UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Keeping Dataset Biases out of the Simulation

A Debiased Simulator for Reinforcement Learning based Recommender Systems

Huang, J.; Oosterhuis, H.; de Rijke, M.; van Hoof, H.

DOI
10.1145/3383313.3412252

Publication date
2020

Document Version
Final published version

Published in
RECSYS 2020

License
Article 25fa Dutch Copyright Act (https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment)

Link to publication

Citation for published version (APA):
Huang, J., Oosterhuis, H., de Rijke, M., & van Hoof, H. (2020). Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *RECSYS 2020: 14th ACM Conference on Recommender Systems : Virtual Event, Brazil, September 22-26, 2020* (pp. 190–199). The Association for Computing Machinery. https://doi.org/10.1145/3383313.3412252

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:13 Oct 2026

Keeping Dataset Biases out of the Simulation

A Debiased Simulator for Reinforcement Learning based Recommender Systems

Jin Huang
University of Amsterdam
Amsterdam, The Netherlands
j.huang2@uva.nl

Harrie Oosterhuis
University of Amsterdam
Amsterdam, The Netherlands
oosterhuis@uva.nl

Maarten de Rijke
University of Amsterdam & Ahold Delhaize
Amsterdam, The Netherlands
m.derijke@uva.nl

Herke van Hoof
University of Amsterdam
Amsterdam, The Netherlands
h.c.vanhoof@uva.nl

ABSTRACT

Reinforcement learning for recommendation (RL4Rec) methods are increasingly receiving attention as an effective way to improve long-term user engagement. However, applying RL4Rec online comes with risks: exploration may lead to periods of detrimental user experience. Moreover, few researchers have access to real-world recommender systems. Simulations have been put forward as a solution where user feedback is simulated based on logged historical user data, thus enabling optimization and evaluation without being run online. While simulators do not risk the user experience and are widely accessible, we identify an important limitation of existing simulation methods. They ignore the interaction biases present in logged user data, and consequently, these biases affect the resulting simulation. As a solution to this issue, we introduce a debiasing step in the simulation pipeline, which corrects for the biases present in the logged data before it is used to simulate user behavior. To evaluate the effects of bias on RL4Rec simulations, we propose a novel evaluation approach for simulators that considers the performance of policies optimized with the simulator. Our results reveal that the biases from logged data negatively impact the resulting policies, unless corrected for with our debiasing method. While our debiasing methods can be applied to any simulator, we make our complete pipeline publicly available as the Simulator for OFfline leArning and evaluation (SOFA): the first simulator that accounts for interaction biases prior to optimization and evaluation.

CCS CONCEPTS

• **Computing methodologies** → **Interactive simulation**; **Reinforcement learning**; • **Information systems** → *Information retrieval*; **Recommender systems**.

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.
RecSys '20, September 22–26, 2020, Virtual Event, Brazil
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7583-2/20/09…$15.00
https://doi.org/10.1145/3383313.3412252

KEYWORDS

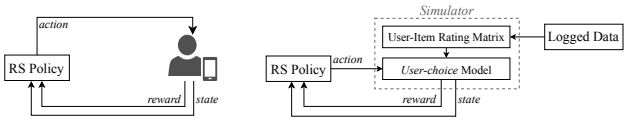
Reinforcement learning, Recommender systems, Simulation, Interaction bias

ACM Reference Format:
Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. 2020. Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383313.3412252

1 INTRODUCTION

In recent years, interest in Reinforcement Learning for Recommendation (RL4Rec) has greatly increased in both academia and industry. The key idea behind Reinforcement Learning (RL) is to optimize a policy that matches states to actions, so that an agent performing these actions maximizes a cumulative reward [31]. RL does not just consider the immediate reward of an action, but also the effect it has on subsequent actions, allowing it to learn w.r.t. long-term goals. For Recommender Systems (RSs), long-term goals are usually some form of *long-term user engagement, e.g.,* the cumulative number of clicks or the dwell time over sessions of multiple recommendations [42]. Furthermore, RL is particularly suited for exploring the item space over multiple interactions [41], learning a recommendation policy directly from complex recommendation scenarios [7, 32, 40, 41], and adapting quickly to real-time user feedback [37]. Figure 1(a) displays the typical flow of RL4Rec: a state is the historical interactions of a user who is about to receive a recommendation, an action is an item being recommended by the policy of the recommender system, and the reward is implicit or explicit user feedback (*e.g.,* a click, a rating, dwell time, an order, etc.). The goal of RL4Rec is to maximize the cumulative reward over multiple sequential recommendations. RL methods learn from experience; in the RS setting this means that they learn by recommending items to users and observing their subsequent interactions.

Despite these advantages, the RL4Rec approach brings risks when applied online: during learning, exploratory or incorrect actions could be taken, which can be detrimental to the user experience [11, 17]. Since RL learns from experience, it is almost unavoidable that initially some disliked items are recommended. Furthermore, online deployment takes time, costs money, and many researchers – both in academia and industry – simply do not have access to an actual platform with live users.



(a) RL4Rec online. (b) RL4Rec with a simulator.

Figure 1: The general framework of RL4Rec, where a state is user historical interactions, an action is an item being recommended by the RS, and a reward is related to user feedback. (a) shows RL4Rec applied online to interact with actual users. (b) shows how RL4Rec typically interacts with a simulation-based environment.

An alternative to online experimentation is provided by *simulation-based* experiments. Here, user feedback on items is simulated in order to enable learning and evaluating RL-based RSs (see Figure 1(b)). Recently, several simulators have been proposed, specifically for RL4Rec [13, 16, 22, 25, 26, 35, 36, 41]. Some work is designed for specific datasets and specific recommendation tasks [16, 26, 35, 41], which makes them unavailable without access to similar data, and inapplicable for simulating more general recommendation tasks. For better applicability, other work has proposed simulators based on fully synthetic data which is completely generated by statistical distribution functions (*e.g.,* Bernoulli distribution) [22]. The fully synthetic approach has been criticized because it oversimplifies user behavior [25]. As a result, the resulting simulated behavior is dissimilar to the complex behavior of real users, and often recognized as unrealistic feedback. To simulate user behavior while maintaining many of its natural complexities, others have proposed to simulate user feedback based on datasets of user logged data [13, 25, 36]. These simulators usually follow user preferences in the logged data (*i.e.,* ratings provided by users) by basing their simulated behavior on them. E.g., a click on an item is more likely to be simulated for a user if in the logged data this user gave a high rating to this specific item. Logged-data based simulators avoid oversimplifying preferences, while still providing simulated user interactions for RL4Rec methods.

While these simulators allow for offline learning, we recognize two significant limitations: they ignore the biases present in logged data; and they have not been evaluated based on the performance of their produced policies. Biases are very prevalent in user-RS interaction data. Two influential types of biases are *popularity bias* and *positivity bias*. Popularity bias occurs because users tend to interact with more popular items [21, 29], which results in the commonly observed long tail distribution of the number of interactions per item in logged data. Positivity bias occurs because users rate the items they like more often [21], which leads to positive feedback being over-represented. These types of biases in logged data may lead to biased parameter estimation and prediction in many methods [19, 24], *e.g.,* it is known to affect Matrix Factorization (MF) [24]. Nevertheless, previous work on simulators has ignored biases and naively uses the observed user-item interaction data when simulating user behavior. As a result, we can expect these biases to affect the simulator and the feedback it generates. For instance, due to positivity bias, we can expect simulated users to be more positive to most items than actual users would be. Consequently, a policy learned with such a simulator would also be affected by the biases in the logged data. These biased policies may result in detrimental performance if exposed to actual users [3, 19, 24, 28]. Hence, there is a need for a simulator that is based on logged data, but that mitigates the effect of bias in the data.

Existing work has evaluated RL4Rec simulators by comparing simulated user feedback with real user feedback from logged data. For instance, some work evaluated the performance of a simulator by considering how well it predicts skip/click behaviors [36] or dwell time [40]. While this type of evaluation can simulate a single user interaction, it does not consider whether using a simulator actually leads to a well-performing policy. However, there is no work that directly considers the performance of the produced policies that result from using a simulator, despite this being the ultimate goal. E.g., if one wants to apply a policy learned in a simulator to a real-world RL4Rec setting, it is generally desired that the policy has the best performance possible. Moreover, simulators can be very effective ways to reproduce and benchmark RL4Rec methods, but such comparisons are considerably less reliable if their results are biased.

In this paper, we propose a debiasing method for RL4Rec simulators that mitigates the effect of bias in logged data. Furthermore, we introduce a novel way of evaluating the effect of bias on the final policy performance of a simulator. Our experimental results reveal that bias in logged data affects simulators and the policies they produce. While both of these contributions can be applied to any RL4Rec simulator, we combine both steps in a newly proposed Simulator for OFfline leArning and evaluation (SOFA). SOFA bases its simulation on a user-item rating matrix learned from logged user data; unlike existing simulators, SOFA corrects for interaction bias when learning this matrix. To evaluate SOFA, we use publicly available datasets where part of the data was logged on randomly recommended items.

The main contributions of this work are as follows:

(1) A novel approach for debiasing simulators that mitigates the effect of bias in logged data.
(2) A novel evaluation method to analyze the effect of bias on RL4Rec.
(3) Two types of experiments, both based on real-world datasets (Yahoo!R3 [19] and coat [24]) and based on a simulation study, that show that bias in logged data affects simulators and the policies they produce.
(4) SOFA, a novel simulator for RL4Rec, the first that corrects for bias in logged data.

We release the code of SOFA[1] so that future work can develop RL4Rec algorithms while mitigating the effect of bias.

2 BACKGROUND: REINFORCEMENT LEARNING FOR RECOMMENDATION

RL methods are commonly studied in the context of an Markov Decision Process (MDP), consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R}$, the transition probabilities $\mathcal{T}$, and a discount factor γ [31]. We will now describe how we model the recommendation task as an MDP [4, 5, 37, 42]:

State space $\mathcal{S}$: A state represents all the current information on which a decision can be based. For RL4Rec, a state $s_t^u \in \mathcal{S}$ stores historical interactions of user u till the t-th turn of interaction, consisting of the recommended items and the corresponding feedback, denoted as $s_t^u = ([i_1, i_2, \ldots, i_t], [f_1, f_2, \ldots, f_t])$, with i_k the item recommended by the RS in turn k, and f_k the corresponding user feedback. The initial state $s_0^u = ([],[])$ is always empty. While contextual information about the user could be

[1] See https://github.com/BetsyHJ/SOFA.

part of the state, in our experiments such information is not available.

Action space $\mathcal{A}$: Action $a_t \in \mathcal{A}$ taken by the RS consists of the recommendation of a single item i_t in turn t.

Reward $\mathcal{R}$: After receiving action a_t, consisting of item i_t being recommended by the RS, the (simulated) user gives feedback $f_t \in \{0,1\}$ (*i.e.,* skip or click) on this item. This feedback is used to generate the immediate reward $r_t = \mathcal{R}(f_t)$.

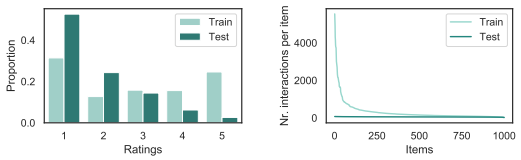
Transition probabilities $\mathcal{T}$: After the user provides feedback f_{t+1} on item i_{t+1}, the state transitions deterministically to the next state $s^u_{t+1} = ([i_1,\ldots,i_{t+1}],[f_1,\ldots,f_{t+1}])$. The interaction terminates after 10 turns.

Discount factor γ: As usual in MDPs, $\gamma \in [0,1]$ aims to balance the effect of immediate rewards and future rewards. At its extremes, if $\gamma = 0$, the RS only considers the immediate reward when taking an action. When $\gamma = 1$, all future rewards will be taken into account evenly.

This completes our description of our RL4Rec MDP, which allows us to apply RL methods to recommendation. The main difference between RL4Rec and the traditional recommendation task is that RL4Rec methods: (i) make multiple sequential recommendations while keeping track of previous interactions with a user, and (ii) try to optimize the cumulative rewards, based on a discounted sum on the observed user feedback f_t, the reward function $\mathcal{R}$, and the discount factor γ. Unlike the traditional recommendation setup, RL4Rec considers the long-term feedback/rewards an RS receives. We further discuss related work on RL4Rec in Section 4.

3 BACKGROUND: INTERACTION BIAS IN LOGGED USER DATA

The recommendation task traditionally has a user set $\mathcal{U} = \{u_1,\ldots,u_N\}$ on the one hand and an item set $\mathcal{I} = \{i_1,\ldots,i_M\}$ on the other hand. Logged user data is usually an observed rating matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times M}$. One slot $y_{u,i}$ in this rating matrix $\boldsymbol{Y}$ denotes the rating user u would give to item i. In practice, the complete rating matrix is rarely known, since users usually do not rate every available item. We use $\mathbf{O} \in \{0,1\}^{N\times M}$ as an observation indicator: $o_{u,i} = 1$ if we observe the rating $y_{u,i}$ given by user u on item i, otherwise $o_{u,i} = 0$. In reality, observed user behavior can be affected by many types of interaction bias. Figure 2 visualizes the effect of positivity bias and popularity bias on the Yahoo!R3 dataset [19]. Positivity bias occurs because users rate the items they like more often [21, 29] and results in positive feedback being over-represented. In Figure 2(a), the naturally observed ratings in the training set (Train) are compared with the test set (Test) where users were provided ratings on randomly selected items. On the randomly selected items we see only 2.6% of ratings are 5, while in the naturally logged data, the proportions of 5 are about 24.6%. Clearly, the natural user behavior results in a large "*oversampling*" of positive feedback. In contrast, popularity bias occurs because users tend to interact more with popular items [21]. Figure 2(b) shows the number of interactions per item in the logged data and reveals a clear long-tail distribution. As a result of types of bias like these, the logged data is not uniform-randomly observed, and the missing slots in the rating matrix are Missing Not At Random (MNAR) [19].



(a) Positivity bias. (b) Popularity bias.

Figure 2: Examples of positivity bias and popularity bias on the Yahoo!R3 dataset [19]. (a) shows positivity bias caused by the fact that users rate items they like more often. (b) shows popularity bias caused by the fact that users tend to interact with more popular items.

3.1 Forms of Bias

Formally, MNAR is often modelled by separating the probability of observance and the probability of a rating. Generally, the rating $y_{u,i}$ a user would give is not conditioned on whether the rating is given or not:

$$P(y_{u,i}, o_{u,i}) = P(o_{u,i} \mid y_{u,i}) P(y_{u,i}). \tag{1}$$

We will now illustrate how this model can capture different forms of bias:

(i) **No Bias** – if every rating is equally likely to be observed, the ratings are not MNAR but Missing Completely At Random (MCAR) and all users and items are equally represented:
$$\forall (u,u') \in \mathcal{U}, (i,i') \in \mathcal{I} \left(P(o_{u,i}) = P(o_{u',i'}) \right). \tag{2}$$

(ii) **Positivity Bias** – when positivity bias is present, items that would receive a higher rating are more likely to be given a rating. One way to model positivity bias is to state that if an item is more preferred it is also more likely to be given a rating:
$$\forall u \in \mathcal{U}, (i,i') \in \mathcal{I} \left(y_{u,i} > y_{u,i'} \rightarrow P(o_{u,i}) > P(o_{u,i'}) \right). \tag{3}$$

(iii) **Popularity Bias** – when popularity bias is present, items that are more popular are more likely to be given a rating. Let $\text{pop}(i)$ denote the popularity of an item; we can model popularity bias by stating that if an item is more popular it is also more likely to be given a rating:
$$\forall u \in \mathcal{U}, (i,i') \in \mathcal{I} \left(\text{pop}(i) > \text{pop}(i') \rightarrow P(o_{u,i}) > P(o_{u,i'}) \right). \tag{4}$$

Now that we have described MNAR types of bias, we can consider the effect they may have on RSs and RL4Rec simulators.

3.2 Effect of Bias on Rating Estimation and User Simulation

Without correction, the types of interaction bias identified above will affect rating prediction, and may thus further influence the RL4Rec simulators and the policies they help produce (see Figure 3). To illustrate how this may happen, we will use a simple example to estimate the average rating of an item. Let $\text{avg}(i)$ be the *true* average rating: $\text{avg}(i) = \frac{1}{N}\sum_{u\in\mathcal{U}} y_{u,i}$; the naive (uncorrected) estimate is simply the average of the observed ratings:

$$\widehat{\text{avg}}(i) = \frac{1}{\sum_{u\in\mathcal{U}} \mathbb{1}[o_{u,i}=1]} \sum_{u\in\mathcal{U}: o_{u,i}=1} y_{u,i}. \tag{5}$$

In expectation, this naive estimate is affected by the observance probabilities:

$$\mathbb{E}_o[\widehat{\text{avg}}(i)] = \frac{1}{\sum_{u\in\mathcal{U}} P(o_{u,i}=1)} \sum_{u\in\mathcal{U}} P(o_{u,i}=1) \cdot y_{u,i}. \tag{6}$$

If we compare the expected average rating estimate with the forms of bias discussed in Section 3.1, we see the following: (i) **No Bias** – if no bias is present (Eq. 2) the estimate is correct in expectation: $\mathbb{E}_o[\widehat{\text{avg}}(i)] = \text{avg}(i)$. (ii) **Positivity Bias** – if positivity bias is present (Eq. 3), the estimate is expected to overestimate the true rating: $\mathbb{E}_o[\widehat{\text{avg}}(i)] \geq \text{avg}(i)$. This happens because higher ratings are over-represented in observance, thus the average is skewed upwards. (iii) **Popularity Bias** – popularity bias (Eq. 4) will also affect the estimate, however, it depends on how popularity is distributed. The more popular items will have a heavier influence on the estimate, thus, if more popular items are highly rated on average, it will overestimate. Conversely, it will underestimate if more popular items are lowly rated on average.

While these effects go beyond estimating the average rating, understanding the effect of bias in this simple case helps us understand the effect it has on rating prediction. E.g., if a model is trained to predict ratings on the observed ratings, then under positivity bias we can expect it to overestimate ratings on average. For the same reasons overestimation happens on the expected average estimate: the model is trained on a sample of ratings where positive ratings were oversampled [24]. In turn, RL4Rec simulators are often based on a predicted rating matrix, and clicks are more likely to occur if an item i is recommended to a user i where a high rating $\hat{y}_{u,i}$ was predicted. Consequently, if logged data contains positivity bias, we would expect a simulator based on that data to simulate users to click more often due to the bias. In contrast, if users were asked to rate randomly sampled items, resulting in MCAR data, then we expect simulated users to click less on average. With more complicated forms of bias, such as popularity bias, the effects of the bias on the final user become less predictable. Nonetheless, without intervention we can expect bias in logged data to affect the simulated users, and unavoidably, it will thus also result in different learned RL4Rec policies. Therefore, it is important to understand the effects of interaction bias on simulations, and to develop methods for mitigating them.

4 RELATED WORK

RL-based Recommendation. Dulac-Arnold et al. [10] apply a Deep Deterministic Policy Gradient (DDPG) algorithm to improve the efficiency of recommender systems with a large number of items. Following this framework, Chen et al. [4] propose a tree-structured policy gradient recommendation framework, where a balanced hierarchical clustering tree is built over the items and picking an item is formulated as seeking a path from the root to a certain leaf of the tree. A branch of research has used Deep Q-Networks (DQNs) (or variants thereof) to improve recommendation performance. Zhao et al. [38] adapt a DQN architecture to incorporate positive and negative feedback of users. Others use DQN to deal with some special recommendation scenarios, such as tip recommendation [7], news recommendation [41], and recommendation mixed with advertisements [40]. Another line of research applies the Actor-Critic framework, which combines the advantages of Q-Learning and policy gradients for accelerated and stable learning. The Actor-Critic architecture is more suitable for large and dynamic action spaces and can reduce redundant computations when dealing with more complex recommendation scenarios, such as, *e.g.,* list-wise recommendation [39], page-wise recommendation [37], and dynamic treatment recommendation [32]. Choi et al. [9] use biclustering to reduce the state and action space, making the resulting MDP easy to solve with RL. Chen et al. [5] propose a policy-gradient-based algorithm that corrects for bias caused by the unobserved feedback of actions not chosen by the previous RS. Zhang et al. [34] introduce a hierarchical RL framework to improve the diversity of recommender systems.

Debiased Recommender Systems. Debiased recommendation focuses on estimating the bias (*e.g.,* positivity bias [21], popularity bias [21, 29]), and correcting for them. Existing work on debiasing mostly focuses on missing interactions (*e.g.,* missing ratings) between users and items, and considers the case when they are Missing Not At Random (MNAR). When missing data is Missing Completely At Random (MCAR), maximum likelihood inference that is only based on the observed data is unbiased because of the key property of Missing At Random (MAR) condition that the observation process is independent of the value of unobserved data [12, 19]. In contrast, MNAR data fails to have this key property and will probably lead to biased parameter estimation and prediction because of using the incorrect likelihood function.

Methods proposed for debiasing MNAR data can be grouped into three categories. The first category applies missing data imputation on MNAR data with the joint likelihood of modeling rating prediction and the observation process [12, 19, 20]. The rating prediction model is meant to complete the rating matrix, while the observation process model is meant to learn how the data point is missing according to its value. The second category makes use of Inverse Propensity Scoring (IPS) from causal inference [14], and integrates it in the learning process [6, 15, 24]. Based on IPS, it is able to derive an unbiased estimator for a wide range of performance estimators, such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) used in rating prediciton models. This type of debiasing work, which separates the estimation of bias from recommendation models, makes it flexible to plug in any conditional probability estimation method as the propensity estimator [24]. The third category is a hybrid method that integrates the above two methods so as to obtain robust performance by avoiding the potentially large bias due to imputation inaccuracy and the high variance of the propensities [33].

User Simulations. A significant volume of research on RL algorithms is focused on games. As a result, many platforms have been built for learning and evaluating RL algorithms on games, such as the Arcade Learning Environment (ALE) [1]. Brockman et al. [2] collect a large series of such environments in the widely used OpenAI Gym platform [2]. An important reason for early work to consider games is that they can be simulated at scale with relatively low computational costs. Thus, RL algorithms can obtain a large number of interactions required to find the optimal policies, making research much easier.

In contrast with games, only recently simulators for RL-based RSs have been proposed. Rohde et al. [22] introduce RecoGym, which simulates an RL environment for online advertising based on completely synthetic data. However, since it uses fully synthetic data, it is unclear how well RecoGym simulates realistic user behavior. Shi et al. [25] propose PyRecGym, which bases its simulation on logged user data, and simulates a more general recommendation task. In order to aid reproducibility and sharing of models in academia, Ie et al. [13] create Recsim: a configurable simulation platform for evaluating RL-algorithms on recommendation tasks. Recsimu [36] and VirtualTabao [26] both use a Generative Adversarial Network to tackle the challenges of complex item distributions based on e-commerce datasets. Surprisingly, none of the existing RL4Rec simulators that

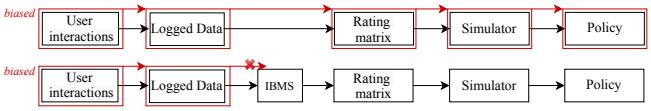


Figure 3: (Top): Bias, indicated in red, when present in logged user interaction data, affects all subsequent steps in simulators for RL4Rec. (Bottom): IBMS mitigates the effect of bias before it reaches the predicted rating matrix; if completely effective, IBMS prevents bias from affecting any further steps.

are based on logged user data, consider the effect of bias in logged data.[2] Thus we can expect that bias – known to be prevalent in interaction data – affects all existing simulators, and by extension, the policies they produce. To the best of our knowledge, we are the first to consider the effects this bias may have, and whether it can be mitigated.

5 A NOVEL METHOD FOR DEBIASING SIMULATORS

We introduce a debiasing method for RL4Rec simulators and an evaluation method to measure the effect of debiasing on the policies produced by simulators. Finally, we propose the SOFA simulator, which applies both contributions.

5.1 Debiasing a Simulator

Ie et al. [13] define the main components of an RL4Rec simulator to be a *user model*, an *item model*, and a *user-choice model*. The *user model* and *item model* aim to capture user preference for items, while the *user-choice model* simulates user feedback when an item is recommended by an RS. Generally, user preferences are modelled using a predicted user-item rating matrix. We focus on RL4Rec simulators that use predicted user-item ratings and a *user-choice model* on top of the predicted ratings (see Figure 1(b)). We consider the case where the rating prediction model is learned from logged data.

As discussed in Section 3.1, logged data suffers from interaction bias, which affects any rating prediction model learned from it. Consequently, any simulator using such a prediction model will also be biased. This poses a problem for RL4Rec, since simulated user behavior should not be affected by the way a dataset was logged. As a solution we propose the *Intermediate Bias Mitigation Step* (IBMS), an intermediate step between the logged data and the learned prediction model that aims to mitigate the bias originating from the data from affecting the model. Figure 3 displays where the Intermediate Bias Mitigation Step (IBMS) fits in the simulator pipeline: by mitigating the effect of bias before the prediction model is learned, it minimizes its effect to reach subsequent steps, including the final produced policy.

The IBMS can apply various debiasing methods; for this paper we use the IPS approach widely used in causal inferece [14] and complete-cases analysis [18]. First, we consider a standard rating prediction loss. Let $\hat{Y}$ be the predicted ratings, Y true ratings, and $o_{u,i} = 1$ indicate that a rating from user u and item i is present in the logged data. The standard loss is based on all the pairs that are present in the logged data:

$$\mathcal{L}_{Naive} = \frac{1}{|\{(u,i) : o_{u,i} = 1\}|} \sum_{(u,i):o_{u,i}=1} \delta_{u,i}(Y,\hat{Y}), \tag{7}$$

[2]Recsim [13] is an exception, as Ie et al. [13] mention bias in logged user data is a challenge but they do not propose a solution.

where $\delta_{u,i}$ is chosen to match some metric, with common choices being MSE and MAE:

$$\delta^{\text{MSE}}_{u,i}(Y,\hat{Y}) = (y_{u,i} - \hat{y}_{u,i})^2, \qquad \delta^{\text{MAE}}_{u,i}(Y,\hat{Y}) = |y_{u,i} - \hat{y}_{u,i}|. \tag{8}$$

We call this standard loss a *naive* approach, because it assumes all ratings are equally likely to be present in the logged data, *i.e.,* the data is MCAR. In contrast, interaction data on RS is usually MNAR, which leads to a biased estimate of the full-information loss (*i.e.,* the loss based on all ratings) since:

$$E[\mathcal{L}_{Naive}] = \frac{1}{\sum_{u=1}^{N}\sum_{i=1}^{M} P(o_{u,i}=1)} \sum_{u=1}^{N}\sum_{i=1}^{M} P(o_{u,i}=1)\delta_{u,i}(Y,\hat{Y}) \neq \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \delta_{u,i}(Y,\hat{Y}).$$

Due to the effect of the bias introduced by $P(o_{u,i} = 1)$, optimizing this naive loss can lead to a gross misprediction of the predicted rating matrix $\hat{Y}$ [24, 30]. To mitigate the effect of bias in MNAR feedback, Schnabel et al. [24] apply an IPS estimator [14]. If the probabilities $P(o_{u,i} = 1)$ are known, they can be corrected for by weighting the logged ratings:

$$\mathcal{L}_{IPS} = \frac{1}{N \cdot M} \sum_{(u,i):o_{u,i}=1} \frac{\delta_{u,i}(Y,\hat{Y})}{P(o_{u,i}=1)}. \tag{9}$$

Basing $\mathcal{L}_{IPS}$ on logged data provides an unbiased estimate of the full-information loss:

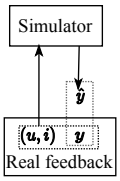
$$E[\mathcal{L}_{IPS}] = \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \frac{P(o_{u,i}=1)\delta_{u,i}(Y,\hat{Y})}{P(o_{u,i}=1)} = \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \delta_{u,i}(Y,\hat{Y}).$$

For this to be truly unbiased, the exact $P(o_{u,i} = 1)$ values have to be known. In practice, the logged data reveals which ratings were logged and which were not, thus an estimation method can be fitted on $o_{u,i} = 1$ to infer a model of $P(o_{u,i} = 1)$. Schnabel et al. [24] propose to use two simple propensity estimation methods: (i) Naive Bayes with Maximum likelihood [19], and (ii) Logistic Regression based on features of a user-item pair [23]. By IPS weighting the ratings, the IBMS can prevent bias from affecting the rating prediction model of a simulator. In the ideal case, this removes the effect of bias on the resulting policies completely. In practice, we do not expect IBMS to completely remove bias but mitigate it to a large degree. The IBMS is applicable to any simulator that simulates interactions based on a rating prediction model.

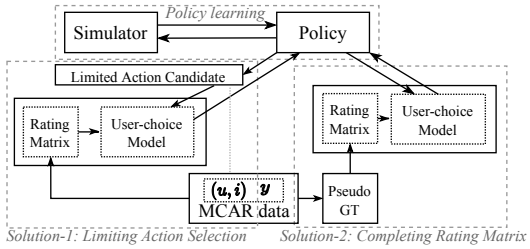
5.2 Evaluating the Effect of Bias in a Simulation

To evaluate how well the IBMS mitigates bias from affecting the resulting policies, we compare the performance of a policy trained in a simulator with and without the IBMS. Simulators are designed for situations where online deployment is impossible, thus, performance also needs to be estimated offline in these situations. Existing work has evaluated RL4Rec simulators by comparing their simulated feedback with logged user feedback [8, 26, 36], as shown in Figure 4(a). The downside of this evaluation is that it does not consider the performance of policies learned with the simulator, despite the fact that finding an optimal policy is the ultimate goal of RL4Rec.

As an alternative, we propose an offline evaluation method that does consider the final produced policies. Our evaluation method only requires a sparse set of MCAR ratings, gathered on randomly



(a) Evaluation based on observed user behavior.



(b) Evaluation with sparse MCAR data.

Figure 4: Different evaluation methods. (a) shows the evaluation of a simulator by comparing the simulated feedback (*e.g.,* ratings) with logged user feedback. (b) shows the process of evaluating a simulator in an unbiased simulator created from the MCAR data, where the problem caused by the sparsity of MCAR data is handled by two solutions: solution 1 is to evaluate on policy with limiting action selection shown on the left-hand side of (b), while solution 2 is to evaluate in the simulator with the complete rating matrix generated based on MCAR data shown on the right-hand side of (b).

selected items. Since publicly available datasets exist that meet this requirement (see Section 6) this method is available to all researchers in the field. Thus, we assume that a large number of MNAR ratings and a sparse set of MCAR ratings are available. Then our evaluation method consists of the following steps: (i) Train a policy using a simulator with the IBMS on the MNAR ratings, we will call the resulting policy the *debiased policy*. (ii) Train another policy using an identical simulator on the MNAR ratings, expect it is does not apply IBMS, resulting in the *biased policy*. (iii) Create another identical simulator, except that it is based on the MCAR ratings; call this the unbiased simulator. (iv) Finally, deploy both the biased and debiased policies in the unbiased simulator to evaluate their performance by looking at cumulative reward; the difference reveals the effect of the IBMS. The intuition behind this approach to evaluation is that because MCAR data is already debiased during logging, we can create an unbiased simulator. By comparing the behavior of two policies trained with and without the IBMS in this simulator, we can see if IBMS truly removed the effect of bias. Importantly, the actual behavior of the produced policies is evaluated; this best indicates the usefulness of a simulator. While the lack of bias in MCAR data is useful, its sparsity is still a problem, as the simulator cannot simulate feedback on items without a rating. We propose two solutions, both visualized in Figure 4(b):

Solution 1 – Limiting Action Selection: During evaluation the RS is limited to only recommend items for which ratings are available in the MCAR data. Thus for each user u the simulator finds the set of items i for which ratings $r_{u,i}$ are available in the MCAR data. The advantage of this approach is that user behavior is always based on real MCAR ratings. The disadvantage is that it limits the behavior of the RS: it could be unable to evaluate the actual behavior the RS would perform, since many items are unavailable for certain users.

Solution 2 – Completing the Rating Matrix: To avoid limiting the behavior of the RS, a pseudo Ground Truth (GT) rating matrix could be generated using a rating prediction model learned from the MCAR data. In contrast with rating matrices based on MNAR data, the resulting pseudo GT is unbiased. The advantage is that the RS is not limited in its behavior during evaluation, thus the actual behavior it would perform is evaluated. The disadvantage is that the pseudo GT is based on predicted ratings, thus it may have some differences with the true user preferences.

5.3 A Simulator for Offline Learning and Evaluation

A predicted user-item rating matrix is first loaded to initialize the simulator. To simulate a user u to interact with the RS, a simulator initializes state s_0^u as empty to simulate user login. In the t-th turn of interaction, the RS recommends an item i_t as action a_t. After receiving this item, the *user-choice* model of the simulator simulates user feedback f_t on item i_t, completes the state transition from state s_t^u to s_{t+1}^u and generates the immediate reward r_t. The RS observes feedback f_t plus the next state s_{t+1}^u, and prepares for the next turn of interaction. After K turns, the episode is terminated, and the RS saves a sequence of transitions $[(s_1,a_2,r_2,s_2),\ldots,(s_{K-1},a_K,r_K,s_K)]$ into experience buffer $\mathcal{D}$. The transitions in $\mathcal{D}$ can be subsequently used to update the parameters of the RS.

To address the functional requirements of a simulator, we design our *Simulator for OFfline leArning and evaluation* (SOFA), a debiased simulator consisting of two components: (i) a debiased user-item rating matrix to present users' preference on items, and (ii) a user-choice model to simulate user feedback, and provide the updated state and immediate reward to RS:

(i) The **debiased user-item rating matrix** is produced using the IBMS where we apply Propensity-Scored Matrix Factorization (MF-IPS) [24]. Given a user u and an item i, MF computes the predicted rating $\hat{y}_{u,i}$ as: $\hat{y}_{u,i} = \boldsymbol{p}_u{}^\top \boldsymbol{q}_i + a_u + b_i + c$, where the $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ are embedding vectors of user u and item i, and the a_u, b_i, and c are offsets for the user, item and global respectively. MF-IPS is optimized by minimizing the prediction error between the observed ratings $y_{u,i}$ and the predicted rating $\hat{y}_{u,i}$, weighted inversely to $P(o_{u,i}=1)$:

$$\underset{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{A}}{\operatorname{argmin}} \left[\sum_{(u,i):o_{u,i}=1} \frac{\delta(y_{u,i},\hat{y}_{u,i})}{P(o_{u,i}=1)} + \lambda\left(\|\boldsymbol{P}\|_F^2 + \|\boldsymbol{Q}\|_F^2\right) \right], \qquad (10)$$

where $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{A}$ denote the embeddings of all users, all items, and the offset terms, respectively. Thus the final predicted rating matrix is: $\hat{\boldsymbol{Y}} = \boldsymbol{P}^\top \boldsymbol{Q} + \boldsymbol{A}$.

(ii) The **user-choice model** simulates user feedback on the item being recommended from the RS, and provides the updated state and immediate reward to RS. Thus, the following steps are required for the user-choice model: (i) *Feedback simulation*: We define ratings higher than 3 as positive preference, and

others as negative preference following common settings in RSs. Based on the assumption that users tend to click items if they have a positive preference for these items, if $y_{u,i_t} > 3$, the user clicks item i_t, denoted as $f_t = 1$; otherwise, the user skips the item, $f_t = 0$. (ii) *State transition*: just concatenate i_t and f_t with s_{t-1} as the updated state s_t as defined in Section 2. (iii) *Reward generation*: The immediate reward r_t of *click* and *skip* feedback is specifically set to 1 and -2, these values were chosen because preliminary experiments showed they lead to efficient and stable policy learning in experiments. Finally, the user-choice model sends the updated state and the immediate reward back to the RS.

6 EXPERIMENTAL SETUP

In response to the limitation of the existing simulators we point out and the solution we propose, we analyze three research questions in the experiments: (RQ1) Does interaction bias in logged data affect a simulator? (RQ2) Can IBMS mitigate this bias effectively? (RQ3) How does the intensity of bias affect the simulators and their resulting policies? Below, we describe the datasets, and present the evaluation details for the simulator and the produced policy including the evaluation metrics and parameters used in our experiments.

Datasets. Our experiments are based on two real-world datasets and several synthetic datasets we generated ourselves, each with MNAR logged data as training set and MCAR data as test set.

Yahoo!R3 dataset [19]. The MNAR logged data of this dataset contains approximately 300,000 user-supplied ratings from 15,400 users on 1,000 items in total. The MCAR data is collected by asking 5,400 users to give ratings on 10 items randomly selected from the 1,000 items. Following [24], we consider positivity bias and use Naive Bayes to estimate propensities $P(o_{u,i})$.

Coat dataset [24]. The dataset includes ratings from 290 users on 24 self-selected items and 16 randomly-selected items from totaly 300 items. Following [24], propensity $P(o_{u,i})$ is estimated by using standard regularized logistic regression trained with the profile of users (*e.g.,* gender and age) and items (*e.g.,* type and color). The bias estimated in the above way is not specified as a certain type of bias and can be recognized as a mixture of different types of biases.

Synthetic data. In order to measure the effect of the degree of bias on simulators, we generate several synthetic datasets with varying degrees of positivity bias. Unlike real-world data, this synthetic setup allows us to keep all factors constant except for the positivity bias. The generation of synthetic data involves two steps:

(i) Generate the complete true user-item rating matrix, denoted as GT. We follow Zou et al. [42] where the generation process is based on a standard Gaussian distribution. Given N users and M items, we generate the associated parameter vectors $\boldsymbol{P} \in \mathbb{R}^{N\times d}$ and $\boldsymbol{Q} \in \mathbb{R}^{M\times d}$ as profiles of users and items. $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ denoting profile vectors of user u and item i, are both drawn from the normal distribution $\mathcal{N}(0,1)$. User preference on items is determined by the inner-product of $\boldsymbol{P}$ and $\boldsymbol{Q}$, denoted as $\boldsymbol{P}^\top\boldsymbol{Q}$. GT is generated by mapping $\boldsymbol{P}^\top\boldsymbol{Q}$ into five rating bins with score from 1 to 5 according to a certain rating distribution $P(Y = y)$. In practice, we choose $N = 300$, $M = 300$, $d = 10$, and set $P(Y = y) = [0.526, 0.242, 0.144, 0.062, 0.026]$ for $y = [1,2,3,4,5]$.

(ii) Generate MNAR logged data under the control of observation probability:

$$P(o_{u,i} \mid y_{u,i}) = \alpha P(o_{u,i} \mid y_{u,i}, \textit{pos-bias}) + (1-\alpha) P(o_{u,i} \mid \textit{uniform}). \tag{11}$$

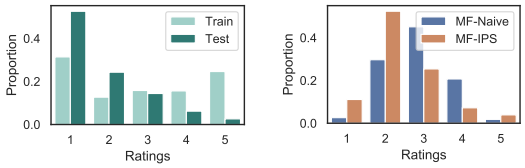
We set the probability of uniform observation $P(o_{u,i} = 1 \mid \textit{uniform}) = 5\%$ so that ∼5% of the user-item rating matrix is observed; thus, the remaining ∼95% is missing. We set $\mathbb{P}(o_{u,i} = 1 | y_{u,i} = y, \textit{pos-bias}) = [0.029, 0.021, 0.035, 0.161, 0.577]$ for $y = [1, 2, 3, 4, 5]$ to obtain a rating distribution similar to that of the Yahoo!R3 dataset. The intensity of bias is controlled by α: if $\alpha = 1.0$, the sampling probability is determined by positivity bias; if $\alpha = 0$, the logged data is sampled completely at random. We vary $\alpha \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ to generate MNAR logged data with different degrees of positivity bias.

Hyperparameters. The simulators rely on the user-item rating matrix generated by MF, including MF-IPS and MF-Naive. We followed the procedure of Schnabel et al. [24] to tune the MF hyperparameters: the L_2 regularization weight $\lambda \in \{10^{-6}, \ldots, 1\}$ and dimension of embeddings of users and items $d \in \{5, 10, 20, 40\}$, were chosen by cross-validation while considering to match the rating distributions of the predicted ratings with the real rating distributions simultaneously.[3]

For the policy used in the experiments, we use a basic DQN policy with a Gated Recurrent Unit (GRU)-based network to encode discrete state and approximate action-value function. Due to space limitations, a detailed description of the architecture of this DQN policy is provided in the released code. The required hyperparameters come in two kinds: (1) Hyperparameters of the used DQN, *e.g.,* γ discount factor, and the dimension h of the look-up layer, and the dimension h^{GRU} of the GRU hidden state. (2) Hyperparameters used in learning process, *e.g.,* the size of replay buffer $\mathcal{D}$, the speed of greedy epsilon decay, the size of minibatch and the frequency of target network update. Following Zou et al. [42], we fix the discount factor γ to 0.9, and choose the other hyperparameters by running multiple experiments and seeing which resulted in the most stable learning curves which was measured by the average cumulative number of clicks over 10-turn interactions with given simulators. The specific values of the hyperparameters for different datasets will be released with the code.

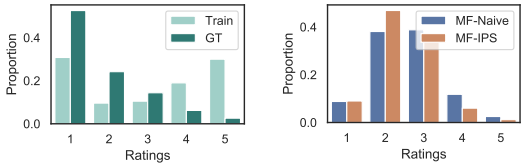
Evaluation Metrics. To evaluate the performance of a policy, we use the cumulative number of clicks received over 10 interaction turns in the *unbiased* simulator. Additionally, we apply the evaluation metrics Mean Squared Error (MSE) and Mean Absolute Error (MAE), both of which are widely used for the rating prediction task. Finally, Accuracy (ACC) and Click-ACC are also used to show the accuracy of the predicted ratings and the predicted click/skip behavior generated by the click model, which maps high ratings into *click*s and low ratings into *skip*s.

[3] This is slightly different from the setting in [24] without matching the real rating distributions. The median rating yields the minimal MSE loss when prediction error is large. This may cause most of the simulated feedback to be negative, and policies cannot learn useful information from the interactions.



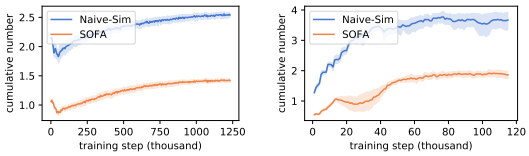
(a) Train vs Test (b) MF-IPS vs. MF-Naive

Figure 5: Rating distributions on Yahoo!R3 dataset.



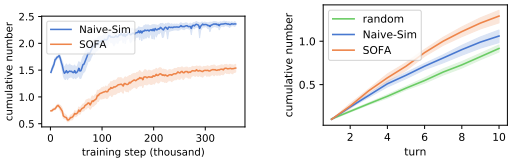
(a) Train vs. GT (b) MF-IPS vs. MF-Naive.

Figure 6: Rating distributions on the synthetic data with $\alpha = 1$.



(a) Yahoo!R3. (b) Coat.

Figure 7: Learning curves tracking average cumulative number of clicks over 10-turn interactions with given simulators.



(a) Policy learning (b) Policy evaluation

Figure 8: Policy learning and evaluation with cumulative number of clicks on the synthetic data with $\alpha = 1$.

7 EXPERIMENTAL RESULTS

7.1 Effect of Interaction Bias

Recall that in Figure 3, we illustrate the propagation of the effect of bias from user interactions to reach the produced policies. We analyze the effect of bias on the task of predicting the rating matrix. In Table 1, we find that MF-IPS outperforms MF-Naive with metrics MSE, MAE and ACC on two real datasets[4] and synthetic data with $\alpha = 1$.

Moreover, for a better understanding of how bias affects MF, we analyze the complete user-item rating matrices generated by MF-IPS and MF-Naive. For the sake of brevity, we only present the analysis of positivity bias on Yahoo!R3 and synthetic data. Figure 5(a) and 6(a) show the rating distributions of the MNAR logged data (Train) and the unbiased data (Test for MCAR data of Yahoo!R3 or GT for the synthetic data). Positivity bias in the logged data is adequately demonstrated resulting in a large "*oversampling*" of the higher ratings. Figure 5(b) and 6(b) show the rating distributions of the complete user-item rating matrices generated by MF-IPS and MF-Naive learned from the logged data. We find that MF-Naive tends to overestimate ratings, and this in turn confirms our theoretical analysis in Section 3.2. MF-IPS can mitigate this kind of bias to some extent, shown here as a larger number of lower ratings than with MF-Naive. We notice a mismatch in the rating distributions between the true rating matrices and the generated rating matrices with ratings concentrating at 2 for MF-IPS or 3 for MF-Naive. The main reason is that MF models learned from the sparse logged data still suffer from large prediction errors, and predicting ratings as the median yields the minimal loss (*e.g.,* MSE loss).

To conclude, interaction bias affects the prediction of the rating matrix based on logged data. Thus, any simulator using such a prediction model will also be biased, and the quality of policies trained using such a biased simulator will be affected.

7.2 Evaluation Results on Resulting Policies

Two DQN policies equipped with the same networks first interact with two simulators, one with IBMS named SOFA and one without IBMS named Naive-Sim, and update their parameters from the interactions. Figure 7 and 8(a) shows the learning curves of these DQN policies, which track average cumulative number of clicks over 10-turn interactions with given simulators SOFA and Naive-Sim on Yahoo!R3, Coat and synthetic dataset with $\alpha = 1$. Results are an average of 10 independent runs, lines show mean performance, and shaded areas are confidence intervals.

We notice that learning curves show a downward trend at the begining of learning, because the simulators follow the basic hypothesis for RSs that users would dislike repeated recommended items and directly skip them. The duplicate recommendation is detrimental to novelty and we should avoid it [27]. We can observe that these policies converge after multiple learning steps, and the cumulative numbers of click for the policies resulting from using Naive-Sim are consistently higher than SOFA over the whole learning process. It is noteworthy that the learning curves are based on the number of clicks received during training; they are an unreliable estimate of actual performance due to bias.

Figure 9 shows the evaluation results on the Yahoo!R3 and Coat datasets with two solutions of evaluation on the sparse MCAR data: (1) Solution-1: **Limiting Action Selection**, (2) Solution-2: **Completing the Rating Matrix**.

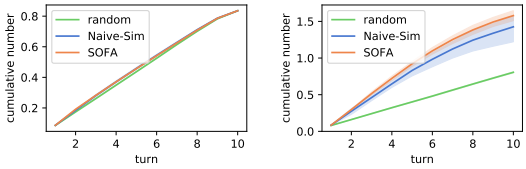
DQN policies resulting from using simulators outperform the random recommendation policy on two real datasets. For Solution-1, the gap of the cumulative number of clicks over interactions between the different policies is not significant on Yahoo!R3. The most plausible reason is that the limited action candidate set is too distinct from the items that policies would actually recommend. For Solution-2, the produced DQN policies clearly outperform random recommendation policies. In the first turn of interaction, policies show the same results because DQN-based policies randomly recommend an item in the first turn when the initial user state is empty. Then the policies recommend the item following the ϵ-greedy strategy to choose the action with highest Q-value, and obtain better performance than random recommendation policy.

DQN policies resulting from using SOFA perform better than the policies resulting from using Naive-Sim in most cases, except for the evaluation results for Solution-1 on Yahoo!R3 dataset, most likely because the limited action candidate set results in very similar rec ommendations for all the different policies. The evaluation results on the *debiased* simulator show a reversal of relative differences compared to the learning curves. This again supports our analysis

[4]The results are similar to those reported in [24], but slightly different because we consider matching the real rating distributions.

Table 1: MAE, MSE, ACC, and Click-ACC performance of MF-IPS and MF-Naive compared with unbiased testset. Click-ACC means the accuracy for the click or skip behaviors generated from the rating scores. ↓/↑ indicate smaller is better or worse.

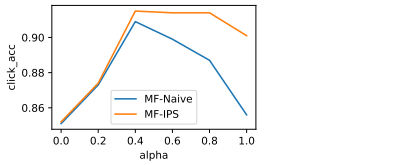
Method	Yahoo!R3				Coat				Synthetic ($\alpha = 1$)			
	MSE↓	MAE↓	ACC↑	Click-ACC↑	MSE↓	MAE↓	ACC↑	Click-ACC↑	MSE↓	MAE↓	ACC↑	Click-ACC↑
MF-IPS	1.518	0.999	0.336	0.889	1.129	0.878	0.311	0.827	1.445	0.997	0.284	0.901
MF-Naive	2.263	1.287	0.222	0.761	1.217	0.914	0.287	0.830	1.780	1.093	0.273	0.856



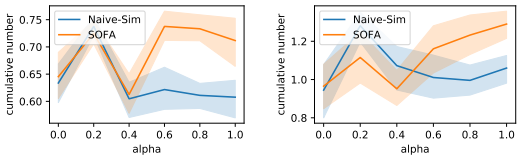
(a) Solution 1 on Yahoo!R3 (b) Solution 2 on Yahoo!R3

(c) Solution 1 on Coat (d) Solution 2 on Coat

Figure 9: Evaluation results of the produced policies on Yahoo and Coat.



(a) Simulated feedback



(b) Clicks over 5 turns (c) Clicks over 10 turns

Figure 10: Varing the intensity of bias. (a) shows the performance of the simulated feedback with Click-ACC. (b) and (c) report the evaluation results for the policies over 5- and 10-turn interactions.

on Section 3.2 that the simulator without IBMS overestimates ratings on average and simulates users to click more often because of bias. This reversal also answers **RQ1** positively: Interaction bias in logged data does affect a simulator.

We also present evaluation results on the synthetic data with observation of the logged data fully associated with positivity bias ($\alpha = 1$) by deploying the resulting policies in the unbiased simulator directly created with the complete true rating matrix GT, shown in Figure 8(b). We observe results consistent with those on the real-world datasets: DQN policies outperform the random recommendation policy, and the policies resulting from using SOFA outperform the policy resulting from using Naive-Sim. Therefore, we answer **RQ2** positively: the proposed SOFA with IBMS does mitigate enough bias to result in better performing policies.

7.3 Effect of the Intensity of Bias

To answer **RQ3**, we evaluate the simulated feedback and the resulting policies in the synthetic simulator built with the complete true rating matrix GT.

Figure 10(a) shows the performance of simulated feedback with evaluation metric Click-ACC. When α equals 0 with no bias in the logged data, the performance of simulated feedback generated from the rating matrices completed by MF-IPS and MF-Naive are similar. With the increase of the bias in logged data, MF-IPS consistently outperforms MF-Naive. Both achieve their best performance when $\alpha = 0.4$. With the increase of bias with α in range of 0.4–1.0, the performance of MF-Naive gradually decreases because the bias in logged data leads to grossly incorrect parameters estimation and rating prediction models. In contrast, MF-IPS with IBMS is more robust, which once again answers **RQ2** positively.

Figure 10(b) and 10(c) show the cumulative numbers of clicks for policies over 5 and 10-turn interactions respectively. When α is bigger than 0.5, leading to a large degree of bias in the logged data, SOFA can lead to a better policy over 5 and 10-turn interactions. When α is smaller, SOFA performs worse over 10-turn interactions, but similar over 5-turn interactions. A plausible explanation is that IPS suffers from high variance and may underestimate ratings on average due to overweighting the lower logged ratings. This underestimation of the ratings results in less positive feedback than the real case, and further affects the policy to obtain similar performance over 5-turn interactions, but worse on turn-10.

Finally, we answer **RQ3**: When the degree of bias in the logged data is very high, IBMS with using an IPS estimator can efficiently mitigate the bias from affecting the simulators and their produced policies. However, a minor flaw is that the IPS estimator used in IBMS can suffer from variance when there is very little bias in the logged data.

8 CONCLUSION

In this paper, we have analyzed the phenomenon that interaction bias in logged data affects RL4Rec simulators and the policies they produce. To mitigate the effect of bias, we have proposed the *Intermediate Bias Mitigation Step* (IBMS), an intermediate step between the logged data and the learned prediction model. Furthermore, we have introduced a novel way of evaluating the effect of bias on the final policy performance of a simulator. Experimental results have revealed that (1) interaction bias in logged data affects a simulator, (2) the proposed IBMS can mitigate the bias, especially in the case of serious bias. We have combined IBMS and the newly proposed evaluation method, in a novel Simulator for OFfline leArning and evaluation (SOFA) to help researchers in the field develop and evaluate Reinforcement Learning for Recommendation (RL4Rec) algorithms while mitigating the effects of interaction bias.

While we think that the IBMS is an important contribution to RL4Rec, SOFA only simulates the single-item recommendation scenario, where only one item is recommended at once. In practice, RSs

often recommend multiple items at once, and thus future work could consider the effect of interaction bias on simulators for multi-item recommendation scenarios. Additionally, future work could consider simulators learned from logged data with implicit feedback and user preferences modeled using ranking approaches.

ACKNOWLEDGMENTS

This research was partially supported by the Innovation Center for Artificial Intelligence (ICAI) and the Netherlands Organisation for Scientific Research (NWO) under project nr 612.001.551. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] Rocío Cañamares and Pablo Castells. 2018. Should I Follow the Crowd?: A Probabilistic Analysis of the Effectiveness of Popularity in Recommender Systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 415–424.

[4] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale Interactive Recommendation with Tree-structured Policy Gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3312–3320.

[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] Ruey-Cheng Chen, Qingyao Ai, Gaya Jayasinghe, and W Bruce Croft. 2019. Correcting for Recency Bias in Job Recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, 2185–2188.

[7] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1187–1196.

[8] 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.

[9] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. 2018. Reinforcement Learning Based Recommender System Using Biclustering Technique. *arXiv preprint arXiv:1801.05532* (2018).

[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] 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*. 198–206.

[12] José Miguel Hernández-Lobato, Neil Houlsby, and Zoubin Ghahramani. 2014. Probabilistic Matrix Factorization with Non-random Missing Data. In *International Conference on Machine Learning*. 1512–1520.

[13] 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).

[14] Guido W Imbens and Donald B Rubin. 2015. *Causal Inference in Statistics, Social, and Biomedical Sciences*. Cambridge University Press.

[15] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 781–789.

[16] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A Contextual-bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th international conference on World wide web*. 661–670.

[17] 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*. 37–46.

[18] Roderick JA Little and Donald B Rubin. 2019. *Statistical Analysis with Missing Data*. Vol. 793. John Wiley & Sons.

[19] Benjamin M Marlin and Richard S Zemel. 2009. Collaborative Prediction and Ranking with Non-random Missing Data. In *Proceedings of the third ACM conference on Recommender systems*. ACM, 5–12.

[20] Benjamin M Marlin, Richard S Zemel, Sam Roweis, and Malcolm Slaney. 2007. Collaborative Filtering and the Missing at Random Assumption. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*. 267–275.

[21] Bruno Pradel, Nicolas Usunier, and Patrick Gallinari. 2012. Ranking with Non-random Missing Ratings: Influence of Popularity and Positivity on Evaluation Metrics. In *Proceedings of the sixth ACM conference on Recommender systems*. ACM, 147–154.

[22] 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).

[23] Paul R Rosenbaum. 2002. Overt Bias in Observational Studies. In *Observational Studies*. Springer, 71–104.

[24] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *International Conference on Machine Learning*. 1670–1679.

[25] Bichen Shi, Makbule Gulcin Ozsoy, Neil Hurley, Barry Smyth, Elias Z Tragos, James Geraci, and Aonghus Lawlor. 2019. PyRecGym: A Reinforcement Learning Gym for Recommender Systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 491–495.

[26] 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.

[27] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. 2019. How Good Your Recommender System Is? A Survey on Evaluations in Recommendation. *International Journal of Machine Learning and Cybernetics* 10, 5 (2019), 813–831.

[28] Harald Steck. 2010. Training and Testing of Recommender Systems on Data Missing Not at Random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 713–722.

[29] Harald Steck. 2011. Item Popularity and Recommendation Accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 125–132.

[30] Harald Steck. 2013. Evaluation of Recommendations: Rating-prediction and Ranking. In *Proceedings of the 7th ACM conference on Recommender systems*. 213–220.

[31] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.

[32] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised Reinforcement Learning with Recurrent Neural Network for Dynamic Treatment Recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2447–2456.

[33] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. 2019. Doubly Robust Joint Learning for Recommendation on Data Missing not at Random. In *International Conference on Machine Learning*. 6638–6647.

[34] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun. 2019. Hierarchical Reinforcement Learning for Course Recommendation in MOOCs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 435–442.

[35] Shuo Zhang and Krisztian Balog. 2020. Evaluating Conversational Recommender Systems via User Simulation. *arXiv preprint arXiv:2006.08732* (2020).

[36] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward Simulating Environments in Reinforcement Learning based Recommendations. *arXiv preprint arXiv:1906.11462* (2019).

[37] 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 Conference on Recommender Systems*. 95–103.

[38] 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.

[39] 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).

[40] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly Learning to Recommend and Advertise. *arXiv preprint arXiv:2003.00097* (2020).

[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 *Proceedings of the 2018 World Wide Web Conference*. 167–176.

[42] 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. *arXiv preprint arXiv:1902.05570* (2019).