UserSim: User Simulation via Supervised Generative Adversarial Network

Xiangyu Zhao[1,2], Long Xia[3], Lixin Zou[4], Hui Liu[1], Dawei Yin[4], Jiliang Tang[1]
[1]Michigan State University, [2]City University of Hong Kong, [3]York University, [4]Baidu
{zhaoxi35,liuhui7,tangjili}@msu.edu,{long.phil.xia,zoulixin15}@gmail.com,yindawei@acm.org

ABSTRACT

With the recent advances in Reinforcement Learning (RL), there have been tremendous interests in employing RL for recommender systems. 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/effort consuming and could negatively impact users' experiences. Thus, it calls for a user simulator that can mimic real users' behaviors to pre-train and evaluate new recommendation algorithms. Simulating users' behaviors in a dynamic system faces immense challenges – (i) the underlying item distribution is complex, and (ii) historical logs for each user are limited. In this paper, we develop a user simulator based on a Generative Adversarial Network (GAN). To be specific, the generator captures the underlying distribution of users' historical logs and generates realistic logs that can be considered as augmentations of real logs; while the discriminator not only distinguishes real and fake logs but also predicts users' behaviors. The experimental results based on benchmark datasets demonstrate the effectiveness of the proposed simulator.

KEYWORDS

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

ACM Reference Format:
Xiangyu Zhao[1,2], Long Xia[3], Lixin Zou[4], Hui Liu[1], Dawei Yin[4], Jiliang Tang[1]. 2021. UserSim: User Simulation via Supervised Generative Adversarial Network. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3442381.3450125

1 INTRODUCTION

With the recent tremendous development in Reinforcement Learning (RL), there have been increasing interests in adapting RL for recommendations [5]. RL-based recommender systems treat the recommendation procedures as sequential interactions between users and a recommender agent (RA) 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; next, the system will update its recommendation strategy according to user's feedback. RL-based recommender systems aim to automatically learn an optimal recommendation strategy (policy) that maximizes cumulative rewards from users without any specific instructions. They can achieve two key advantages: (i) the recommender agent can learn their recommendation strategies based on users' real-time feedback during the user-agent interactions continuously; and (ii) the optimal strategy targets at maximizing the long-term reward from users (e.g., the overall revenue of a recommendation session). Given the advantages of reinforcement learning, very recently, it allures tremendous interest in developing RL-based recommender systems. [9, 43, 48].

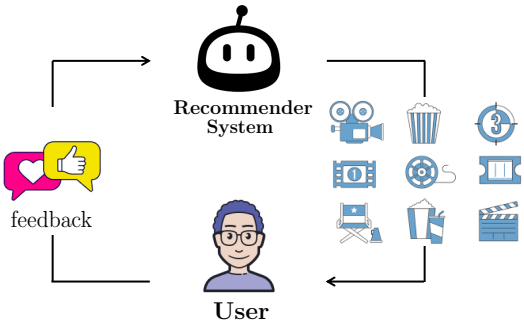

Figure 1: An example of system-user interactions.

RL-based recommendation algorithms are desired to be trained and evaluated based on users' real-time feedback (reward function). The most practical and precise way is online A/B test [20, 39], 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 inefficient and expensive: (i) online A/B tests usually take several weeks to collect sufficient data for the sake of statistical sufficiency, and (ii) numerous engineering efforts are typically required to deploy the new algorithm in the real system [14, 22, 38]. 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 [21]. These reasons prevent us from quickly training and testing new RL-based recommendation algorithms. The common practice to handle these challenges in the RL community is to 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 [12]. Thus, following the best routine, 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 in a dynamic recommendation environment is very challenging. First, the underlying distribution of recommended item sequences is extremely complex in

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450125

historical logs since there are millions of items in practical recommender systems. Second, 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. Recent efforts have demonstrated that Generative Adversarial Network (GAN) and its variants are able to generate fake but realistic images [15, 16], which implies their potential in modeling complex distributions. Furthermore, the generated images can be considered as augmentations of real images to enlarge the data space. Driven by these advantages, we propose to build a GAN-based user simulator (UserSim) for RL-based recommenders, which can capture the complex distribution of users' browsing logs and generate realistic logs to enrich the training dataset. We summarize our major contributions as follows: (i) We introduce a principled approach to capture the underlying distribution of recommended item sequences in historical logs, and generate realistic item sequences; (ii) We propose a user behavior simulator UserSim, which can be utilized to simulate environments with limited training data to pre-train and evaluate RL based recommender systems; and (iii) We conduct experiments based on real-world data to demonstrate the effectiveness of the proposed simulator and validate the contributions of its components.

2 THE PROPOSED SIMULATOR

This section 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.

2.1 Problem Statement

RL-based recommender systems treat the recommendation task as sequential interactions between a recommender system (agent) and users (environment $\mathcal{E}$), and use a Markov Decision Process (MDP) to model them, which consist of a sequence of states, actions and rewards: (i) We define the state $s = \{i_1, \cdots, i_N\}$ 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; (ii) An action a from the recommender system perspective is defined as recommending a set of items to the 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; (iii) When the system takes an action a based on the state s, the user will browse the recommended item and provide her feedback on the item, such as to skip, click, or purchase the item. The recommender system will then receive a reward $r(s, a)$ solely according to the type of feedback.

With the aforementioned definitions and notations, the goal of a simulator can be formally defined as follows: *Given a state-action pair (s, a), the goal is to imitate user's feedback (behavior) on a recommended item according to user's preference learned from the user's browsing history.*

2.2 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 2 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 $s = \{i_1, \cdots, i_N\}$ 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 to as p^E. Each item $i_n \in s$ involves two types of information: $i_n = (e_n, f_n)$, where e_n is a low-dimensional and dense item-embedding of the recommended item, and f_n is an embedding to denote user's feedback on the recommended item[1]. 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. Then, we concatenate e_n and f_n, and get a low-dimensional and dense vector: $I_n = concat(e_n, f_n)$.

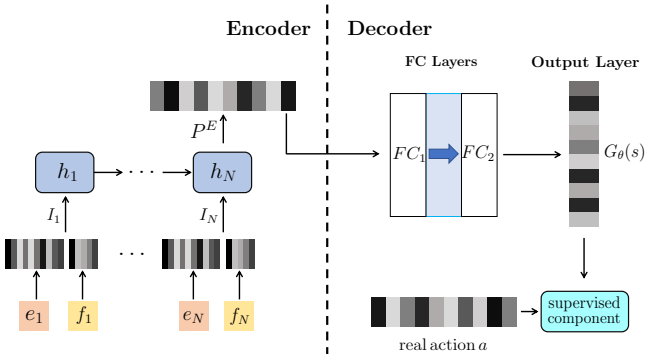

Figure 2: The generator with Encoder-Decoder architecture.

We introduce a Recurrent Neural Network (RNN) with Gated Recurrent Units (GRU) to capture the sequential patterns of items in the logs. We consider the RNN's final hidden state h_N as the output of Encoder component, i.e., the lower dimensional representation of user's current preference: $p^E = h_N$.

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 appear at next position in the log, referred to as $G_\theta(s)$. We leverage a MLP component with several fully-connected layers as the Decoder to directly transform p^E to $G_\theta(s)$. 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 2.4. Next, we will discuss the architecture of discriminator.

2.3 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 on 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

[1]These embeddings are jointly trained with neural networks in an end-to-end manner.

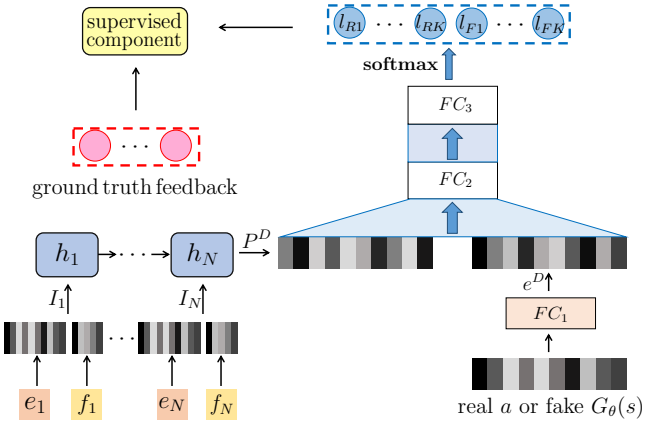

Figure 3: The discriminator architecture.

observed from historical logs, and K classes of *fake* feedback for the recommended items yielded by the generator.

Figure 3 illustrates the architecture of the discriminator. Similar with the generator, we introduce an RNN with GRU to capture user's dynamic preference. Note that the architecture is the same as the RNN in generator, but they have separate parameters. The input of the RNN is the state $s = \{i_1, \cdots, i_N\}$ observed in the historical logs, where $i_n = (e_n, f_n)$, and the output is the dense representation of the user's current preference, referred to as 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 to as e^D. Then we concatenate p^D and e^D, and feed the concatenation (p^D, e^D) into fully-connected layers, whose goals are (1) to judge whether the recommended items are real or fake, and (2) to 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:

$$output = [l_{R1}, \cdots, l_{RK}, l_{F1}, \cdots, l_{FK}] \tag{1}$$

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^{th} class is:

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

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 2.4.

2.4 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 on 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 to calculate the probability that a state-action pair is *real* or *fake*. From Eq (2), we know the probability that a state-action pair observed from historical logs is classified as *real*, referred to as $D_\phi(s,a)$, is the summation of the probabilities of K *real* feedback:

$$D_\phi(s,a) = \sum_{k=1}^{K} p_{model}(l_k|s,a) \tag{3}$$

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 *fake* feedback:

$$D_\phi(s, G_\theta(s)) = \sum_{k=K+1}^{2\times K} p_{model}(l_k|s, G_\theta(s)) \tag{4}$$

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

$$\begin{aligned} L_D^{unsup} = &- \{\mathbb{E}_{s,a\sim p_{data}} \log D_\phi(s,a) \\ &+ \mathbb{E}_{s\sim p_{data}} \log D_\phi(s, G_\theta(s))\} \end{aligned} \tag{5}$$

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_θ.

The supervised component aims 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 between ground truth class l_k 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 between ground truth class l_k and the predicted class for a fake state-action pair, where the action $G_\theta(s)$ is yielded by the generator. Thus, the supervised component of the loss function L_D is defined as follows:

$$\begin{aligned} L_D^{sup} = &-\{\mathbb{E}_{s,a,r\sim p_{data}}[\log p_{model}(l_k|s,a,k\leq K)] \\ &+ \lambda \cdot \mathbb{E}_{s,r\sim p_{data}}[\log p_{model}(l_k|s, G_\theta(s), K<k\leq 2K\} \end{aligned} \tag{6}$$

where λ 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 (6) 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. In other words, user will provide the same feedback if the generated item is sufficiently similar to the real one. The overall loss function of the discriminator L_D is defined as follows:

$$L_D = L_D^{unsup} + \alpha \cdot L_D^{sup} \tag{7}$$

Algorithm 1 Training Algorithm for the Simulator.

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

where parameter α 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 (5) with respect to G_θ. In other words, the first component minimizes that probabilities that fake state-action pairs are classified as *fake*, thus we have:

$$L_G^{unsup} = \mathbb{E}_{s \sim p_{data}}[\log D_\phi(s, G_\theta(s))] \tag{8}$$

where s is sampled from real historical logs distribution p_{data} and the action $G_\theta(s)$ is yielded by generator policy G_θ. Inspired by a supervised version of GAN [23], we introduce a supervised loss L_G^{sup} as the second component of L_G, which is the ℓ_2 distance between the ground truth item a and the generated item $G_\theta(s)$:

$$L_G^{sup} = \mathbb{E}_{s,a \sim p_{data}} \|a - G_\theta(s)\|_2^2 \tag{9}$$

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 generator L_G is defined as follows:

$$L_G = L_G^{unsup} + \beta \cdot L_G^{sup} \tag{10}$$

where β controls the contribution of the second component.

We detail our simulator training algorithm 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, 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.

Table 1: Statistics of the datasets.

Dataset	user (session)	item	interaction	ave. length
JD.com	283,228	1,355,255	97,713,660	345

3 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of the proposed simulator on real-world datasets.

3.1 Experimental Settings

We evaluate our method on public JD.com dataset[2]. The statistics are shown in Table 1. We consider the whole sequence of item-feedback pairs of each user as a session, and consider *click* as positive and *skip* as negative. For each session, we use first $N = 20$ 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 2.1. We collect the last 30% (state,action,reward) tuples from each session as the test set, while using the pervious 70% tuples as the training/validation 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 $|E| = 35$, and the dimension of feedback-embedding f_n is $|F| = 15$. The output of discriminator is a 4 ($i.e., K = 2$) dimensional vector of logits, and each logit represents *real-positive*, *real-negative*, *fake-positive* and *fake-negative* respectively:

$$output = [l_{rp}, l_{rn}, l_{fp}, l_{fn}] \tag{11}$$

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 the positive feedback such as a user clicks/purchases the recommended item; and *negative* denotes the negative feedback such as 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 (e.g., purchase and leave). AdamOptimizer is used for 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 hyper-parameters of we use in the proposed framework such as $N = 20$, $\lambda = 0.3$, $\alpha = 0.7$ and $\beta = 0.7$, we select them via cross-validation. Correspondingly, we also do parameter-tuning for baselines for a fair comparison.

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* [26] as the metric, which is a measure that combines precision and recall, namely the harmonic mean of precision and recall. Moreover, we leverage $p_{model}(l_{rp}|s, a)$ (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) [8] as the metric to evaluate the performance.

[2]https://datascience.jd.com/page/opendataset.html

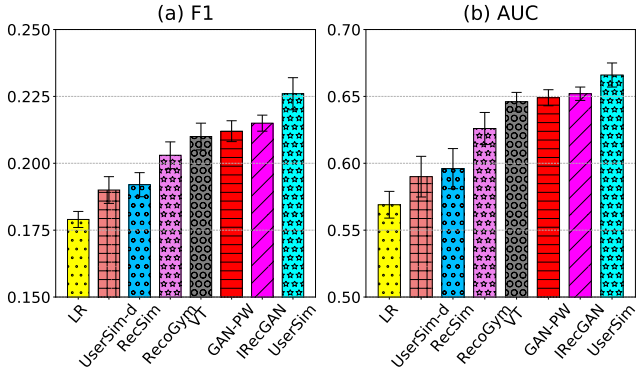

Figure 4: The results of overall performance comparison.

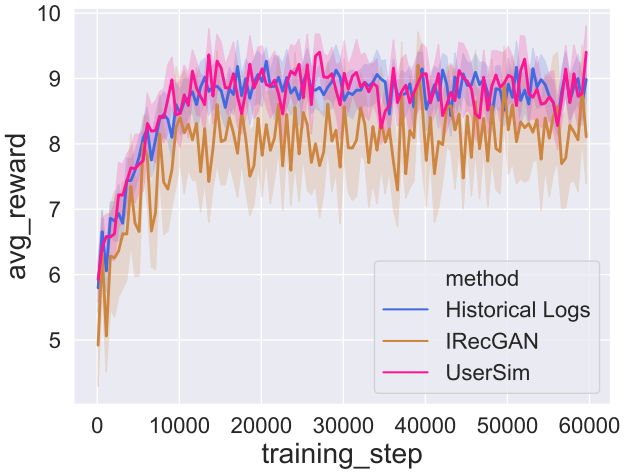

Figure 5: The training process of RL-based recommenders.

3.2 Overall Performance

We compare the proposed model with state-of-the-art baseline methods: **LR** [24], **UserSim-d**, **RecSim** [19], **RecoGym** [28], **VT** (Virtual-taobao) [31], **GAN-PW** [6] and **IRecGAN** [1]. UserSim-d has a similar architecture with the proposed discriminator. The differences are: we feed real recommended items as input action rather than both real and fake items; and in output layer, we predict the class of user's feedback to this real item without considering the fake feedback.

The overall performances of UserSim (discriminator) and baselines are shown in Figure 4. We make the following observations: (1) UserSim-d and RecSim outperform LR, since LR neglects the temporal sequence within users' browsing history, while UserSim-d and RecSim 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 users' preferences. (2) RecoGym achieves worse performance than VT and UserSim, since RecoGym is a bandit-based model that does not allow user state transitions. Furthermore, VT and UserSim are GAN-based models that are efficient at capturing the underlying distribution of item sequences. (3) UserSim outperforms UserSim-d, which shares a similar architecture with the proposed discriminator, but lacks the generator component. This observation validates that the generated logs can actually lead to improvements on feedback predictions. (4) UserSim performs better than VT, because VT is built upon two independent GANs trained separately, while UserSim jointly learns the sequence generation and user feedback prediction into one unified GAN framework. Also, UserSim takes advantage of both the unsupervised and supervised components, while VT consists of only unsupervised ones. (5) UserSim gets better than performance GAN-PW, the reasons involve: (i) UserSim leverages RNN with GRU to capture user's preference from browsing history, while GAN-PW uses positional weight separately without efficient weight sharing [32]; (ii) UserSim's supervision signal enhances its performance. (6) UserSim outperforms IRecGAN, where the user model and next fake item producer are both involved in IRecGAN's generator, and its discriminator only distinguishes real and fake items. This design breaks the naturally adversarial relationship between user model and next fake item producer (i.e., the producer aims to generate near-real items to fool user model, while user model targets distinguishing real and fake items), which leads to suboptimal performance.

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

3.3 RL-based Recommender Training

In this subsection, we evaluate the effectiveness of the proposed simulator on training reinforcement learning based recommender systems. Since real online environment is not available, we train a recommender based on a deep Q-network (DQN) framework. This off-policy reinforcement learning method can train on historical offline user behavior logs. We compare the training process of three recommenders with the same architecture, where the first is directly trained based on historical offline logs, the second is trained based on IRecGAN (the best baseline in Section 3.2), and the third is trained based on UserSim. Note that both IRecGAN and UserSim are learned upon the same historical offline logs. We use the averaged rewards on the test set (say *avg_reward*) as metric to evaluate the performance of recommenders.

Figure 5 illustrates the training process of the recommenders. We can observe that: (1) In the initial training stage, the *avg_reward* of all recommenders grow rapidly, then their growth speed gradually becomes slower with more interaction data. (2) When the recommenders achieve convergence, the recommender trained upon UserSim converges to the similar *avg_reward* value with the one trained upon offline logs, while IRecGAN's recommender converges to a distinctly different *avg_reward*. (3) The recommender trained upon UserSim performs much more stably than the one trained based upon IRecGAN.

To sum up, the above observations demonstrate that UserSim can effectively mimic user behaviors in real-world recommender systems. Therefore, it has the potential to take the place of real users to train RL-based recommender systems. Note that some

Table 2: Generator effectiveness.

Method	ROUGE-1	diff.	ROUGE-2	diff.
FM	0.312	-34.5%	0.143	-35.2%
W&D	0.336	-29.3%	0.159	-27.7%
Autorec	0.361	-24.2%	0.165	-24.9%
GRU4Rec	0.380	-20.1%	0.175	-21.2%
RRN	0.415	-13.2%	0.191	-13.6%
IRGAN	0.437	-8.19%	0.202	-8.59%
SSRM	0.452	-5.05%	0.208	-5.88%
UserSim	**0.476**	-	**0.221**	-

on-policy RL algorithms such as SARSA [34] cannot be directly trained on historical logs. Thus a simulator is necessary to train these RL algorithms before launching them online.

3.4 Effectiveness of Generator

Our proposed generator aims to generate indistinguishable logs (action) based on users' browsing history (state). In other words, it mimics the recommendation policy of the recommender system that generates the historical logs. The aforementioned comparison (UserSim v.s. UserSim-d) proves that the generated logs can enhance feedback predictions. Here, we investigate whether the proposed generator can generate indistinguishable logs. We train several representative recommendation algorithms based on the historical logs, then use them to generate a sequence of recommendations (with the same length of real logs), then compare the sequence similarity of real logs and generated logs. We compare the generator of the proposed generator with the following representative recommender methods: **FM** [27], **W&D** [7], **Autorec** [30], **GRU4Rec** [18], **RRN** [37], **IRGAN** [35], **SSRM** [17].

To evaluate the sequence similarity of real and generated logs, we select widely used metrics in NLP community **ROUGE** [11], where ROUGE-N measures the overlap ratio of N-grams between the real and generated sentences. The results are shown in Table 2. Compared with baselines, it can observe that the generator of UserSim could generate the most similar sequence with real-world logs. This result validates that the competition between the generator and discriminator and the proposed supervised components can enhance the generator to capture the complex item distribution in historical logs.

3.5 Component Anslysis

To study how the components in the generator and discriminator contribute to the performance, we systematically eliminate the corresponding components of the simulator by defining UserSim's following variants: (i) **UserSim-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., it will not distinguish the generated positive and negative items. (ii) **UserSim-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 $\beta = 0$. (iii) **UserSim-3**: This variant is to evaluate the adversarial training; hence, we remove L_G^{unsup} and L_D^{unsup} from loss function.

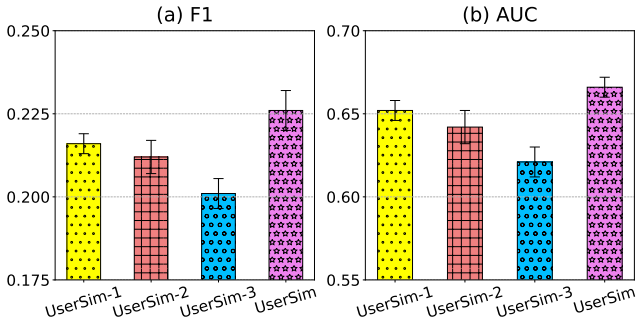

Figure 6: The results of component analysis.

The results are shown in Figure 6. It can be observed: (1) UserSim performs better than UserSim-1, which demonstrates that distinguishing the generated positive and negative items can enhance the performance. This also validates that the generator's output can be considered as augmentations of real-world data, which resolves the data limitation challenge. (2) UserSim-2 performs worse than UserSim, which suggests that the supervised component helps the generator to produce more indistinguishable items. (3) UserSim-3 first trains a generator, then uses real data and generated data to train the discriminator; while UserSim updates the generator and discriminator iteratively. UserSim outperforms UserSim-3, which indicates that the adversarial training can enhance both the generator (to capture complex data distribution) and the discriminator (to classify real and fake samples).

3.6 Parametric Sensitivity Analysis

Our method has several key parameters, i.e., (1) N that controls the length of state; (2) λ that controls the contribution of the second term in Eq (6), which classifies the generated items into positive or negative class; (3) α that adjusts the importance of supervised component in discriminator; and (4) β that calibrates the supervised component in generator. To study the impact of these parameters, we investigate how the proposed framework UserSim works with the changes of one parameter, while fixing other parameters.

The results are shown in Figure 7. We have following observations: (1) Figure 7 (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. (2) Figure 7 (b) shows the sensitivity of λ. The performance for the simulator achieves the peak when $\lambda = 0.3$. In other words, the second term in Eq (6) indeed improves the performance of the simulator; however, the performance mainly depends on the first term in Eq (6), which classifies the real items into positive and negative classes. (3) Figure 7 (c) - (d) illustrate the model performance with respect to α and β. We can observe that the simulator achieves its optimal performance when $\alpha = 0.7$ and $\beta = 0.7$, while a lower/higher value of both parameters will lead to a lower F1. These observations validate that the supervised components can enhance the performance.

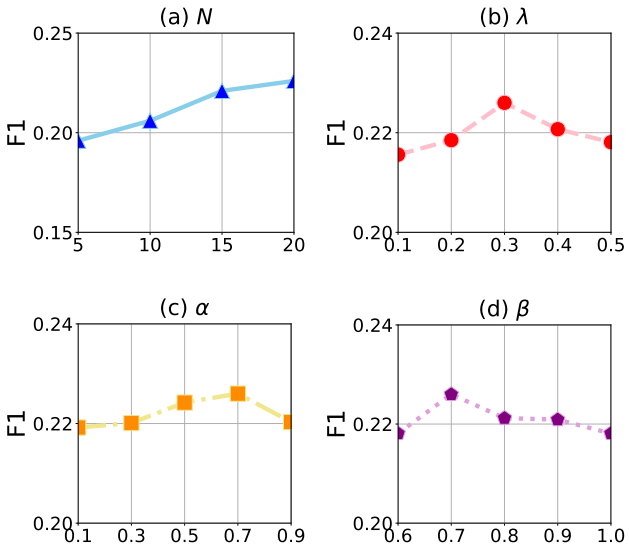

Figure 7: The results of parametric analysis.

4 RELATED WORK

In this section, we briefly review works related to our study. In general, the related work can be 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[40, 42]. A Deep Deterministic Policy Gradient (DDPG) algorithm is introduced to mitigate the large action space issue in practical RL-based recommender systems [9], 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. Users' positive and negative feedback are jointly considered in one framework to boost recommendations [45]. A page-wise framework is proposed to jointly recommend a page of items and display them within a 2-D page [43, 46]. A multi-agent reinforcement learning based approach (DeepChain) is proposed in to jointly optimize multiple recommendation strategies among sequential scenarios [44]. A unified framework is proposed to to jointly optimize user experience of recommendations and revenue of advertisements [41, 47]. In news feed scenario, a DQN based framework is proposed to handle the challenges of conventional models [48]. Other applications includes sellers' impression allocation [3], cold-start problem [52], long-term engagement [49] and fairness [13], top-N recommendation [50], attacking black-box recommendations [10] and spatial recommendation [36].

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 to train the algorithms to learn experts' behavior based on the simulation of the environment, before launching the algorithms online [1, 6, 19, 31]. 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 [29], self-driving car [2], playing table tennis [4], object manipulation [25] and making coffee [33]. For example, Ross et al. [29] develop a method that autonomously navigates a small helicopter at low altitude in a natural forest environment. Bojarski et al. [2] train a CNN to directly map the raw pixels of a single front-facing camera to the steering commands. Calinon et al. [4] propose a probabilistic method to train robust models of human motion by imitating, e.g., playing table tennis. Sung et al. [33] proposed a manipulation planning approach according to the assumption that many household items share similar operational components. Zou et al. [51] propose a customer simulator, referred to as the World Model, which is designed to simulate the environment and handle the selection bias of logged data.

5 CONCLUSION

In this paper, we propose a novel user simulator, UserSim, based 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 underlying item distribution of users' historical logs and generates indistinguishable fake logs that can be used as augmentations of real logs. The discriminator can predict users' feedback of a recommended item based on users' browsing logs, taking 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 benchmark datasets. The results show that the proposed user simulator can improve the user behavior prediction performance in recommendation tasks over representative baselines.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation (NSF) under grant numbers IIS1907704, IIS1928278, IIS1714741, IIS1715940, IIS1845081 and CNS1815636.

REFERENCES

[1] Xueying Bai, Jian Guan, and Hongning Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. In *Advances in Neural Information Processing Systems*. 10735–10746.
[2] 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).
[3] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for e-commerce. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 1339–1348.
[4] 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.
[5] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[6] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System. In *International Conference on Machine Learning*. 1052–1061.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 7–10.

[8] Corinna Cortes and Mehryar Mohri. 2004. AUC optimization vs. error rate minimization. In *Advances in neural information processing systems*. 313–320.

[9] 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).

[10] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2020. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. *arXiv preprint arXiv:2005.08147* (2020).

[11] Mamdouh Farouk. 2019. Measuring sentences similarity: a survey. *arXiv preprint arXiv:1910.03940* (2019).

[12] Jim Gao. 2014. Machine learning applications for data center optimization. (2014).

[13] Yingqiang Ge, Shuchang Liu, Ruoyuan Gao, Yikun Xian, Yunqi Li, Xiangyu Zhao, Changhua Pei, Fei Sun, Junfeng Ge, Wenwu Ou, et al. 2021. Towards Long-term Fairness in Recommendation. *arXiv preprint arXiv:2101.03584* (2021).

[14] 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.

[15] 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.

[16] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. 2020. A review on generative adversarial networks: Algorithms, theory, and applications. *arXiv preprint arXiv:2001.06937* (2020).

[17] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming session-based recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1569–1577.

[18] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[19] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. *arXiv preprint arXiv:1909.04847* (2019).

[20] Ron Kohavi and Roger Longbotham. 2017. Online Controlled Experiments and A/B Testing. *Encyclopedia of machine learning and data mining* 7, 8 (2017), 922–929.

[21] 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.

[22] 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.

[23] Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. 2016. Semantic segmentation using adversarial networks. *arXiv preprint arXiv:1611.08408* (2016).

[24] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.

[25] 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.

[26] David Martin Powers. 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. (2011).

[27] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[28] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. *arXiv preprint arXiv:1808.00720* (2018).

[29] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta 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.

[30] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*. 111–112.

[31] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4902–4909.

[32] Hyungseok Song, Hyeryung Jang, Hai Tran Hong, Seeun Yun, Donggyu Yun, Hyoju Chung, and Yung Yi. 2019. Solving Continual Combinatorial Selection via Deep Reinforcement Learning. (2019).

[33] 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.

[34] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[35] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 515–524.

[36] Yanan Wang, Tong Xu, Xin Niu, Chang Tan, Enhong Chen, and Hui Xiong. 2020. STMARL: A Spatio-Temporal Multi-Agent Reinforcement Learning Approach for Cooperative Traffic Light Control. *IEEE Transactions on Mobile Computing* (2020).

[37] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 495–503.

[38] 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.

[39] Scott WH Young. 2014. Improving library user experience with A/B testing: Principles and process. *Weave: Journal of Library User Experience* 1, 1 (2014).

[40] Weinan Zhang, Xiangyu Zhao, Li Zhao, Dawei Yin, Grace Hui Yang, and Alex Beutel. 2020. Deep Reinforcement Learning for Information Retrieval: Fundamentals and Advances. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2468–2471.

[41] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. 2019. Deep Reinforcement Learning for Online Advertising in Recommender Systems. *arXiv preprint arXiv:1909.03602* (2019).

[42] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: a survey by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring (2019), 4.

[43] 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.

[44] Xiangyu Zhao, Long Xia, Lixin Zou, Hui Liu, Dawei Yin, and Jiliang Tang. 2020. Whole-Chain Recommendations. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1883–1891.

[45] 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.

[46] 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).

[47] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3319–3327.

[48] 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 *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 167–176.

[49] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.

[50] Lixin Zou, Long Xia, Zhuoye Ding, Dawei Yin, Jiaxing Song, and Weidong Liu. 2019. Reinforcement Learning to Diversify Top-N Recommendation. In *International Conference on Database Systems for Advanced Applications*. Springer, 104–120.

[51] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 816–824.

[52] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 749–758.