State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study

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

Harrie Oosterhuis
Radboud University
Nijmegen, The Netherlands
harrie.oosterhuis@ru.nl

Bunyamin Cetinkaya
University of Amsterdam
Amsterdam, The Netherlands
bun.cet20@gmail.com

Thijs Rood
University of Amsterdam
Amsterdam, The Netherlands
thijs.rood@hotmail.com

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

ABSTRACT

Methods for reinforcement learning for recommendation (RL4Rec) are increasingly receiving attention as they can quickly adapt to user feedback. A typical RL4Rec framework consists of (1) a state encoder to encode the state that stores the users' historical interactions, and (2) an RL method to take actions and observe rewards. Prior work compared four state encoders in an environment where user feedback is simulated based on real-world logged user data. An attention-based state encoder was found to be the optimal choice as it reached the highest performance. However, this finding is limited to the actor-critic method, four state encoders, and evaluation-simulators that do not debias logged user data. In response to these shortcomings, we reproduce and expand on the existing comparison of attention-based state encoders (1) in the publicly available debiased RL4Rec SOFA simulator with (2) a different RL method, (3) more state encoders, and (4) a different dataset. Importantly, our experimental results indicate that existing findings do *not* generalize to the debiased SOFA simulator generated from a different dataset and a Deep Q-Network (DQN)-based method when compared with more state encoders.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → Modeling and simulation.

KEYWORDS

Reinforcement Learning; Recommendation; State Encoders

ACM Reference Format:
Jin Huang, Harrie Oosterhuis, Bunyamin Cetinkaya, Thijs Rood, and Maarten de Rijke. 2022. State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study. In *Proceedings of the 45th Int'l ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3477495.3531716

SIGIR '22, July 11–15, 2022, Madrid, Spain.

ACM ISBN 978-1-4503-8732-3/22/07.
https://doi.org/10.1145/3477495.3531716

1 INTRODUCTION

With the development of interactive recommender systems (RSs), reinforcement learning for recommendation (RL4Rec) is receiving increased attention as reinforcement learning (RL) methods can quickly adapt to user feedback [2, 32]. RL4Rec has been applied in a variety of domains, such as movie [60, 62], news [68], and music recommendations [41]. A typical process flow of RL4Rec starts with an action of the system, which is an item being recommended to the user. Subsequently, user interactions with the item are returned as feedback (*e.g.,* dwell time, a rating, or a click) to the system, which then interprets the feedback as a reward signal. Finally, with this new interaction, the system updates a state representation that keeps track of the user's historical interactions with the recommended items. The cycle then repeats as the system again tries to recommend the best item to the user based on its updated state representation. The goal of RL4Rec is to optimize the system so as to achieve the maximum cumulative reward.

An RL4Rec framework typically consists of two parts: (1) the *state encoder* that encodes the state – a user's historical interactions – into a dense representation that is used to estimate the user's preference and the value of state-action pairs; and (2) an *RL method* (*e.g.,* the Deep Q-Network (DQN) [39] or the actor-critic [31] method) that is applied to generate actions based on an estimated state-action value function and observed reward. While RL4Rec methods have achieved good performance, the effect of the state encoder on RL4Rec methods has rarely been explicitly looked at. To bridge this gap, Liu et al. [35] compared four state encoders in a simulated RL4Rec environment and concluded that an attention-based state encoder leads to the best recommendation performance. Their findings revealed that the choice of state encoders is important for effective RL4Rec and, accordingly, this shows that research into state encoders could further improve the performance of RL4Rec methods. However, the analysis of Liu et al. [35] is limited to the actor-critic method and only four different state encoders. Moreover, their evaluation was based on simulated user feedback that was directly inferred from logged user data, which is typically subject to heavy selection bias, *e.g.,* popularity bias [51]. Consequently, due to a lack of any bias correction, it is very likely that the results and findings of Liu et al. [35] are also affected by the selection bias present in the data.

In response to these shortcomings, we reproduce the work by Liu et al. [35] and generalize its findings concerning state encoders

in the following directions:

(1) *Different simulated environments*: The simulated user feedback used in [35] is generated from logged user data which is inevitably subject to user selection bias, *e.g.,* popularity bias [43]. Recently, Huang et al. [22] pointed out that simulators that do not debias logged user data yield RL4Rec methods that are heavily affected by selection bias. Hence, we use SOFA [22] – the only publicly available debiased simulator [5] – to mitigate the effect of selection bias on the resulting RL4Rec methods.
(2) *Different RL method*: DQN is the most popular RL method used in RL4Rec [9, 10, 25, 33, 34, 36, 44, 53, 60, 65, 67, 68]; it is structurally simpler than the actor-critic method by only optimizing one objective. Thus, it matters to find out whether comparisons of state encoders generalize to DQN-based RL4Rec methods.
(3) *More state encoders*: Several typical neural networks – multi-layer perceptrons (MLPs), gated recurrent units (GRUs), and convolutional neural networks (CNNs) – are not considered as state encoders by Liu et al. [35]. We expand their comparison by adding these three state encoders based on widely-used typical neural networks.
(4) *Different dataset*: Besides the Yahoo! R3 dataset [38] used by Liu et al. [35], we also use the Coat shopping dataset [47] to build the debiased SOFA simulator.

We report on our efforts to reproduce the main finding in [35]:

> *The attention state encoder for RL4Rec provides significantly higher performance than the bag of items (BOI), pairwise local dependency between items (PLD) and average (Avg) state encoders.*

Moreover, we investigate whether this finding generalizes in the four directions described above. Our experimental results show that Liu et al.'s finding *is* reproducible when applying a DQN method and evaluating in the debiased SOFA simulator on the Yahoo! R3 dataset. However, we also find that it does *not* generalize to debiased simulations generated from the Coat shopping dataset [47].

Our study addresses the following research questions:

(RQ1) Does Liu et al.'s main finding generalize to the DQN-based RL4Rec methods when evaluating in the debiased SOFA simulator and compared with more state encoders, *i.e.,* with the MLP, GRU and CNN state encoders?
(RQ2) Does Liu et al.'s main finding generalize to a debiased simulation based on a different dataset?
(RQ3) Should the choice of activation function be taken into account when using the MLP-based state encoder for RL4Recs?

2 RELATED WORK

RL4Rec methods. Deep RL methods (*e.g.,* DQN, actor-critic and REINFORCE) are able to handle high-dimensional spaces and are therefore particularly suitable for RSs with large state spaces where the user state involves combinatorial user interaction behavior [2]. DQN has been the most popular choice among the RL4Rec methods [9, 10, 25, 33, 34, 36, 44, 53, 60, 65, 67, 68]. Chen et al. [9] integrate stratified sampling action replay and approximated regretted rewards with Double DQN to stabilize the RL4Rec methods in dynamic environments. Zhao et al. [65] incorporate positive and negative feedback in a RL4Rec method. Chen et al. [10] propose a cascading DQN method to obtain a combinatorial recommendation policy with large item space. Liu et al. [34] introduce a supervised signal to enable stable training of RL4Rec methods. Others use DQN in special recommendation scenarios, *e.g.,* for news [68], movies [60], education [36], projects [44], slates [25, 53], or mobile users [33]. REINFORCE and actor-critic are other two important methods adapted in RL4Rec. REINFORCE is a policy gradient method that directly updates the policy weights [55]. Liang [30] adapts REINFORCE to find a path between users and items in an external heterogenous information network. REINFORCE with importance sampling can be used to correct for biases caused by only observing feedback on items recommended by other RSs [7, 37]. Additionally, REINFORCE is commonly used in conversational RSs [15, 52] and explainable RSs [57]. Actor-critic combines REINFORCE and the value-based method [31], thus benefiting from both components; it is able to handle large action spaces in RSs [13]. Actor-critic has been used for diverse recommendation tasks [64, 66] and domains [59, 62].

In general, DQN is the most popular RL method used in RL4Recs and has a simpler structure than actor-critic. Accordingly, we use DQN and investigate whether the findings on actor-critic based RL4Rec in [35] generalize to DQN based RL4Rec.

State encoders. Neural networks are widely used in collaborative filtering (CF) based recommendation methods; popular choices are multi-layer perceptrons (MLPs) [12, 19], convolutional neural networks (CNNs) [18], recurrent neural networks (RNNs) [20, 56, 58] and attention [11, 23]. Based on logged user behavior, these methods usually use a neural network to generate a dense vector that captures user preference and can be further used to infer the users' preference over items. That makes it suitable to adapt these recommendation methods in the state encoders. Most of the above RL4Rec methods use neural networks, such as variants of RNNs [7, 65], to construct the state encoder and generate state representation, which can subsequently be used by the previously discussed RL methods. However, the effect of state encoders has rarely been explored explicitly. To the best of our knowledge, Liu et al. [35] are the first to compare the effects of different state encoders in RL4Rec methods. We continue this research direction by reproducing and generalizing Liu et al.'s comparison.

Debiasing recommendations. Bias is prevalent in interactions with RSs, such as users choosing to rate certain items more often (self selection bias) [43, 50] and RSs showing certain items to users more often (algorithmic selection bias) [16]. As a result, user preference prediction may be biased and over-specialization [1], consequently, filter bubbles [40, 42] and unfairness [6] may occur. To correct for bias, debiasing methods may be applied, such as the error-imputation-based method [50], inverse propensity scoring (IPS) [21], and the doubly robust method [28, 45]. IPS is the most popular method and widely used in debiasing recommendations [7, 8, 27, 37, 47]. Corrections of the debiased methods may lead to substantially improved prediction performance [47].

RL4Rec simulators. The usage of simulated RL4Rec environments is widespread [5, 24, 29, 46, 48, 49, 61, 63, 68] and for a good reason: RL4Rec methods learn by directly interacting with users but

the online nature of this learning process brings risks and limitations: (1) in practice, the user experience can be negatively affected during the early stages of the learning process; and (2) research and experimentation with RL4Rec systems is often infeasible since most researchers have no access to real interactions with live users. RS simulators mitigate these issues as they allow RS developers and researchers to optimize and evaluate their RL4Rec methods on simulated user behavior [5, 24, 46, 48]. Some simulators generate user behavior based on fully synthetic data (*e.g.,* generated from a Bernoulli distribution [46]). These have been critiqued for oversimplifying user behavior [5, 48]. Alternatively, to match real user behavior more closely, other simulators generate user behavior based on logged user data [24, 48, 63]. While these simulators are widely accessible, most ignore the interaction biases present in the logged user data from which they generate simulated user behavior. Recently, Huang et al. [22] have pointed out that simulators that do not debias logged user data result in RL4Rec models that are also heavily affected by the selection biases. They argue that, as a result, findings based on the outcomes of such biased simulators can be misleading because the effect of the interaction biases extend to the results underlying such findings. To mitigate the effect of bias, the SOFA environment [22] applies inverse propensity scoring (IPS) to reduce selection bias in logged user data when learning user preference and thus provides a debiased simulator. To the best of our knowledge, SOFA is the only publicly available debiased simulator. Therefore, we use SOFA to train and evaluate RL4Rec methods with different state encoders.

3 PRELIMINARIES – RL4REC

RL4Rec methods commonly model the recommendation task as a Markov decision process (MDP), where optimization is based on interactions between the RS (*i.e.,* the agent) and users (*i.e.,* the environment). The elements of an MDP for RL4Rec are:

State space $\mathcal{S}$: A state s_t^u stores the interaction history of user u at t-th turn. For clarity and brevity, we omit the superscript u when the user is clear from the context. The state s_t consists of the items recommended by the RS and the corresponding user feedback (*e.g.,* click or skip), denoted as $s_t = ([i_1, i_2, \ldots, i_t], [f_1, f_2, \ldots, f_t])$. In turn $t+1$, the RS takes an action based on the information represented in state s_t. The state s_0^u is always initialized as empty, denoted as $s_0^u = ([\,], [\,])$.

Action space $\mathcal{A}$: The action a_t is to recommend an item i_t to user u by the RS based on state s_{t-1} in turn t. Similar to the setup of Liu et al. [35], in the SOFA simulator the RS only recommends one item to the user at every turn.

Reward $\mathcal{R}$: The immediate reward $r(s_{t-1}, a_t)$ is generated according to user's feedback f_t (*e.g.,* skip or click) on a_t.

Transition probability $\mathcal{P}$: In turn $t+1$, SOFA receives an item i_{t+1} being recommended from the RS and assumes that the state s_t transitions deterministically to the next state s_{t+1} by appending item i_{t+1} and the corresponding user feedback f_{t+1}, denoted as $s_{t+1} = ([i_1, i_2, \ldots, i_{t+1}], [f_1, f_2, \ldots, f_{t+1}])$.

Discount factor γ: $\gamma \in [0, 1]$ determines the degree to which the RS cares about future rewards: if $\gamma = 0$, the RS only takes the immediate reward into account when taking an action; if $\gamma = 1$, the sum of all future rewards is considered.

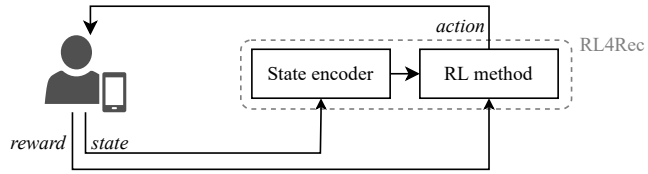


Figure 1: The general framework of RL4Rec.

Generally, the RL4Rec method includes two components as shown in Fig. 1: (1) the state encoder is applied to encode a state s into a dense representation that captures the user preference and is subsequently used to approximate the state-action value function $\widehat{Q}(s, a\,; \theta)$; for every action $a \in \mathcal{A}$, $\widehat{Q}(s, a; \theta)$ represents the expected reward following the recommendation of item a in state s; and (2) the RL method decides which action to take based on the state representation, and chooses how the parameters of the policy and state encoder models should be updated according to the rewards received from the user.

While the RL method chooses items to recommend to the user, it bases its decisions on the state representations provided by the state encoder. Therefore, the performance of an RL4Rec system heavily relies on the functioning of the state encoder. As a result, understanding how the choice of state encoder should be made is central to RL4Rec.

4 ORIGINAL STATE ENCODER COMPARISON

Liu et al. [35] follow the RL4Rec framework detailed in Section 3 and apply an actor-critic RL method to take actions and update the model parameters. The applied actor-critic method comprises two components: (1) the actor network follows the policy $\pi_{\theta^A}(s_{t-1}) \in \mathbb{R}^d$ and takes the action a_t, *i.e.,* to recommend item i with the maximum ranking score $\boldsymbol{q}_i^\top \pi_{\theta^A}(s_{t-1})$, where $\boldsymbol{q}_i$ denotes the embedding of item i; (2) the critic network estimates state-action value function $\widehat{Q}(s, a; \theta^C)$ as the approximation of the true state-action value function that represents the merits of the recommendation policy generated by the actor network. The target network technique is also adopted, where an identical actor network with policy $\pi_{\theta^{A'}}$ and an identical critic network with state-action value function $\widehat{Q}(s, a;\ \theta^{C'})$ are used. The recommendation agent makes use of experience replay and employs a replay memory $\mathcal{D}$ to store the agent's experience, *i.e.,* the user interactions with the recommended items in the RL4Rec domain. Given transitions $(s_{t-1}, a_t, r_t, s_t) \in \mathcal{D}$ generated based on the interactions between the user and recommendation policy π_{θ^A}, *i.e.,* $a_t \sim \pi_{\theta^A}(s_{t-1})$, the parameters θ^A of the actor network and θ^C of the critic network are updated as:

$$\begin{aligned}
\theta^A &\leftarrow \theta^A + \alpha^A \widehat{Q}(s_{t-1}, a_t; \theta^C) \nabla_{\theta^A} \log \pi_{\theta^A}(s_{t-1}), \\
\theta^C &\leftarrow \theta^C + \big(\alpha^C (r_t + \gamma \widehat{Q}(s_t, a_{t+1}; \theta^{C'}) \\
&\qquad - \widehat{Q}(s_{t-1}, a_t; \theta^C)) \nabla_{\theta^C} \widehat{Q}(s_{t-1}, a_t; \theta^C)\big),
\end{aligned} \tag{1}$$

where α^A and α^C denote the learning rates for the actor network and the critic network, respectively, and $a_{t+1} \sim \pi_{\theta^{A'}}(s_t)$. The target network is updated following the soft replace technique: given a soft-replace parameter τ, the parameters $\theta^{A'}$ of the actor network

and $\theta^{C'}$ of the critic network are updated as follows:

$$\theta^{A'} \leftarrow \tau\theta^{A} + (1-\tau)\theta^{A'}, \qquad \theta^{C'} \leftarrow \tau\theta^{C} + (1-\tau)\theta^{C'}. \quad (2)$$

Liu et al. [35] consider two types of state encoder methods for representing states and approximating the state-action value functions, with and without user embedding $\boldsymbol{p}_u$. First, they introduce an item-to-item collaborative filtering method, DRR-p, without taking user embeddings into account, which uses an element-wise product to capture the pairwise local dependency between items:

$$\boldsymbol{s}_t = [\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \ldots, \boldsymbol{q}_{i_t}, \{w_i\boldsymbol{q}_i \otimes w_j\boldsymbol{q}_j \mid i, j \in \{i_1, i_2, \ldots, i_t\}\}], \quad (3)$$

where $\otimes$ denotes the element-wise product; and the scalars w_i and w_j indicate the importance weights of items i and j, respectively. Additionally, three state encoders, DRR-u, DRR-ave and DRR-att, with user embeddings are introduced and outperform the state encoder DRR-p without user embeddings: (1) the element-wise product on user-item embedding pairs is incorporated: $\boldsymbol{s}_t = [\{\boldsymbol{p}_u \otimes w_i\boldsymbol{q}_i \mid i \in \{i_1, i_2, \ldots, i_t\}\}]$; and (2) to reduce computational costs, the weighted average pooling schema is used to aggregate the item embeddings: $\boldsymbol{s}_t = [\boldsymbol{p}_u, \boldsymbol{p}_u \otimes \{\text{ave}(w_i\boldsymbol{q}_i) \mid i \in \{i_1, i_2, \ldots, i_t\}\}]$, where $\text{ave}(\cdot)$ denotes the average pooling operator. Finally, (3) an attention network is applied:

$$\boldsymbol{s}_t = [\boldsymbol{p}_u, \boldsymbol{p}_u \otimes \{\text{ave}(a_{u,i}\boldsymbol{q}_i) \mid i \in \{i_1, i_2, \ldots, i_t\}\}], \quad (4)$$

$$a_{u,i} = \frac{\exp(a'_{u,i})}{\sum_{i' \in \{i_1, i_2, \ldots, i_t\}} \exp(a'_{u,i'})}, \quad (5)$$

$$a'_{u,i} = \text{ReLU}(([\boldsymbol{p}_u, \boldsymbol{q}_i]\boldsymbol{W}_2) + b_2)\boldsymbol{W}_1 + b_1, \quad (6)$$

where the weight matrices $\boldsymbol{W}_1, \boldsymbol{W}_2$ and the bias vectors b_1, b_2 project the input into a hidden layer; ReLU is the activation function for the hidden layer.

Given the state representation $\boldsymbol{s}_t$, the ranking score of item i, *i.e.,* $\boldsymbol{p}_i^\top \pi_{\theta^A}(s_t)$, can be used to execute policy $\pi_{\theta^A}(s_t)$ and approximate state-action value function $\widehat{Q}(s_t, a; \theta^C)$. Consequently, the resulting actor-critic based RL4Rec method can interact with the (simulated) users and update the parameters iteratively. Liu et al. [35] compare the actor-critic based RL4Rec method with four state encoders, DRR-p, DRR-u, DRR-ave and DRR-att, in simulators generated from two datasets [17] containing temporal information and two datasets not containing temporal information [14, 38]. They conclude that: (1) state encoders that utilize user embeddings outperform state encoders without user embeddings; (2) the average pooling schema can decrease the dimensionality of the state representation to reduce overfitting and improve recommendation performance; and (3) the attention-based state encoder provides the best performance among the four state encoders introduced above.

5 OUR REPRODUCED STATE ENCODER COMPARISON

Having specified the setting of Liu et al. [35]'s study (cf. Section 4), we generalize Liu et al.'s finding to four directions and can summarize the following key differences: (1) **Different simulated environments**: We adopt SOFA, the debiased simulator, which mitigates the effect of bias present in logged data when generating user preferences on items; in contrast, Liu et al. [35] use a simulation directly generated from logged data without considering bias. (2) **Different RL method**: We apply DQN, which is widely used in RL4Rec research and has a simpler structure with optimizing only one objective, whereas Liu et al. [35] apply the actor-critic method. (3) **More state encoders**: Besides the four state encoders proposed by Liu et al. [35], we expand the comparison by adding three more state encoders based on typical neural network architectures: MLP, GRU and CNN, which are widely used in recommendation methods to generate representations according to historical user interactions. (4) **Different dataset**: Our comparison uses the Yahoo! R3, which is also used by Liu et al. [35], as well as the Coat shopping dataset [47], which is not considered in [35]. To the best of our knowledge, these two datasets are the only publicly available datasets that can be used to unbiasedly simulate recommendations, since part of their data is gathered on randomized recommendations. Unfortunately, there are two more datasets used by Liu et al. [35] that cannot be used in SOFA due to a lack of randomized data for debiasing.

It is crucial to understand whether the choice of state encoder is important, and if so, what factors should be considered when making this choice. In particular, the aim of our reproducibility study is to analyze whether the choice of state encoder is robust w.r.t. the effect of bias, the choice of RL method, and the sources of data used. The differences listed above allow us to address this aim and investigate whether the findings of Liu et al. [35] generalize along these dimensions.

Below, we describe the setting in which we reproduce and expand on the comparisons performed by Liu et al. [35]. Section 5.1 details the debiased SOFA simulator that we use, Section 5.2 explains the DQN RL method that is applied, and finally, Section 5.3 lists the state encoders included in our comparison.

5.1 Simulator for OFfline leArning and evaluation (SOFA)

To mitigate the effect of bias present in logged data, Huang et al. [22] propose a debiased simulator, named SOFA, which consists of two components: (1) a debiased user-item rating matrix to present user preferences for items, and (2) a user choice model to simulate user feedback and generate the next state and the immediate reward. The bias mitigation step is applied between the logged data and the learned user preference prediction model, thereby mitigating the bias originating from the logged data from affecting the user preference prediction model. User behavior (*e.g.,* ratings) could be affected by various forms of selection bias, *e.g.,* users tend to rate more popular items (*popularity bias*) [43, 51] or the items that they expect to enjoy beforehand (*positivity bias*) [43]. This is generally modelled by decomposing the probability of observing a rating $y_{u,i}$ given by user u on item i into (1) the preference $P(y_{u,i})$, *i.e.,* the distribution over rating values the user u would give to item i; and (2) the propensity $P(o_{u,i})$, *i.e.,* the probability of observing any rating from user u for item i in the dataset. The assumed model is thus:

$$P(o_{u,i}, y_{u,i}) = P(o_{u,i})P(y_{u,i}), \quad (7)$$

where $o_{u,i}$ denotes the observation indicator: $o_{u,i} = 1$ if the rating $y_{u,i}$ is observed, otherwise, $o_{u,i} = 0$ indicates a rating is missing. Due to bias, certain ratings are more likely to be observed than others. In other words, $P(o_{u,i})$ is not uniform over all user-item pairs. As a result, naively ignoring the propensities during evaluation or optimization gives more weight to the user-item pairs that are

overrepresented due to bias [47], *e.g.*, giving the most weight to the most popular items. In turn, this results in biased user rating predictions $\hat{y}_{u,i}$ that fail to match the true user ratings $y_{u,i}$. The bias mitigation step of SOFA applies inverse propensity scoring (IPS) [26] to inversely weight ratings according to the corresponding observation probabilities so that, in expectation, each user-item pair is represented equally. Let $\delta(y_{u,i}, \hat{y}_{u,i})$ indicate the loss resulting from the match between true rating $y_{u,i}$ and the predicted rating $\hat{y}_{u,i}$ [47]:

$$\mathbb{E}[\mathcal{L}_{\text{IPS}}] \propto \mathbb{E}\left[\frac{\delta(y_{u,i}, \hat{y}_{u,i})}{P(o_{u,i}=1)}\right] = \frac{\mathbb{E}[o_{u,i}]\delta(y_{u,i}, \hat{y}_{u,i})}{P(o_{u,i}=1)} = \delta(y_{u,i}, \hat{y}_{u,i}). \quad (8)$$

Therefore, using the IPS debiasing method, SOFA can learn debiased user preferences for items and mitigate the effect of bias on the resulting simulated user behavior and the final produced RL4Rec methods [22].

Before the interaction starts, SOFA uniformly randomly samples a batch of users and initializes their states as empty; then SOFA interacts with RL4Rec methods over ten turns. Within SOFA, the RL4Rec methods aim to maximize the cumulative number of clicks received over ten interaction turns, and are accordingly evaluated on the cumulative number of clicks they receive over ten interaction turns. Furthermore, SOFA provides a general DQN-based RL4Rec framework, which Section 5.2 describes in detail.

5.2 Deep Q-Network based recommendation

Deep Q-Networks (DQNs) [39] are based on Q-learning, one typical value-based RL method [54], while the actor-critic methods integrate a value-based method with the policy gradient REINFORCE method [55]. As a result, DQNs have a simpler structure than actor-critic methods by only optimizing one objective; thus, while actor-critic methods are potentially more powerful for handling large state and action spaces, DQNs can be more data-efficient. DQNs have been widely used in RL4Rec to improve recommendation performance [9, 10, 25, 33, 34, 36, 44, 53, 60, 65, 67, 68]. For these reasons, we follow SOFA and choose to use the basic DQN for our reproducibility study. We optimize the DQN by fitting its predicted state-action function $\widehat{Q}(s, a; \theta)$ to the expected discounted cumulative reward $\sum_t \gamma^t r_t$. To stabilize the training process, DQN introduces a behavior network separate from the target network. Here, we apply a state encoder as the behavior network and an identical state encoder as the target network. These two state encoders have the same structure and use the same item embeddings, but are updated in different ways. Moreover, DQN makes use of experience replay and employs a replay memory $\mathcal{D}$ to store the agent's experience, *i.e.,* the user interactions with the recommended items in the RL4Rec domain.

Given a transition $(s_{t-1}, a_t, r_t, s_t) \in \mathcal{D}$, the behavior network estimates Q-value function $\widehat{Q}(s_{t-1}, a_t; \theta)$ on the given state-action pair (s_{t-1}, a_t), where θ denotes the parameters of the behavior network; the target network is used to estimate Q-value function $\widehat{Q}'(s_t, a; \theta')$ for any action $a \in \mathcal{A}$ given state s_t, with the parameters θ' fixed and periodically copied from θ in the behavior network. Following [39], the parameters θ of the behavior network are updated by minimizing the following smooth L1 loss function for steady gradients with the Adam optimizer:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_{t-1}, a_t, r_t, s_t)\sim D} \begin{cases} 0.5(\delta^{\text{TD}})^2 & \text{if } |\delta^{\text{TD}}| < 1, \\ |\delta^{\text{TD}}| & \text{otherwise.} \end{cases} \quad (9)$$

$$\delta^{\text{TD}} = r_t + \gamma \max_a \widehat{Q}'(s_t, a; \theta') - \widehat{Q}(s_{t-1}, a_t; \theta). \quad (10)$$

Note that the parameters θ' of the target network are not updated in each learning step, but periodically replaced by θ after multiple learning steps.

5.3 State encoders in our comparison

As described in Section 3, the state encoder is used to generate representations of the state that can be used as input to the approximated state-action value function. The choice of state encoder can have a large impact on the performance of the RL4Rec system [35]. Accordingly, it is crucial to select an appropriate and effective state encoder. Since Liu et al. [35] have not made their source code publicly available, we have reimplemented the four state encoders of their original comparison (see Section 4): DRR-p, DRR-u, DRR-ave and DRR-att. Due to the increasing importance of privacy and the fact that SOFA does not provide user information, we drop the user embedding w.r.t. the user id and add user feedback to the recommended items to obtain user preferences in these four state encoders, renamed as pairwise local dependency between items (PLD), bag of items (BOI), average (Avg) and Attention. Additionally, we consider three more typical neural networks – MLP, GRU and CNN – when constructing the state encoders.

We use $\boldsymbol{q}_i$ to denote the embedding of item i and $\boldsymbol{f}_i$ for the embedding of feedback $f_i \in \{0, 1\}$ from the user on the item i. Given state $s_t = ([i_1, i_2, \ldots, i_t], [f_1, f_2, \ldots, f_t])$, we have the corresponding item embeddings $[\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \ldots, \boldsymbol{q}_{i_t}]$ and feedback embeddings $[\boldsymbol{f}_{i_1}, \boldsymbol{f}_{i_2}, \ldots, \boldsymbol{f}_{i_t}]$. The state-action value function $\widehat{Q}(s_t, a)$ can be approximated by the following state encoders:

BOI: Corresponding to DRR-u from Liu et al. [35], the state representation $\boldsymbol{s}_t^{\text{BOI}}$ is formulated as a list of weighted element-wise products of historical item embeddings and the corresponding feedback embeddings. Then, one linear layer is applied and the dimensionality of the output space is set to the number of items:

$$\begin{aligned} \boldsymbol{s}_t^{\text{BOI}} &= [\{w_i \boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1, i_2, \ldots i_t\}\}], \\ \widehat{Q}(s_t, a) &= \boldsymbol{W}^\top \boldsymbol{s}_t^{\text{BOI}} + b. \end{aligned} \quad (11)$$

PLD: Corresponding to DRR-p from Liu et al. [35], the pairwise local dependency between items $e_{i,j}$ is also considered in modeling state representation $\boldsymbol{s}_t^{\text{PLD}}$:

$$\begin{aligned} \boldsymbol{s}_t^{\text{PLD}} &= [\{w_i \boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1, i_2, \ldots i_t\}\}, \\ &\qquad \{e_{i,j} \mid i, j \in \{i_1, i_2, \ldots, i_t\}\}], \\ e_{i,j} &= w_i(\boldsymbol{q}_i \otimes \boldsymbol{f}_i)^\top (\boldsymbol{q}_j \otimes \boldsymbol{f}_j) w_j, \\ \widehat{Q}(s_t, a) &= \boldsymbol{W}^\top \boldsymbol{s}_t^{\text{PLD}} + b. \end{aligned} \quad (12)$$

Avg: Corresponding to DRR-ave from Liu et al. [35], one linear layer is applied with no activation function and the dimensionality of the output space is set to the number of items:

$$\widehat{Q}(s_t, a) = \boldsymbol{W}^\top \text{ave}(\{\boldsymbol{q}_i \otimes \boldsymbol{f}_i | i \in \{i_1, i_2, \ldots, i_t\}\}) + b, \quad (13)$$

where $\text{ave}(\cdot)$ denotes the component-wise average operator on

a set of vectors; and $\boldsymbol{W}$ and b are the weight and bias term of the linear layer, respectively.

MLP: Novel in our comparison, on top of Avg, we lift the linear assumption of state and state-action value function by applying a non-linear activation function σ, *e.g.,* tanh, ReLU, or sigmoid:

$$\widehat{Q}(s_t, a) = \sigma(\boldsymbol{W}^\top \text{ave}(\{\boldsymbol{q}_i \otimes \boldsymbol{f}_i | i \in \{i_1, i_2, \ldots, i_t\}\}) + b). \quad (14)$$

CNN: Novel in our comparison, a basic CNN with one convolution layer and one max-pooling layer is applied; to compute $\widehat{Q}(s_t, a)$, a fully-connected layer is also adopted with the dimensionality being the number of items:

$$\widehat{Q}(s_t, a) = \boldsymbol{W}^\top \max(W_C(([\boldsymbol{q}_{i_1}, \ldots, \boldsymbol{q}_{i_t}, \boldsymbol{f}_{i_1}, \ldots, \boldsymbol{f}_{i_t}]^\top))) + b, \quad (15)$$

where $\max(\cdot)$ denotes the max operator of the max-pooling layer; W_C indicates the weight function of a l-dilated convolution filter of size 3×3 and the activation function ReLU; and $\boldsymbol{W}$ and b are the weight and bias term of the fully-connected layer, respectively.

GRU: Novel in our comparison, a basic GRU layer and a dense layer are applied:

$$\begin{aligned} \boldsymbol{h}_k &= W_G(\boldsymbol{h}_{k-1}, \boldsymbol{q}_{i_k} \otimes \boldsymbol{f}_{i_k}), \quad \forall k = 1, 2, \ldots, t \\ \widehat{Q}(s_t, a) &= \boldsymbol{W}^\top \boldsymbol{h}_t + b, \end{aligned} \quad (16)$$

where W_G indicates the weight function of the GRU unit with the activation funtion tanh; and $\boldsymbol{h}_0$ is set as a zero-vector. The hidden state vector $\boldsymbol{h}_k$ is computed conditioned on the previous hidden state vector $\boldsymbol{h}_{k-1}$ and the input $\boldsymbol{q}_{i_k} \otimes \boldsymbol{f}_{i_k}$.

Attention: Corresponding to DRR-att from Liu et al. [35], following [3] we insert an attention layer into the GRU-based state encoder:

$$a_k = \frac{\exp(a'_k)}{\sum_{k'=1}^{t} \exp(a'_{k'})}, \quad a'_k = (\boldsymbol{W}_A^\top \boldsymbol{h}_t)^\top \boldsymbol{h}_k, \quad (17)$$

$$\widehat{Q}(s_t, a) = \boldsymbol{W}^\top \left[\left(\sum_{k=1}^{t} a_k \boldsymbol{h}_k \right), h_t \right] + b, \quad (18)$$

where W_A denotes the weight function of the attention layer; a_k denotes the attention weight on the hidden state vector $\boldsymbol{h}_t$; and the attentive combination of all the hidden state vectors is used to compute the state-action value function $\widehat{Q}(s_t, a)$.

6 EXPERIMENTAL SETUP

In this section, we describe the experiments performed to answer the research questions presented in Section 1.

Datasets and simulators. We use SOFA to generate two debiased simulations that simulate user behavior based on two real-world datasets: Yahoo! R3 [38] and Coat shopping [47], which – to the best of our knowledge – are the only publicly available datasets that include a uniformly-random sampled test-set that allows for unbiased evaluation. The number of users in the Yahoo! R3 and Coat shopping datasets are 15,400 and 290, respectively; and the number of items are 1,000 and 300, respectively. Both datasets include a biased training set and an unbiased test set: the training set contains ratings observed from *natural* real-world user behavior, whereas the test set contains ratings asked from users on uniformly randomly sampled items. Consequently, the training set is affected by the forms of bias present in standard user interactions, but the test set is unaffected by any selection bias since it relies on uniform random sampling. The simulations used for training RL4Rec methods are based on debiased user preferences generated from IPS-based rating prediction methods (Eq. 8) on the biased training set; in contrast, the evaluation of the RL4Rec methods is performed on the unbiased simulations generated from the unbiased test sets.

Hyperparameters. The required hyperparameters come in two kinds: (1) Hyperparameters of the used DQN: we follow the hyperparameters reported by Huang et al. [22] (see Table 1) and fix the values for the DQN based RL4Rec methods with different state encoders. (2) Hyperparameters used in the state encoders: the common hyperparameters are tuned per state encoder in the following ranges: learning rate $\eta \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ and the dimension of item embedding $d \in \{16, 32, 64\}$. Additionally, the dimensions of the weight functions in the CNN, GRU and attention state encoders are taken from $d' \in \{16, 32, 64\}$.

Evaluation metrics. As introduced in Section 5.1 we use the cumulative number of clicks received over 10 interaction turns in the unbiased simulated online environments to evaluate the performance of the state encoders in the Deep Q-Network based recommendation (DQN4Rec) method. The cumulative or average number of clicks is a common choice of metric [35, 65] for online evaluation of RL4Rec since it can indicate the long-term user engagement performance achieved by RL4Recs.

Release of implementation. The complete implementation of our experiments with accompanying documentation and additional resources are publicly available for future reproducibility at https://github.com/BetsyHJ/RL4Rec.

7 EXPERIMENTAL RESULTS AND ANALYSIS

Our experiments results are meant to determine whether the main finding of Liu et al. [35] can be reproduced:

> *The attention state encoder for RL4Rec provides significantly higher performance than the BOI, PLD and Avg state encoders.*

Moreover, in our analysis we investigate whether this finding generalizes in the four directions described at the start of Section 5.

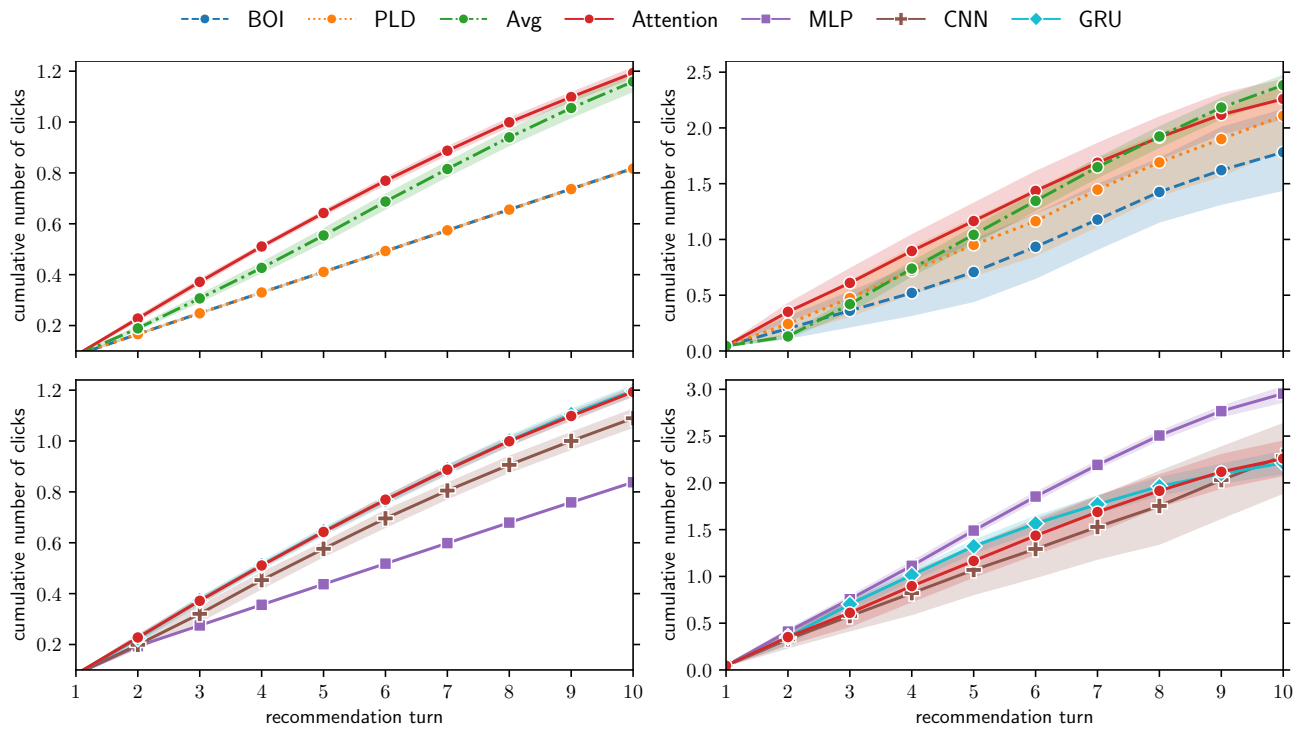
7.1 Comparison of state encoders on debiased simulation of Yahoo! R3 dataset

We start our analysis by considering our first research question (RQ1): *whether Liu et al. [35]'s finding generalizes to DQN-based RL4Rec methods when evaluated in the debiased SOFA simulator and compared with more state encoders.*

Fig. 2(a) (top) displays the evaluation performance of the optimized policies based on four state encoders proposed by Liu et al. [35]; the reported metric is the average cumulative number of clicks received over 10 interaction turns. The first interaction turn is always represented by the empty state, and as a result, the choice of state encoder is inconsequential and the performance of all state encoders is identical. As the number of interaction turns becomes larger, the differences between the state encoders become more apparent. On the simulations of Yahoo! R3 dataset – the same dataset used by Liu et al. –, as shown in Fig. 2(a) (top), we see results consistent with those reported by Liu et al.: (1) BOI and PLD perform comparably and worse than Avg; and (2) on average the attention

Table 1: List of hyperparameters for DQN and their values.

Hyperparameter	Definition	Value
Memory Size	The number of transitions stored in the replay memory.	6,000
Discount factor	Discount factor γ used in the DQN.	0.9
Epsilon	The minimal probability of recommending an item randomly when taking an action.	0.1
Epsilon decay frequency	The number of step with which the epsilon ϵ (initial value as 0.8) minus 0.1.	20,000
Minibatch size	The number of training cases randomly selected from replay memory and being used to update the parameters of policy.	128
Targetnet replacement frequency	The number of step with which the target network is updated.	20



(a) Yahoo! R3.

(b) Coat shopping.

Figure 2: Comparisons of evaluation performance (the cumulative number of clicks) among policies with four state encoders proposed by Liu et al. [35] (top), and between attention and the additional MLP,[1] GRU, and CNN-based state encoders (bottom) on the unbiased simulations generated from the unbiased test sets of Yahoo! R3 and Coat shopping datasets, respectively.

state encoder outperforms BOI, PLD, and Avg.

Fig. 2(a) (bottom) displays the evaluation performance of the attention, MLP,[1] CNN and GRU state encoders on the Yahoo! R3 dataset. We see that on average the attention state encoder performs similarly to the GRU state encoder over ten interaction turns and better than the CNN state encoder. Thus, we confirm that attention is the optimal choice in our experimental setting on the Yahoo! R3, the same dataset as used in the original comparison [35]. The main difference between MLP and GRU is the recurrent nature of the latter, thus it is the likely reason for why GRU outperforms MLP. Similarly, the higher performance of attention over GRU must be because of the additional attention layer, as this is the sole difference between the two state encoders.

We answer (RQ1) in the affirmative: Liu et al.'s finding regarding the superiority of using the attention state encoder generalizes to DQN-based RL4Rec methods when evaluating in the debiased SOFA simulation based on the Yahoo! R3 dataset used by Liu et al., and compared with three more state encoders, MLP, GRU, and CNN.

7.2 Comparison on a different dataset

Now that we have found Liu et al.'s finding to be reproducible in a debiased simulation generated from the Yahoo! R3 dataset, we consider the second research question (RQ2): *whether it also generalizes to a debiased simulation based on a different dataset.*

Fig. 2(b) displays the performance of different state encoders

[1] For the MLP-based state encoder, we use a ReLU for Yahoo! R3 and tanh for the Coat shopping dataset, which we found to be the optimal choices for the corresponding datasets.

Table 2: Training time in seconds for 1,000 training steps.

Dataset	BOI	PLD	Avg	MLP	CNN	GRU	Att
Coat	6.6	7.0	6.4	6.4	7.6	23.4	25.4
Yahoo! R3	9.0	9.6	8.4	9.4	10.8	26.8	30.4

on the debiased simulation based on the Coat shopping dataset, which was not part of the original comparison [35]. We make two observations from the top part of Fig. 2(b): (1) on average, PLD performs better than BOI, but worse than Avg; (2) attention has worse performance than Avg over 10 interaction turns. Furthermore, in the bottom part of Fig. 2(b) we see that: (3) attention does not have better performance than the additional MLP, CNN and GRU state encoders; (4) on average, attention performs comparably with GRU and CNN, although CNN does suffer from a much higher variance; (5) the MLP state encoder outperforms other state encoders significantly. Thus, in stark contrast with our results on the Yahoo! R3 dataset, on the Coat shopping dataset we do not observe the attention state encoder to have the highest performance.

Two potential reasons for this observed inconsistency between the two datasets could be (1) the difference in size between the two datasets: in contrast to attention, the Avg and MLP methods with fewer parameters are possibly more effective on the smaller Coat shopping dataset; and (2) the different recommendation scenarios: there could be a stronger dependency between items in user interactions in an online shopping scenario (Coat shopping) than in a music recommendation scenario (Yahoo! R3).

Therefore, we answer (RQ2) negatively: Liu et al.'s finding does *not* generalize to the debiased simulation with a different dataset. In particular, attention is *not* the optimal choice of state encoder for RL4Recs when evaluating in the Coat shopping dataset, which was not considered by Liu et al. [35].

In addition, we also did not observe a consistent performance for the additional MLP, CNN and GRU state encoders across the two datasets. On the Yahoo! R3 dataset, GRU performs best (out of the three) and MLP performs worst; yet on the Coat shopping dataset GRU performs similarly to CNN but considerably worse than MLP. This observation suggests that the relative effectiveness of state encoders depends on the dataset to which they are applied. Importantly, there is no single optimal state encoder applicable to RL4Recs for all datasets.

7.3 Convergency of RL4Recs state encoders

Convergence is also a crucial property for RL4Rec methods because they are more prone to divergence problems as they continuously update recommendation policies while interacting with users. Next, we investigate how the choice of state encoder affects the convergence of DQN, in terms of the number of training steps and training time needed to converge.

Fig. 3 displays the learning curves of policies with different state encoders, which track the average cumulative number of clicks over 10 interaction turns on the debiased simulations on the Yahoo! R3 and the Coat shopping datasets. We observe that: (1) BOI and PLD converge the earliest but to policies that receive only a small cumulative number of clicks; (2) MLP has a similar convergence speed as BOI and PLD but its performance at convergence greatly varies between the datasets; (3) MLP converges faster than Avg, suggesting that its activation function speeds up the learning process; (4) attention converges slightly slower than GRU, most likely due to having more parameters; and (5) the convergence speed of CNN greatly varies between the two different datasets. In summary, state encoders with few parameters, *e.g.,* Avg and MLP, converge faster than those with more parameters, *e.g.,* attention.

Furthermore, Table 2 clearly shows that the time for training on the larger Yahoo! R3 dataset is longer than on the Coat shopping dataset, which contains fewer items and users. As expected, Avg and MLP have the fewest parameters and accordingly also require less training time per thousand training steps. BOI and PLD take slightly more time than Avg which could be explained by the higher dimensionality of their state representations. Lastly, attention is more time consuming than GRU, which is likely due to its additional attention layer. In summary, the attention state encoders require a higher computation cost, despite the fact that they do not always guarantee to reach the highest performance, *e.g.,* on the Coat shopping dataset.

7.4 Choice of activation functions for MLP

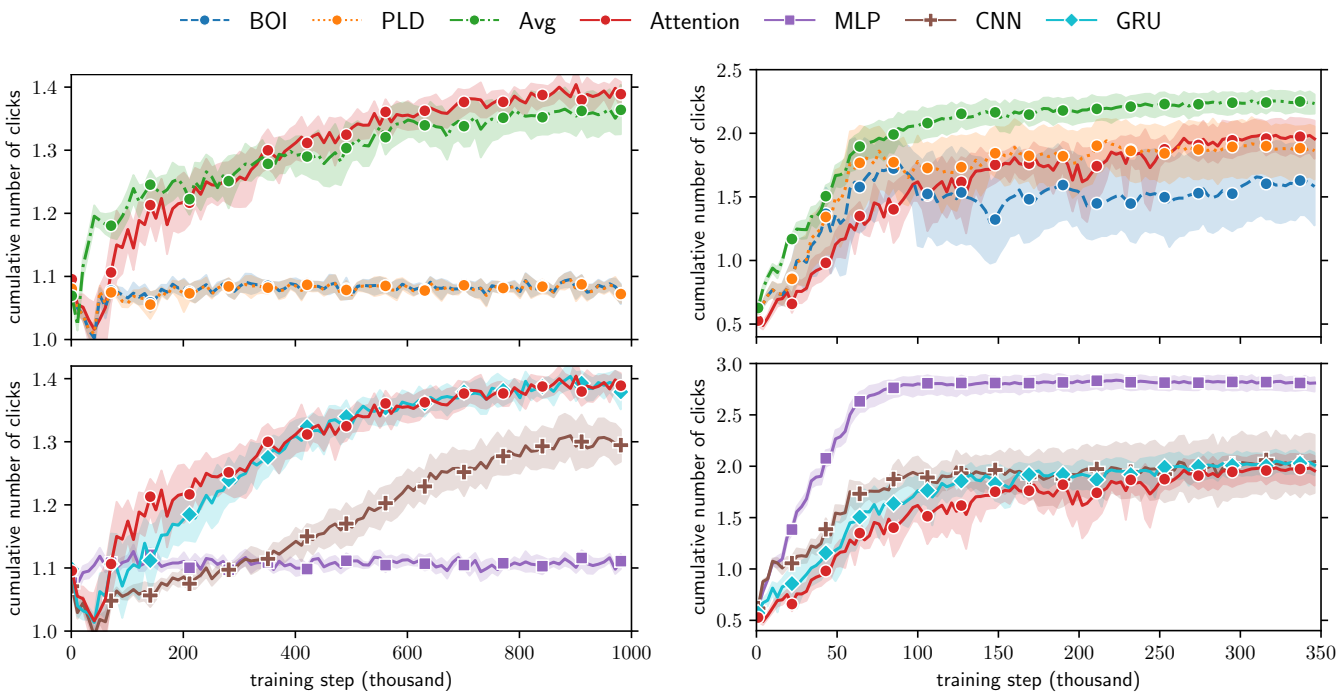
The MLP state encoders apply a non-linear activation function on top of Avg and show varying evaluation performance when applied to different datasets: we have seen that it performs best on the Coat shopping dataset, but worse than Avg on Yahoo! R3, as shown in Fig. 2 (bottom row). These observations prompt us to consider (RQ3): *whether the choice of activation functions should be taken into account when using the MLP-based state encoder for RL4Recs.*

Fig. 4 displays the comparison of evaluation performance between Avg and MLPs with the tanh, ReLU, and sigmoid activation functions. We observe that: (1) interestingly, MLPs perform better on the Yahoo! R3 dataset but worse than Avg on the Coat shopping dataset; we speculate that this is due to the different sizes of the two datasets and the different recommendation scenarios they represent; (2) for MLPs, sigmoid is the worst choice of activation function for simulations on both datasets, probably because it is more prone to the vanishing gradient problem [4]; and (3) the performance with tanh and ReLU is not consistent across both datasets: tanh has the best performance on Coat, but is worse than ReLU on Yahoo! R3.

Therefore, we answer (RQ3) in the affirmative: the choice of activation function should certainly be taken into account when using MLP-based state encoders for RL4Rec. Furthermore, our observations also suggest that the choice of activation function greatly depends on the dataset to which the MLP state encoder will be applied.

8 CONCLUSION

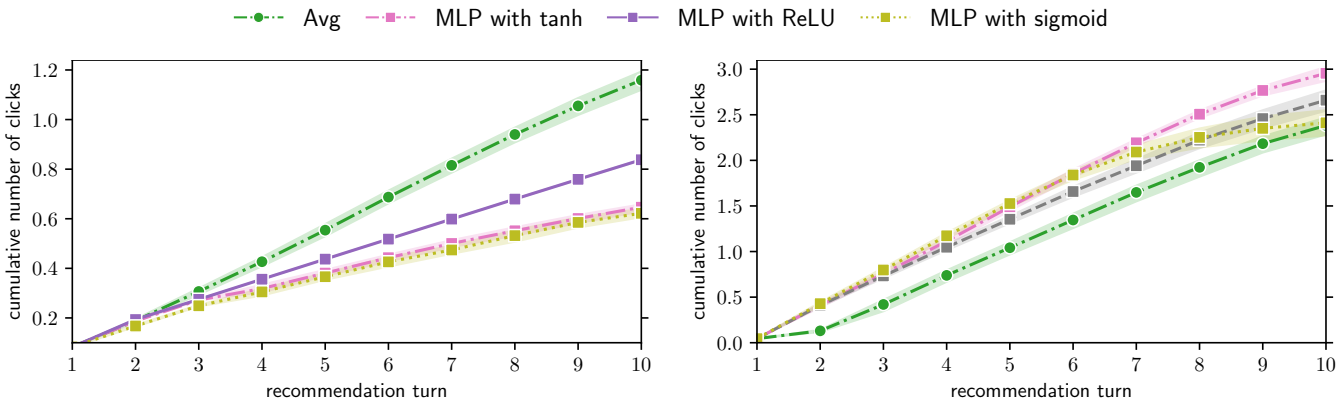
In this paper, we have reproduced and generalized a previous study by Liu et al. [35] regarding the choice of state encoder for reinforcement learning for recommendations (RL4Recs) in four directions: (1) a debiased simulated environment, named SOFA; (2) RL4Rec methods based on Deep Q-Network (DQN), the most popular RL method used in RL4Recs; (3) three additional state encoders based on three typical neural networks: multi-layer perceptrons (MLPs), gated recurrent units (GRUs), and convolutional



(a) Yahoo! R3.

(b) Coat shopping.

Figure 3: Learning curves tracking average cumulative number of clicks received by policies with four state encoders (top), and with attention and additional MLP, GRU and CNN-based state encoders (bottom) on the debiased simulations generated from training sets of Yahoo!R3 and Coat shopping datasets, respectively.



(a) Yahoo! R3.

(b) Coat shopping.

Figure 4: Evaluation performance (the cumulative number of clicks) of policies with Avg, and MLP state encoders with different activation functions (tanh, ReLU, and sigmoid) on the simulations of Yahoo! R3 and Coat shopping datasets, respectively.

neural networks (CNNs); (4) besides the Yahoo! R3 dataset used in the original study [35], we also considered the Coat shopping dataset as the basis for debiased simulations. Our experimental results show that the higher performance of the attention state encoder over the bag of items (BOI), pairwise local dependency between items (PLD), and average (Avg) state encoders is reproducible in the debiased simulation generated from the Yahoo! R3 dataset, where DQN was used instead of actor-critic RL; moreover, the attention state encoder also outperforms the three additional multi-layer perceptron (MLP), convolutional neural network (CNN) and gated recurrent unit (GRU) state encoders on the debiased simulation based on the Yahoo! R3 dataset. However, the attention state encoder performed worse than Avg and MLP when the simulation is based on the Coat shopping dataset, a dataset not used in [35], despite the fact that it has the highest computational costs.

In summary, our results confirm that Liu et al.'s finding gen-

eralizes in the first three directions, *i.e.,* the debiased simulation, DQN-based RL4Rec method, and more state encoders, but does *not* generalize to the debiased simulation generated from a different dataset, *i.e.,* the Coat shopping dataset. In addition, we have found that the choice of activation function plays a crucial role when constructing a state encoder for RL4Recs.

Future work should further investigate the importance of the choice of RL methods for RL4Rec. A comparison of different RL methods, such as DQN, REINFORCE and actor-critic, in various RL4Rec frameworks could reveal whether comparisons of RL methods generalize across different settings. The resulting insights could greatly aid researchers and practitioners in the RL4Rec domain.

IMPLEMENTATION RESOURCES AND DATA

To facilitate the reproducibility of the reported results, this study only made use of publicly available data. Our complete experimental implementation is publicly available with detailed instructions for reproducing our experiments at https://github.com/BetsyHJ/RL4Rec.

ACKNOWLEDGEMENTS

This research was supported by the Hybrid Intelligence Center, a 10-year program funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, https://hybrid-intelligence-centre.nl and partially by the Google Research Scholar Program. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Panagiotis Adamopoulos and Alexander Tuzhilin. 2014. On Over-Specialization and Concentration Bias of Recommendations: Probabilistic Neighborhood Selection in Collaborative Filtering Systems. In *RecSys*. ACM, 153–160.

[2] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. 2021. Reinforcement Learning based Recommender Systems: A Survey. *arXiv preprint arXiv:2101.06286* (2021).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[5] Lucas Bernardi, Sakshi Batra, and Cintia Alicia Bruscantini. 2021. Simulations in Recommender Systems: An Industry Perspective. *arXiv preprint arXiv:2109.06723* (2021).

[6] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and Debias in Recommender System: A Survey and Future Directions. *arXiv preprint arXiv:2010.03240* (2020).

[7] 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 *WSDM*. ACM, 456–464.

[8] Ruey-Cheng Chen, Qingyao Ai, Gaya Jayasinghe, and W Bruce Croft. 2019. Correcting for Recency Bias in Job Recommendation. In *CIKM*. ACM, 2185–2188.

[9] 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 *KDD*. ACM, 1187–1196.

[10] 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 *ICML*. PMLR, 1052–1061.

[11] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential Recommendation with User Memory Networks. In *WSDM*. ACM, 108–116.

[12] 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.

[13] 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).

[14] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *information retrieval* 4, 2 (2001), 133–151.

[15] Claudio Greco, Alessandro Suglia, Pierpaolo Basile, and Giovanni Semeraro. 2017. Converse-Et-Impera: Exploiting Deep Learning and Hierarchical Reinforcement Learning for Conversational Recommender Systems. In *AIxIA*. Springer, 372–386.

[16] Sara Hajian, Francesco Bonchi, and Carlos Castillo. 2016. Algorithmic Bias: From Discrimination Discovery to Fairness-Aware Data Mining. In *KDD*. ACM, 2125–2126.

[17] F Maxwell Harper and Joseph A Konstan. 2015. The Movielens Datasets: History and Context. *TiiS* 5, 4 (2015), 1–19.

[18] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer Product-based Neural Collaborative Filtering. In *IJCAI*. ijcai.org, 2227–2233.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. ACM, 173–182.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. (2016).

[21] Daniel G Horvitz and Donovan J Thompson. 1952. A Generalization of Sampling without Replacement from a Finite Universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[22] 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 *RecSys*. ACM, 190–199.

[23] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks. In *SIGIR*. ACM, 505–514.

[24] 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).

[25] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Morgane Lustman, Vince Gatto, Paul Covington, et al. 2019. Reinforcement Learning for Slate-based Recommender Systems: A Tractable Decomposition and Practical Methodology. *arXiv preprint arXiv:1905.12767* (2019).

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

[27] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *WSDM*. ACM, 781–789.

[28] Joseph DY Kang, Joseph L Schafer, et al. 2007. Demystifying Double Robustness: A Comparison of Alternative Strategies for Estimating a Population Mean from Incomplete Data. *Statistical science* 22, 4 (2007), 523–539.

[29] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A Contextual-bandit Approach to Personalized News Article Recommendation. In *WWW*. ACM, 661–670.

[30] Huizhi Liang. 2020. Drprofiling: Deep Reinforcement User Profiling for Recommendations in Heterogenous Information Networks. *TKDE* (2020).

[31] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous Control with Deep Reinforcement Learning. In *ICLR (Poster)*.

[32] Yuanguo Lin, Yong Liu, Fan Lin, Pengcheng Wu, Wenhua Zeng, and Chunyan Miao. 2021. A Survey on Reinforcement Learning for Recommender Systems. *arXiv preprint arXiv:2109.10665* (2021).

[33] Dong Liu and Chenyang Yang. 2019. A Deep Reinforcement Learning Approach to Proactive Content Pushing and Recommendation for Mobile Users. *IEEE Access* 7 (2019), 83120–83136.

[34] Feng Liu, Huifeng Guo, Xutao Li, Ruiming Tang, Yunming Ye, and Xiuqiang He. 2020. End-to-End Deep Reinforcement Learning based Recommendation with Supervised Embedding. In *WSDM*. ACM, 384–392.

[35] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, Yuzhou Zhang, and Xiuqiang He. 2020. State Representation Modeling for Deep Reinforcement Learning based Recommendation. *Knowledge-Based Systems* 205 (2020), 106170.

[36] Su Liu, Ye Chen, Hui Huang, Liang Xiao, and Xiaojun Hei. 2018. Towards Smart Educational Recommendations with Reinforcement Learning in Classroom. In *TALE*. IEEE, 1079–1084.

[37] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H Chi. 2020. Off-policy Learning in Two-stage Recommender Systems. In *WWW*. ACM / IW3C2, 463–473.

[38] Benjamin M Marlin and Richard S Zemel. 2009. Collaborative Prediction and Ranking with Non-Random Missing Data. In *RecSys*. ACM, 5–12.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level Control through Deep Reinforcement Learning. *nature* 518, 7540 (2015), 529–533.

[40] Tien T Nguyen, Pik-Mai Hui, F Maxwell Harper, Loren Terveen, and Joseph A

Konstan. 2014. Exploring the Filter Bubble: the Effect of Using Recommender Systems on Content Diversity. In *WWW*. ACM, 677–686.
[41] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy Gradients for Contextual Recommendations. In *WWW*. ACM, 1421–1431.
[42] Eli Pariser. 2011. *The Filter Bubble: How the New Personalized Web is Changing What We Read and How We Think*. Penguin.
[43] Bruno Pradel, Nicolas Usunier, and Patrick Gallinari. 2012. Ranking with Non-Random Missing Ratings: Influence of Popularity and Positivity on Evaluation Metrics. In *RecSys*. ACM, 147–154.
[44] Faxin Qi, Xiangrong Tong, Lei Yu, and Yingjie Wang. 2019. Personalized Project Recommendations: Using Reinforcement Learning. *EURASIP Journal on Wireless Communications and Networking* 2019, 1 (2019), 1–17.
[45] James M Robins, Andrea Rotnitzky, and Lue Ping Zhao. 1994. Estimation of Regression Coefficients When Some Regressors are not Always Observed. *Journal of the American statistical Association* 89, 427 (1994), 846–866.
[46] 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).
[47] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *ICML*. JMLR.org, 1670–1679.
[48] 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 *RecSys*. ACM, 491–495.
[49] 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 *AAAI*, Vol. 33. AAAI Press, 4902–4909.
[50] Harald Steck. 2010. Training and Testing of Recommender Systems on Data Missing Not at Random. In *KDD*. ACM, 713–722.
[51] Harald Steck. 2011. Item Popularity and Recommendation Accuracy. In *RecSys*. ACM, 125–132.
[52] Yueming Sun and Yi Zhang. 2018. Conversational Recommender System. In *SIGIR*. ACM, 235–244.
[53] Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. 2015. Deep Reinforcement Learning with Attention for Slate Markov Decision Processes with High-Dimensional States and Actions. *arXiv preprint arXiv:1512.01124* (2015).
[54] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement Learning: An Introduction*. MIT press.
[55] Ronald J Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning* 8, 3 (1992), 229–256.
[56] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent Recommender Networks. In *WSDM*. ACM, 495–503.
[57] Yikun Xian, Zuohui Fu, Shan Muthukrishnan, Gerard De Melo, and Yongfeng Zhang. 2019. Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. In *SIGIR*. ACM, 285–294.
[58] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A Dynamic Recurrent Model for Next Basket Recommendation. In *SIGIR*. ACM, 729–732.
[59] Tong Yu, Yilin Shen, Ruiyi Zhang, Xiangyu Zeng, and Hongxia Jin. 2019. Vision-Language Recommendation via Attribute Augmented Multimodal Reinforcement Learning. In *MM*. ACM, 39–47.
[60] Zhang Yuyan, Su Xiayao, and Liu Yong. 2019. A Novel Movie Recommendation System based on Deep Reinforcement Learning with Prioritized Experience Replay. In *ICCT*. IEEE, 1496–1500.
[61] Shuo Zhang and Krisztian Balog. 2020. Evaluating Conversational Recommender Systems via User Simulation. In *KDD*. ACM, 1512–1520.
[62] Chenfei Zhao and Lan Hu. 2019. CapDRL: A Deep Capsule Reinforcement Learning for Movie Recommendation. In *PRICAI*. Springer, 734–739.
[63] 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).
[64] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *RecSys*. ACM, 95–103.
[65] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *KDD*. ACM, 1040–1048.
[66] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2017. Deep Reinforcement Learning for List-Wise Recommendations. *arXiv preprint arXiv:1801.00209* (2017).
[67] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly Learning to Recommend and Advertise. In *KDD*. ACM, 3319–3327.
[68] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *WWW*. ACM, 167–176.