

$$L_G = L_G^{unsup} + \beta \cdot L_G^{sup}, \quad (17)$$

where  $\beta$  controls the contribution of the supervised component.

---

#### Algorithm 1 An Training Algorithm for the Proposed Simulator.

---

- 1: Initialize the generator  $G_\theta$  and discriminator  $D_\phi$  with random weights  $\theta$  and  $\phi$
  - 2: Sample a pre-training dataset of  $s, a \sim p_{data}$
  - 3: Pre-train  $G_\theta$  by minimizing  $L_G^{sup}$  in Eq (16)
  - 4: Generate fake-actions  $G_\theta(s) \sim G_\theta$  for training  $D_\phi$
  - 5: Pre-train  $D_\phi$  by minimizing  $L_D^{sup}$  in Eq (13)
  - 6: **repeat**
  - 7:   **for** d-steps **do**
  - 8:     Sample minibatch of  $s, a \sim p_{data}$
  - 9:     Use current  $G_\theta$  to generate minibatch of  $G_\theta(s) \sim G_\theta$
  - 10:    Update the  $D_\phi$  by minimizing  $L_D$  in Eq (14)
  - 11:   **end for**
  - 12:   **for** g-steps **do**
  - 13:     Sample minibatch of  $s, a \sim p_{data}$
  - 14:     Update the  $G_\theta$  by minimizing  $L_G$  in Eq (17)
  - 15:   **end for**
  - 16: **until** simulator converges
- 

We present our simulator training algorithm in details shown in Algorithm 1. At the beginning of the training stage, we use standard supervised methods to pre-train the generator (line 3) and discriminator (line 5). After the pre-training stage, the discriminator (lines 7-11) and generator (lines 12-15) are trained alternatively. For



training the discriminator, state  $s$  and real action  $a$  are sampled from real historical logs, while fake actions  $G_\theta(s)$  are generated through the generator. To keep balance in each d-step, we generate fake actions  $G_\theta(s)$  with the same number of real actions  $a$ .

## 4 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of the proposed simulator with a real-world dataset from an e-commerce site. We mainly focus on two questions: (1) how the proposed simulator performs compared to the state-of-the-art baselines; and (2) how the components in the generator and discriminator contribute to the performance. We first introduce experimental settings. Then we seek answers to the above two questions. Finally, we study the impact of important parameters on the performance of the proposed framework.

### 4.1 Experimental Settings

We evaluate our method on a dataset of July 2018 from a real e-commerce company. We randomly collect 272,250 recommendation sessions, and each session is a sequence of item-feedback pairs. After filtering out items that appear less than 5 times, there remain 1,355,255 items. For each session, we use first  $N$  items and corresponding feedback as the initial state, the  $N + 1^{th}$  item as the first action, then we could collect a sequence of (state, action, reward) tuples following the MDP defined in Section 2. We collect the last (state, action, reward) tuples from all recommendation sessions as the test set, while using the other tuples as the training set.

In this paper, we leverage  $N = 20$  items that a user browsed and user's corresponding feedback for each item as state  $s$ . The dimension of the item-embedding  $e_n$  is  $|E| = 20$ , and the dimension of action representation  $F_n$  is  $|F| = 10$  ( $f_n$  is a 2-dimensional one-hot vector:  $f_n = [1, 0]$  when feedback is negative, while  $f_n = [0, 1]$  when feedback is positive). The output of discriminator is a 4 ( $K = 2$ ) dimensional vector of logits, and each logit represents *real-positive*, *real-negative*, *fake-positive* and *fake-negative* respectively:

$$\text{output} = [l_{rp}, l_{rn}, l_{fp}, l_{fn}], \quad (18)$$

where *real* denotes that the recommended item is observed from historical logs; *fake* denotes that the recommended item is yielded by the generator; *positive* denotes that a user clicks/purchases the recommended item; and *negative* denotes that a user skips the recommended item. Note that though we only simulate two types of behaviors of users (i.e., positive and negative), it is straightforward to extend the simulators with more types of behaviors. AdamOptimizer is applied in optimization, and the learning rate for both Generator and Discriminator is 0.001, and batch-size is 500. The hidden size of RNN is 128. For the parameters of the proposed framework such as  $\alpha$ ,  $\beta$  and  $\lambda$ , we select them via cross-validation. Correspondingly, we also do parameter-tuning for baselines for a fair comparison. We will discuss more details about parameter selection for the proposed simulator in the following subsections.

In the test stage, given a state-action pair, the simulator will predict the classes of user's feedback for the action (recommended item), and then compare the prediction with ground truth feedback observed from the historical log. For this classification task, we select the commonly used *F1-score* as the metric, which is a measure

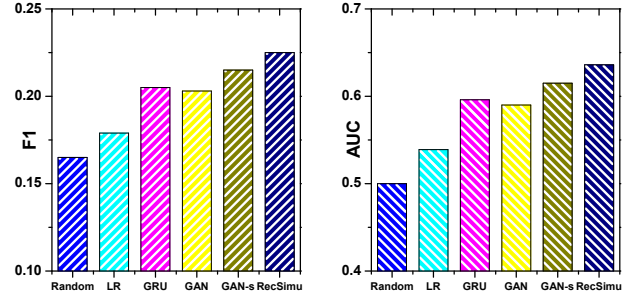


Figure 6: The results of overall performance comparison.

that combines precision and recall, namely the harmonic mean of precision and recall. Moreover, we leverage  $p_{model}(r = l_{rp}|s, a)$  (i.e. the probability that user will provide positive feedback to a real recommended item) as the score, and use *AUC* (Area under the ROC Curve) as the metric to evaluate the performance.

### 4.2 Comparison of the Overall Performance

To answer the first question, we compare the proposed simulator with the following state-of-the-art baseline methods:

- **Random:** This baseline randomly assigns each recommended item a *score*  $\in [0, 1]$ , and uses 0.5 as the threshold value to classify items as positive or negative; this *score* is also used to calculate *AUC*.
- **LR:** Logistic Regression [23] uses a logistic function to model a binary dependent variable through minimizing the loss  $\mathbb{E}_{\frac{1}{2}}(h_\theta(x) - y)^2$ , where  $h_\theta(x) = \frac{1}{1 + e^{-w^T x}}$ ; we concatenate all  $i_n = (e_n, f_n)$  as the feature vector for the  $i$ -th item, and set ground truth  $y = 1$  if feedback is positive, otherwise  $y = 0$ .
- **GRU:** This baseline utilizes an RNN with GRU to predict the class of user's feedback to a recommended item. The input of each unit is  $i_n = (e_n, f_n)$ , and the output of RNN is the representation of user's preference, say  $u$ , then we concatenate  $u$  with the embedding of a recommended item, and leverage a *softmax* layer to predict the class of user's feedback to this item (output is a 2-dimensional vector).
- **GAN:** This baseline is based on Generative Adversarial Network [13], where the generator takes state-action pairs (the browsing histories and the recommended items) and outputs user's feedback (reward) to the items, while the discriminator takes (state, action, reward) tuples and distinguishes between real tuples (whose rewards are observed from historical logs) and fake ones. Note that we also use an RNN with GRU to capture user's sequential preference.
- **GAN-s:** This baseline is a supervised version of GAN [22], where the setting is similar with the above GAN baseline, while a supervise component is added on the output of the generator, which minimizes the difference between real feedback and predicted feedback.

The results are shown in Figure 6. We make the following observations:

- LR achieves worse performance than GRU, since LR neglects the temporal sequence within users' browsing history, while GRU

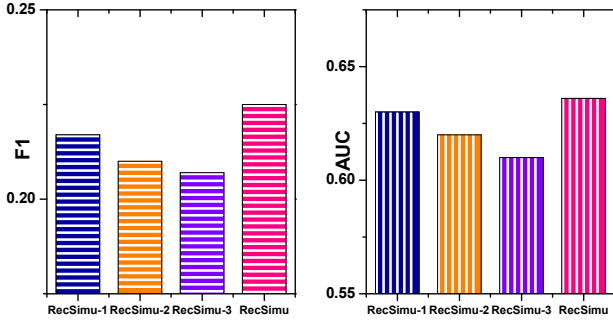


Figure 7: The results of component analysis.

can capture the temporal patterns within the item sequences and users' feedback for each item. This result demonstrates that it is important to capture the sequential patterns of users' browsing history when learning user's dynamic preference.

- GAN-s performs better than GRU and GAN, since GAN-s benefits from not only the advantages of the GAN framework (the unsupervised component), but also the advantages of the supervised component that directly minimizes the cross-entropy between the ground truth feedback and the predicted feedback.
- RecSimu outperforms GAN-s because the generator imitates the recommendation policy that generates the historical logs, and the generated logs can be considered as augmentations of real logs, which solves the data limitation challenge; while the discriminator can distinguish real and generated logs (unsupervised component), and simultaneously predict user's feedback of a recommended item (supervised component). In other words, RecSimu takes advantage of both the unsupervised and supervised components. The contributions of model components of RecSimu will be studied in the following subsection.

To sum up, the proposed framework outperforms the state-of-the-art baselines, which validates its effectiveness in simulating users' behaviors in recommendation tasks.

### 4.3 Component Analysis

To answer the second question, we systematically eliminate the corresponding components of the simulator by defining following variants of RecSimu:

- **RecSimu-1:** This variant is a simplified version of the simulator who has the same architecture except that the output of the discriminator is a 3-dimensional vector  $output = [l_{rp}, l_{rn}, l_f]$ , where each logit represents *real-positive*, *real-negative* and *fake* respectively, i.e., this variant will not distinguish the generated positive and negative items.
- **RecSimu-2:** In this variant, we evaluate the contribution of the supervised component  $L_G^{sup}$ , so we eliminate the impact of  $L_G^{sup}$  by setting  $\beta = 0$ .
- **RecSimu-3:** This variant is to evaluate the effectiveness of the competition between generator and discriminator, hence, we remove  $L_G^{unsup}$  and  $L_D^{unsup}$  from the loss function.

The results are shown in Figure 7. It can be observed:

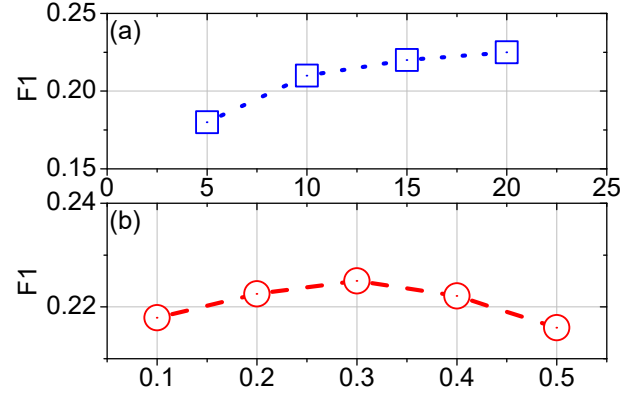


Figure 8: The results of parametric sensitivity analysis.

- RecSimu performs better than RecSimu-1, which demonstrates that distinguishing the generated positive and negative items can enhance the performance. This also validates that the generated data from the generator can be considered as augmentations of real-world data, which resolves the data limitation challenge.
- RecSimu-2 performs worse than RecSimu, which suggests that the supervised component is helpful for the generator to produce more indistinguishable items.
- RecSimu-3 first trains a generator, then uses real data and generated data to train the discriminator; while RecSimu updates the generator and discriminator iteratively. RecSimu outperforms RecSimu-3, which indicates that the competition between the generator (to capture complex data distribution) and the discriminator (to classify real and fake samples).

### 4.4 Parametric Sensitivity Analysis

Our method has two key parameters, i.e., (1)  $N$  that controls the length of state, and (2)  $\lambda$  that controls the contribution of the second term in Eq (13), which classifies the generated items into positive or negative class. To study the impact of these parameters, we investigate how the proposed framework RecSimu works with the changes of one parameter, while fixing other parameters. The results are shown in Figure 8. We have following observations:

- Figure 8 (a) demonstrates the parameter sensitivity of  $N$ . We find that with the increase of  $N$ , the performance improves. To be specific, the performance improves significantly first and then becomes relatively stable. This result indicates that introducing longer browsing history can enhance the performance.
- Figure 8 (b) shows the parameter sensitivity of  $\lambda$ . The performance for the simulator achieves the peak when  $\lambda = 0.3$ . In other words, the second term in Eq (13) indeed improves the performance of the simulator; however, the performance mainly depends on the first term in Eq (13), which classifies the real items into positive and negative classes.

## 5 RELATED WORK

In this section, we briefly review works related to our study. In general, the related work can be mainly grouped into the following categories.

The first category related to this paper is reinforcement learning based recommender systems, which typically consider the recommendation task as a Markov Decision Process (MDP), and model the recommendation procedure as sequential interactions between users and recommender system. Practical recommender systems are always with millions of items (discrete actions) to recommend. However, most RL-based models will become inefficient since they are not able to handle such a large discrete action space. A Deep Deterministic Policy Gradient (DDPG) algorithm is introduced to mitigate the large action space issue in practical RL-based recommender systems [10], where an Actor produces the optimal action based on current state, and a Critic outputs the action-value (Q-value) for this state-action pair. To avoid the inconsistency of DDPG and improve recommendation performance, a tree-structured policy gradient is proposed in [7], which constructs a balanced hierarchical clustering tree over items and pick an item through seeking a path to a specific leaf of the tree. Biclustering technique is also introduced to model recommender systems as grid-world games so as to reduce the state/action space [9]. To solve the unstable reward distribution problem in dynamic recommendation environments, approximate regretted reward technique is proposed with Double DQN to obtain a reference baseline from individual customer sample, which can effectively stabilize the reward value estimation and enhance the recommendation quality [8]. Users' positive and negative feedback, i.e., purchase/click and skip behaviors, are jointly considered in one framework to boost recommendations, since both types of feedback can represent part of users' preference [39]. Architecture aspect and formulation aspect improvement are introduced to capture both positive and negative feedback in a unified RL framework. A page-wise recommendation framework is proposed to jointly recommend a page of items and display them within a 2-D page [36, 40]. CNN technique is introduced to capture the item display patterns and users' feedback of each item in the page. In news feed scenario, a DQN based framework is proposed to handle the challenges of conventional models, i.e., (1) only modeling current reward like CTR, (2) not considering click/skip labels, and (3) feeding similar news to users [41]. Other applications includes sellers' impression allocation [3], fraudulent behavior detection [4] and user state representation [21].

The second category related to this paper is behavior simulation. Reinforcement learning and supervised learning algorithms typically learn experts' behavior with the guidance of the rewards, feedback or labels from real-world environment. However, deploying algorithms in real environment cost money and time, which calls for estimation of environment so as to train the algorithms to learn experts' behavior based on the simulation of the environment, before launching the algorithms online. One of the most effective approaches is Learning from Demonstration (LfD), which estimates implicit reward function from expert's behavior state to action mappings. Successful LfD applications include autonomous helicopter maneuvers [28], self-driving car [1], playing table tennis [5], object manipulation [27] and making coffee [32]. For example, Ross et

al. [28] develop a method that autonomously navigates a small helicopter at low altitude in a natural forest environment. Given the demonstration of a small group of human pilots, the authors leverage LfD techniques to train a controller to learn how a human expert would control a helicopter in similar environment to successfully avoid collisions with trees and leaves, using only low-weight visual sensors. Bojarski et al. [1] train a CNN to directly map the raw pixels of a single front-facing camera to the steering commands. With minimal human expert data, the system can automatically learn the representation of the environment, such as useful road features, only from the human driving angle as a training signal, and then learn to drive on local roads and highways with or without lane markings. Calinon et al. [5] propose a probabilistic method to train robust models of human motion by imitating, e.g., playing table tennis. The association of HMM, Gaussian mixture regression and dynamical systems enable the method to extract redundancy from multiple demonstrations and develop time-independent models to mimic the dynamic nature of the demonstration behaviors. Pastor et al. [27] present a general method to learn robot motor skills from human demonstrations. To represent an observed motion, the model learns a nonlinear differential equation that reproduces the motion. According to this representation, a library of movements is developed that marks each recorded motion based on the task and context, such as grasping, placing, and releasing. Sung et al. [32] proposed a manipulation planning approach according to the assumption that many household items share similar operational components. Thus the manipulation planning is formulated a structured prediction problem, and a DNN-based model is developed that can deal with large noise in manipulation demonstrations and can learn characteristics from three different patterns: point cloud, language, and trajectory. To gather a large number of manipulation demonstrations of different objects, the authors develop a new crowd-sourcing platform.

## 6 CONCLUSION

In this paper, we propose a novel user simulator RecSimu base on Generative Adversarial Network (GAN) framework, which models real users' behaviors from users' historical logs, and tackle the two challenges: (i) the recommended item distribution is complex within users' historical logs, and (ii) labeled training data from each user is limited. The GAN-based user simulator can naturally resolve these two challenges and can be used to pre-train and evaluate new recommendation algorithms before launching them online. To be specific, the generator captures the underlining item distribution of users' historical logs and generates indistinguishable fake logs that can be used as augmentations of real logs; and the discriminator is able to predict users' feedback of a recommended item based on users' browsing logs, which takes advantage of both supervised and unsupervised learning techniques. In order to validate the effectiveness of the proposed user simulator, we conduct extensive experiments based on real-world e-commerce dataset. The results show that the proposed user simulator can improve the user behavior prediction performance in recommendation task with significant margin compared with several state-of-the-art baselines.



There are several interesting research directions. First, for the sake of generalization, in this paper, we do not consider the dependency between consecutive actions, in other words, we split one recommendation session to multiple independent state-action pairs. Some recent techniques of imitation learning, such as Inverse Reinforcement Learning and Generative Adversarial Imitation Learning, consider a sequence of state-action pairs as a whole trajectory and the prior actions could influence the posterior actions. We will introduce this idea as one future work. Second, positive (click/purchase) and negative (skip) feedback is extremely unbalanced in users' historical logs, which makes it even harder to collect sufficient positive feedback data. In this paper, we leverage traditional up-sampling techniques to generate more training data of positive feedback. In the future, we consider leverage the GAN framework to automatically generate more data of positive feedback. Finally, users skip items for many reasons, such as (1) users indeed don't like the item, (2) users do not look the item in detail and skip it by mistake, (3) there exists a better item in the nearby position, etc. These reasons result in predicting skip behavior even harder. Thus, we will introduce explainable recommendation techniques to identify the reasons why users skip items.

## REFERENCES

- [1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
- [2] John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 43–52.
- [3] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for e-commerce. In *WWW '18*. <https://doi.org/10.1145/3178876.3186039>
- [4] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for Fraudulent Behaviour in e-Commerce. In *AAAI '18*. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16650>
- [5] Sylvain Calinon, Florent D'halluin, Eric L Sauser, Darwin G Caldwell, and Aude G Billard. 2010. Learning and reproduction of gestures by imitation. *IEEE Robotics & Automation Magazine* 17, 2 (2010), 44–54.
- [6] Chia-Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. 2006. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering* 18, 10 (2006), 1411–1428.
- [7] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2018. Large-scale Interactive Recommendation with Tree-structured Policy Gradient. *CoRR abs/1811.05869* (2018). [arXiv:1811.05869](http://arxiv.org/abs/1811.05869)
- [8] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing Reinforcement Learning in Dynamic Environment with Application to Online Recommendation. In *SIGKDD '18*. <https://doi.org/10.1145/3219819.3220122>
- [9] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. 2018. Reinforcement Learning based Recommender System using Biclustering Technique. *CoRR abs/1801.05532* (2018). [arXiv:1801.05532](http://arxiv.org/abs/1801.05532)
- [10] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
- [11] Jim Gao. 2014. Machine learning applications for data center optimization. (2014).
- [12] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline A/B testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 198–206.
- [13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
- [14] Hao Guo, Xin Li, Ming He, Xiangyu Zhao, Guiquan Liu, and Guandong Xu. 2016. CoSoLoRec: Joint Factor Model with Content, Social, Location for Heterogeneous Point-of-Interest Recommendation. In *International Conference on Knowledge Science, Engineering and Management*. Springer, 613–627.
- [15] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.
- [16] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. 2016. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research* 35, 11 (2016), 1289–1307.
- [17] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.
- [18] Lihong Li, Wei Chu, John Langford, Taesup Moon, and Xuanhui Wang. 2012. An unbiased offline evaluation of contextual bandit algorithms with generalized linear models. In *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation* 2. 19–36.
- [19] Lihong Li, Jin Young Kim, and Imed Zitouni. 2015. Toward predicting the outcome of an A/B experiment for search relevance. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. ACM, 37–46.
- [20] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
- [21] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. 2018. Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling. *arXiv preprint arXiv:1810.12027* (2018).
- [22] Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. 2016. Semantic segmentation using adversarial networks. *arXiv preprint arXiv:1611.08408* (2016).
- [23] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.
- [24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.
- [25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
- [26] Raymond J Mooney and Loriene Roy. 2000. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*. ACM, 195–204.
- [27] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. 2009. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 763–768.
- [28] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadepta Dey, J Andrew Bagnell, and Martial Hebert. 2013. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*. IEEE, 1765–1772.
- [29] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265–1295.
- [30] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
- [31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.
- [32] Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. 2018. Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *Robotics Research*. Springer, 701–720.
- [33] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. 2017. Adapting Markov Decision Process for Search Result Diversification. In *SIGIR '17*. <https://doi.org/10.1145/3077136.3080775>
- [34] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 279–287.
- [35] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Reinforcement Learning for Online Information Seeking. *arXiv preprint arXiv:1812.07127* (2018).
- [36] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Recommender Systems Conference*. ACM, 95–103.
- [37] Xiangyu Zhao, Long Xia, Yihong Zhao, Dawei Yin, and Jiliang Tang. 2019. Model-Based Reinforcement Learning for Whole-Chain Recommendations. *arXiv preprint arXiv:1902.03987* (2019).
- [38] Xiangyu Zhao, Tong Xu, Qi Liu, and Hao Guo. 2016. Exploring the Choice Under Conflict for Social Event Participation. In *International Conference on Database Systems for Advanced Applications*. Springer, 396–411.

- [39] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.
- [40] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. 2017. Deep Reinforcement Learning for List-wise Recommendations. *arXiv preprint arXiv:1801.00209* (2017).
- [41] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *WWW '18*. <https://doi.org/10.1145/3178876.3185994>