MBCAL: Sample Efficient and Variance Reduced Reinforcement Learning for Recommender Systems

Fan Wang*† wang.fan@baidu.com Baidu Inc.

Xiaomin Fang* fangxiaomin01@baidu.com Baidu Inc.

Lihang Liu* liulihang@baidu.com Baidu Inc.

Hao Tian† tianhao@baidu.com Baidu Research

Zhiming Peng pengzhiming01@baidu.com Baidu Inc.

ABSTRACT

In recommender systems such as news feed stream, it is essential to optimize the long-term utilities in the continuous user-system interaction processes. Previous works have proved the capability of reinforcement learning in this problem. However, there are many practical challenges to implement deep reinforcement learning in online systems, including low sample efficiency, uncontrollable risks, and excessive variances. To address these issues, we propose a novel reinforcement learning method, namely model-based counterfactual advantage learning (MBCAL). The proposed method takes advantage of the characteristics of recommender systems and draws ideas from the model-based reinforcement learning method for higher sample efficiency. It has two components: an environment model that predicts the instant user behavior one-by-one in an auto-regressive form, and a future advantage model that predicts the future utility. To alleviate the impact of excessive variance when learning the future advantage model, we employ counterfactual comparisons derived from the environment model. In consequence, the proposed method possesses high sample efficiency and significantly lower variance; Also, it is able to use existing user logs to avoid the risks of starting from scratch. In contrast to its capability, its implementation cost is relatively low, which fits well with practical systems. Theoretical analysis and elaborate experiments are presented. Results show that the proposed method transcends the other supervised learning and RL-based methods in both sample efficiency and asymptotic performances.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Computing methodologies → Sequential decision making**.

*The authors contributed equally to this research.
†Corresponding Authors.

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.
KDD '20, August 22–27, 2020, San Diego, CA, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

Recommender Systems, Model-based Reinforcement Learning

ACM Reference Format:
Fan Wang, Xiaomin Fang, Lihang Liu, Hao Tian, and Zhiming Peng. 2020. MBCAL: Sample Efficient and Variance Reduced Reinforcement Learning for Recommender Systems. In *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 22–27, 2020, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A recommender system (RS) provides users with personalized contents, which significantly improves the efficiency of information acquisition. Nowadays, a typical RS such as news feed stream needs to address multiple steps of user-system interactions in a session. The recommended content in historical interactions can affect the subsequent user behaviors. For instance, exploration of new topics may stimulate the user's interest in related topics; Repeating overlapped contents can make the user lose his/her interest quickly. Traditional recommender systems employ collaborative filtering [15, 22], or neural networks [2, 11, 12] to estimate users' instant behaviors, e.g., instant clicks. However, merely focusing on users' instant behaviors causes many problems, such as overcrowded recommendations, which damage users' experience in the long run. Recently there is increased attention on applying deep reinforcement learning (Deep RL) [17, 20] to recommender systems, which models the user-system interactions as Markov Decision Processes (MDP). A large number of the studies in this area lies in model-free reinforcement learning (MFRL) methods such as Policy Gradient [1], DDPG [6], DRR [18], DeepPage [32] and DQN based recommender systems [33, 34]. However, many challenges remain in this area. One of them is the over-consumption of data in the training process, which is also referred to as low sample efficiency. Another challenge of MFRL is the practical risks in implementation. On the one hand, on-policy RL can hardly utilize off-policy user logs for training, which raises challenges in online infrastructures and performances at the early stage. On the other hand, off-policy RL suffers from the risk of falling into *Deadly Triad* [27]. It refers to the case of non-convergence when combining off-policy RL with function approximation (such as neural networks) and offline training.

As an alternative choice of MFRL, model-based RL (MBRL) possesses higher sample efficiency and lower practical risks. In MBRL, an environment model (also known as world model) is employed to predict the instant feedbacks and state transitions, and a planning module is implemented to search for an optimal trajectory [4]. However, MBRL needs much computation when coming to inference. To make things worse, planning is completely infeasible in a multi-stage retrieval framework, which is widely used in modern recommender systems. In such systems, an earlier stage generates the candidate set of items for the next one; thus, the candidates can not be predetermined. To avoid those issues, authors have proposed to apply Dyna algorithms in recommender systems [19, 35]. The Dyna algorithm [24] accelerate the convergence through generating virtual interaction by taking advantage of the environment model. However, as a cost of faster convergence, Dyna suffers from losses in asymptotic performance, due to the accumulation of error from the virtual interactions.

Excessive variance of gradients in optimization is an important challenge of deploying RL as well. The variance may originate from stochastic transition, noisy rewards, and stochastic policy. Longer horizons are found to exacerbate the variance. Excessive variance significantly slows down the convergence and introduces instabilities. Previous work [23, 30] has shed light on this issue by showing that using advantage function instead of value function can reduce the variance and thus improve the performance. However, those proposals aim at MFRL, and variance reduction in MBRL has not been studied yet.

Specifically, in recommender systems, the variance may come from the following aspects. First, there are extensive noises in the observed user feedbacks. For example, some users are more prone to give positive/negative feedbacks than the others. Even for a single user, he/she may behave differently at different times of a day (e.g., before sleeping vs. at work). Second, for stochastic policy, re-sampling the trajectory starting from any state can lead to variant long-term returns. Although the influence of variances can be alleviated by inducing from a sufficiently large amount of data, the variances still have negative impacts due to the sparsity of the data for specific user and item.

To clearly explain the influence of variances, we show a demonstration of the interaction process in Figure 1(a). For each step, the agent selects an item for display, and the user returns his/her feedbacks. An *observed trajectory* includes multiple steps of interactions, as T_a and T_b shown in Figure 1(b). In our settings, the candidate user behaviors include "Like (Thumbs Up)" and "Unlike (Thumbs Down)", and the utility is the number of "Likes" in the trajectory. Here, we consider the document d_3 in the 3rd step in trajectory T_a, which yields 2 "Likes" considering the instant and future utilities. It is hard to judge whether d_3 is a superior action here due to the lack of comparison. To give a better evaluation, we can find a comparison such as Trajectory T_b in Figure 1(b). T_b possesses the same historical interactions as T_a, which implies that the user share many interests with that of T_a. The trajectory starting from d'_3 in T_b yields 3 following "Likes", by which we may judge that d'_3 is better than d_3 . However, it is a quite arbitrary conclusion, because the difference in future utility can possibly be attributed to the user biases, or the quality of follow-up items (d_4, d_5 vs. d'_4, d'_5).

Aiming at further reducing the variances, a key idea of our work is to compare T_a with another trajectory, T_c in Figure 1c. T_c shares all the contexts with T_a, including the user, historical interactions, and follow-up items (d_4, d_5), except for replacing the current document d_3 with some other documents. By comparing T_a with T_c, we can come to a more solid conclusion on the advantage of taking d_3. Unfortunately, it is impossible to find records such as T_c from user logs, as a user can not possibly go through the same trajectory twice. However, by taking advantage of the environment model, we can do simulations into the future (often referred to as simulated rollout), and we can indeed generate trajectories like T_c.

Following the idea mentioned above, we propose a novel MBRL solution toward RS, namely the **Model-based Counterfactual Advantage Learning** (MBCAL). First, the overall utility is decomposed into the instant utility (the rewards acquired in the current step) and the future utility (the rewards acquired in the future). The instant utility is naturally predicted with the environment model, and the future utility is approximated through simulated rollout. Second, to further reduce the variance in the future utilities, we try to do two comparative simulated rollouts. Before doing so, we introduce the masking item to the environment model, which allows us to generate simulated rollouts by masking the document in the step that we are interested in (the trajectory T_c). We then calculate the counterfactual future advantage (CFA) as the difference of the future utility with and without masking. At last, we introduce the future advantage model to approximate the CFA.

We conduct simulative experiments by utilizing three real-world datasets. The methods for comparison include supervised learning, MFRL and MBRL. We also put our attention on Batch-RL and Growing Batch-RL settings[16], which is more compatible with the practical infrastructures. Extensive results of experiments show the superiority of the proposed method.

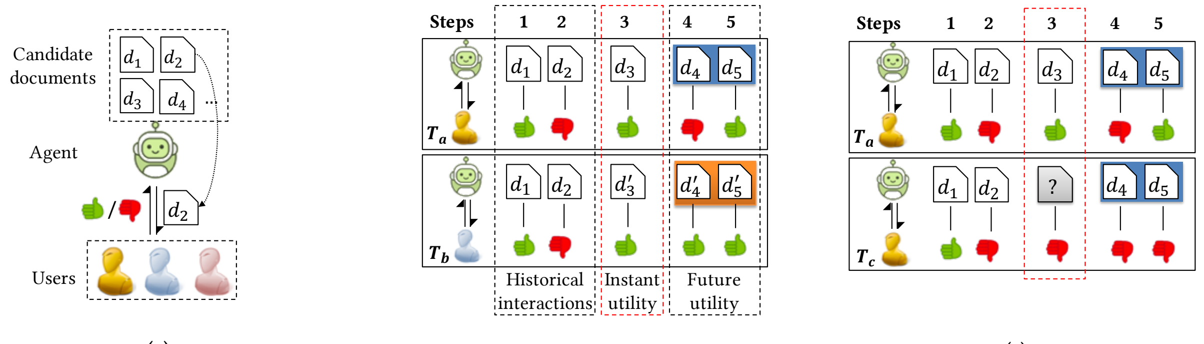
2 PRELIMINARIES

2.1 Problem Settings and Notations

We formalize the recommendation as Markov Decision Processes (MDP), denoted with the symbols $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mu)$. The details are explained as follows:

- *State* $s_t \in \mathcal{S}$ represents the unknown user interests and contextual information at step t. As it is impossible to know the user interest exactly, the observed user interaction history is frequently used as the state descriptor [18, 31], i.e.,
$$s_t = (o_1, o_2, ..., o_{t-1}) \tag{1}$$
where we use o_t to represent the pair of exposed item $a^{\mathbf{o}}_t$ and corresponding feedback $b^{\mathbf{o}}_t$, i.e., $o_t = (a^{\mathbf{o}}_t, b^{\mathbf{o}}_t)$. For simple we also use $\mathbf{o}_{[1:t-1]}$ to represent the trajectory $o_1, o_2, ..., o_{t-1}$.
- *Action* $a_t \in \mathcal{A}_t$ denotes the selected item by the agent, with $\mathcal{A}_t$ being the candidate set that is passed from the earlier stage of the recommender system.
- *Reward* $r_t \in \mathcal{R}$ is the utility that we want to maximize. Usually, it depends on the observed user behaviors.
- *Conditional Transition Probabilities* $\mu(s'|s, a)$ denotes the transition probability to state s' given state and action pair (s, a), which is hidden in most cases.

Additionally, we denote the user feedback (behavior) at step t with $b_t \in \mathcal{B}$, where $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_n\}$ is the set of candidate user behaviors (e.g., "Click", "Skip", "Click and Thumbs Up"). Also, notice that we use the superscript $\mathbf{o}$ to denote that an action a_t


Candidate documents
d_1 d_2 d_3 d_4 ...
Agent
d_2
Users
Steps 1 2 3 4 5
T_a
T_b
d_1 d_2 d_3 d_4 d_5
d_1 d_2 d'_3 d'_4 d'_5
Historical interactions
Instant utility
Future utility
Steps 1 2 3 4 5
T_a
T_c
d_1 d_2 d_3 d_4 d_5
d_1 d_2 ? d_4 d_5

(a) (b) (c)

Figure 1: (a) Illustration of the user-system interaction; (b) Two examples of trajectories of interaction, T_a and T_b, generated by different users; (c) The original trajectory T_a and couterfactual comparison T_c from the same user

Table 1: Notations.

Notations	Descriptions
$\mathcal{B}$	Categories of user behaviors, e.g., $\mathcal{B}$=("Skip", "Click", "Click and Thumbs up")
$\mathcal{R}$	Rewards corresponding to the behaviors in $\mathcal{B}$, e.g., $\mathcal{R} = (0, 1, 5)$
r	$r \in \mathcal{R}$, reward / utility
s	state
a	action
$\pi(a\|s)$	policy for the recommendation
$r(s, a)$	reward function
$\mu(s'\|s, a)$	transition probabilities
$Q^\pi(s, a)$	state-action value function of π
$V^\pi(s)$	state value function of π
$\mathbf{o}$	observed interactive trajectory $\mathbf{o} = (a^{\mathbf{o}}_1, b^{\mathbf{o}}_1, a^{\mathbf{o}}_2, b^{\mathbf{o}}_2, ...)$
$\mathbf{o}_{[t_1:t_2]}$	sub-trajectories from step t_1 to t_2
$a^{\mathbf{o}}_t$	action taken at t in trajectory $\mathbf{o}$
$b^{\mathbf{o}}_t$	$b^{\mathbf{o}}_t \in \mathcal{B}$, user behavior observed at t in $\mathbf{o}$
$\mathcal{D}^\pi$	collection of trajectories with policy π

and user feedback b_t belongs to a specific trajectory, given $a^{\mathbf{o}}_t$ and $b^{\mathbf{o}}_t$. Without losing generality, we focus on maximizing the overall utility within a fixed T steps of interactions. In addition to the symbols mentioned above, we also adopt other commonly used notations in RL, shown in Table 1.

2.2 RL-based Recommender Systems

Without confusion, we replace state s_t with the trajectory $\mathbf{o}_{[1:t-1]}$, and denote the value and policy function as $Q^\pi(\mathbf{o}_{[1:t-1]}, a)$ and $\pi(a|\,\mathbf{o}_{[1:t-1]})$, respectively. It is worth to notice that other user-side features (user-ids, user properties, context features such as time of the day) can be involved in the state representation without any difficulty. However, for simplicity, we omit the notation of those features.

Typical MFRL methods try to learn the function approximator denoted as Q_θ, with θ being trainable parameters, which is optimized by minimizing the loss function:

$$\mathcal{L}_{\text{MFRL}}(\theta) = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{o}\in\mathcal{D}}\frac{1}{T}\sum_{t=1}^{T}[\hat{Q}_{\text{target}} - Q_\theta(\mathbf{o}_{[1:t-1]}, a^{\mathbf{o}}_t)]^2. \quad (2)$$

$\hat{Q}_{\text{target}}$ here represents the value backup (backup: updating the value of current states with that of future states [26]), which is calculated differently in different algorithms. E.g., Deep Q Network (DQN) [20] uses the temporal difference errors, where $\hat{Q}_{\text{target}} = r_t + \gamma \max_{a'} Q_{\theta'}(\mathbf{o}_{[1:t]}, a')$. θ' here is the set of parameters which is periodically copied from θ during the optimization process.

2.3 Growing Batch-RL

As online update is typically not achievable in realistic recommender systems, we are more interested in Batch-RL and Growing Batch-RL [16] settings. Batch-RL refers to the policy learning with static user logs. It can usually be used to evaluate the capability of utilizing the offline data. Growing Batch-RL lies somewhere in between the Batch-RL and online learning. The data can be recollected in a batch manner and used for policy update iteratively. Many recently proposed RL-based systems use a framework that is close to Growing Batch RL settings [1, 32, 34]. More specifically, Growing Batch RL includes two periodic interleaving stages:

Data Collection. The agent uses the policy π_k to interact with the environment (the users). The policy is kept static during this process. The collected interactive trajectories are denoted as $\mathcal{D}^{\pi_k}$.

Policy Update. The interactive trajectories $\mathcal{D}^{\pi_k}$ are used for training, the agent update the policy from π_k to π_{k+1}.

The essential problems of Batch-RL and Growing Batch-RL are to guarantee the **Policy Improvement**, i.e., how much the performance of π_{k+1} is improved compared with π_k.

3 METHODOLOGY

The fundamental idea of MBCAL is explained with Figure 2. We use two models to approximate the instant user behavior and the future advantage separately: the Masked Environment Model (MEM) and the Future Advantage Model (FAM). For the training, we start with optimizing the environment model to predict user behaviors, where we also introduce the masking item into the model. With the MEM, we can calculate the Counterfactual Future Advantage (CFA) by

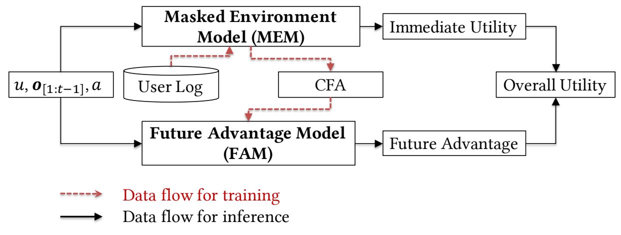


Figure 2: Overview of MBCAL

comparing the future utility of masking the action or not. CFA further serves as the label of FAM. For the inference, we use the combination of the two models to pick actions.

In this section, we first formalize the environment model, and then we describe MEM, FAM, and the learning process overall. They are followed by the theoretical analysis of the proposed method.

3.1 Environment Modeling

As the most common setting, an environment model predicts the transition and the reward separately. Here, we use $\hat{\mu}(s_t, a_t, s_{t+1})$ to approximate $\mu(s_{t+1}|s_t, a_t)$ and $\hat{r}(s_t, a_t)$ to approximate the reward $r(s_t, a_t)$. Specifically, to derive the formulation of environment model in RS, we use Equation (1), and $\mu(s_{t+1}|s_t, a_t)$ can be rewritten by Equation (3).

$$\begin{aligned}\mu(s_{t+1}|s_t, a_t) &= \mu(\mathbf{o}_{[1:t]}|\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}) \\ &= \mu(\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}, b_t^{\mathbf{o}}|\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}) \\ &= \mu(b_t^{\mathbf{o}}|\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}) \end{aligned} \tag{3}$$

In other words, the prediction of transition degenerates into the prediction of instant user behaviors. Notice that reward is dependent on the user behavior only as well, thus it is possible to use only one model in place of $\hat{\mu}$ and $\hat{r}$. We introduce $f_\phi(\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}, b_t^{\mathbf{o}})$ with trainable parameters ϕ to approximate $\mu(b_t^{\mathbf{o}}|\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}})$, which predicts the probability of the next user behavior being $b_t^{\mathbf{o}}$. The transition and the reward are then naturally approximated with

$$\hat{\mu}(s_t, a_t, s_{t+1}) = f_\phi(\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}, b_t^{\mathbf{o}}). \tag{4}$$

$$\hat{r}(s_t, a_t) = \sum_n \mathcal{R}_n \cdot f_\phi(\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}, \mathcal{B}_n), \tag{5}$$

3.2 Masked Environment Model

To eliminate the intractable noises hidden in the feedbacks, we introduce a masking item into the model. The motivation of this is to try to find a counterfactual comparison to the current trajectory, which answers the question: "If this action was not taken, what would the future behavior be like?" To do so, we introduce a virtual item a_M, which is represented by a trainable embedding vector. For convenience, given an observation trajectory $\mathbf{o}$, we denote the trajectory where the actions at positions $y = \{t_1, t_2, ...\}$ are replaced by a_M as $\mathcal{M}(\mathbf{o}, y, a_M)$. For example, suppose $y = \{2, 3\}$, it gives a masked trajectory of

$$\mathcal{M}(\mathbf{o}, \{2,3\}, a_M) = (a_1^{\mathbf{o}}, b_1^{\mathbf{o}}, a_M, b_2^{\mathbf{o}}, a_M, b_3^{\mathbf{o}}, a_4^{\mathbf{o}}, b_4^{\mathbf{o}}, ...),$$

where actions $a_2^{\mathbf{o}}$ and $a_3^{\mathbf{o}}$ in $\mathbf{o}$ are replaced by a_M.

The training is straightforward. We sample random positions $y^{\mathbf{o}}$ for each trajectory $\mathbf{o}$, such that each position has uniform probability of p_{mask} to be replaced. We want the MEM to recover the user behavior as close as possible in case some items are masked. With the collected masked trajectories $\mathcal{D}_M = \{\mathcal{M}(\mathbf{o}, y^{\mathbf{o}}, a_M)| \mathbf{o} \in \mathcal{D}\}$, we maximize the likelihood, or minimize the negative log-likelihood (NLL).

$$\mathcal{L}_{\text{MEM}}(\phi) = -\frac{1}{|\mathcal{D}_M|}\sum_{\mathbf{o}\in\mathcal{D}_M}\frac{1}{T}\sum_{t=1}^{T}[\log f_\phi(\mathbf{o}_{[:t-1]}, a_t^{\mathbf{o}}, b_t^{\mathbf{o}})]. \tag{6}$$

To model the sequential observations, the architecture of MEM follows that of session-based recurrent RS([12, 13]). We use Gated Neural Network[9] to encode the trajectory $\mathbf{o}_{[1:t-1]}$. As we need to encode $\mathbf{o}_{[1:t-1]}$ and $a_t^{\mathbf{o}}$ at the same time, we concatenate the input in a staggered way, equivalent to the setting of [21]. For each step t, the model takes $b_{t-1}^{\mathbf{o}}$ and $a_t^{\mathbf{o}}$ as input and output the probability of the next possible behavior. An additional b_s is introduced as the start of the observed user behavior (see Figure 3). Concretely, the architecture is formulated as follows.

$$h_0^{MEM} = \mathbf{Emb}(u), \tag{7}$$

$$x_t^{MEM} = \mathbf{Concat}(\mathbf{Emb}(b_{t-1}), \mathbf{Emb}(a_t)), \tag{8}$$

$$h_t^{MEM} = \mathbf{GRU}(h_{t-1}^{MEM}, x_t^{MEM}), \tag{9}$$

$$f_\phi = \mathbf{Softmax}(\mathbf{MLP}(h_t^{MEM})). \tag{10}$$

Here **Emb** denotes a representation layer; **Concat** denotes a concat operation and **MLP** denotes multilayer perceptron; **GRU** represents a Gated Recurrent Unit.

3.3 Counterfactual Future Advantage

With the MEM, we can estimate the difference in future utilities between the original trajectory and the counterfactual comparison, namely the Counterfactual Future Advantage (CFA). Specifically, given the trained MEM f_ϕ, we first define the Simulated Future Reward (SFR, denoted with $\hat{R}^{\text{future}}$) of the observed trajectory $\mathbf{o}$ at time step $t \in [1, T]$ as

$$\hat{R}_\phi^{\text{future}}(\mathbf{o}, t) = \sum_{\tau=t+1}^{T}\gamma^{\tau-t} r_\phi(\mathbf{o}_{[1:\tau-1]}, a_\tau^{\mathbf{o}}). \tag{11}$$

We then calculate CFA (denoted with $\hat{A}^{\text{future}}$) by subtracting the SFR of counterfactual comparison from the original one, see Equation (12).

$$\hat{A}_\phi^{\text{future}}(\mathbf{o}, t) = \hat{R}_\phi^{\text{future}}(\mathbf{o}, t) - \hat{R}_\phi^{\text{future}}(\mathcal{M}(\mathbf{o}, \{t\}, a_M), t). \tag{12}$$

Finally, we introduce the Future Advantage Model (FAM) denoted with $g_\eta(\mathbf{o}_{[1:t-1]}, a_t)$, with trainable parameters η. To train FAM, we minimize the mean square error shown in Equation (13).

$$\mathcal{L}_{\text{FAM}}(\eta) = -\frac{1}{|\mathcal{D}|}\sum_{(\mathbf{o})\in\mathcal{D}}\frac{1}{T}\sum_{t=1}^{T}[g_\eta(\mathbf{o}_{[1:t-1]}, a_t^{\mathbf{o}}) - \hat{A}_\phi^{\text{future}}(\mathbf{o}, t)]^2. \tag{13}$$

FAM takes the equivalent input as the MEM. We use the same neural architecture as MEM except for the last layer, but with different parameters. For the last layer FAM predicts a scalar (the

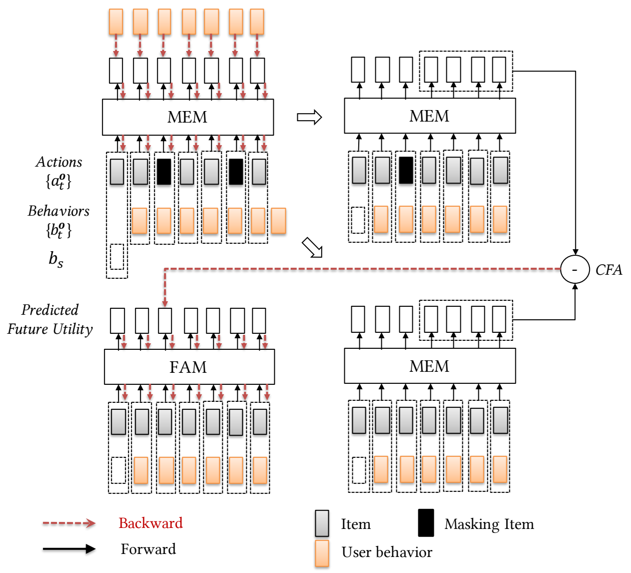


Figure 3: Illustration of the MBCAL architectures.

advantage) instead of a distribution, shown as follows:

$$h_0^{FAM} = \mathbf{Emb}(u), \tag{14}$$

$$x_t^{FAM} = \mathbf{Concat}(\mathbf{Emb}(b_{t-1}), \mathbf{Emb}(a_t)), \tag{15}$$

$$h_t^{FAM} = \mathbf{GRU}(h_{t-1}^{FAM}, x_t^{FAM}), \tag{16}$$

$$g_\eta = \mathbf{MLP}(h_t^{FAM}). \tag{17}$$

3.4 Summary of MBCAL

An integrate illustration of MBCAL is shown in Figure 3. For the inference, we select the item(action) based on both the MEM and FMA. Formally, given the user information u and the observation trajectory $\mathbf{o}_{[1:t-1]}$, we pick the next action according to Equation (18).

$$a_t^* = argmax_a[r_\phi(\mathbf{o}_{[1:t-1]}, a) + g_\eta(\mathbf{o}_{[1:t-1]}, a)]. \tag{18}$$

To avoid the local optimum in policy improvement, we employ ϵ-greedy strategy [25]. With probability ϵ, we select a random action, and with the left probability, we select according to Equation (18). It is written as

$$a_t = \begin{cases} a_t^*, & \text{with probability } 1-\epsilon; \\ \text{random action } a \in \mathcal{A}_t, & \text{with probability } \epsilon. \end{cases} \tag{19}$$

MBCAL fits well with the Growing Batch-RL settings. The summary of the algorithm is shown in Algorithm 1, with the function *PolicyUpdate* shown in Algorithm 2. Although we use the notation π to represent the policy, we do not require any explicit formulation of the policy; The common policy gradient type algorithms require the explicit form of policy, which is hard to acquire in many recommender systems.

The essence of the variance reduction in MBCAL lies in Equation (12), where the subtraction eliminates the noises from user feedbacks and other sources. We borrow ideas from the advantage function [29], however, CFA is different from the advantage function in that we do not resample the trajectory but we keep the rest

Algorithm 1 Model-Based Counterfactual Advantage Learning(MBCAL)

```
1: Initial Policy $\pi_0$.
2: for k = 0,1,2,... until convergence do
3:     Use policy $\pi_k$ to interact with users for $N$ trajectories, and record the interaction history $\mathcal{D}^{\pi_k}$.
4:     $f_\phi, g_\eta$ = PolicyUpdate($\mathcal{D}^{\pi_k}$).
5:     Set $\pi_{k+1}$ according to Equation (19).
6: Return $\pi_{k+1}$ in the last iteration.
```

Algorithm 2 *PolicyUpdate*($\mathcal{D}$)

```
1: Input: Interaction history $\mathcal{D}$.
2: Randomly masking $\mathcal{D}$ to acquire $\mathcal{D}_M$.
3: Minimize $\mathcal{L}_{\text{MEM}}$(Equation (6)) on $\mathcal{D}_M$ to optimize $f_\phi$.
4: For $\mathbf{o} \in \mathcal{D}$, calculate $\hat{A}_{\mathbf{o},t}$ with $t \in [1,T]$ by Equation (11) and Equation (12)
5: Minimize $\mathcal{L}_{\text{FAM}}$(Equation (13)) on $\mathcal{D}$ to optimize $g_\eta$.
6: Return $f_\phi, g_\eta$.
```

of trajectory (that is $\mathbf{o}_{[t+1:T]}$) unchanged. Although this could bring severe biases in many MDP problems, we argue that the recommender systems embrace weaker correlations between sequential decisions than the other problems (such as robot control and game control). Additionally, as the FAM averages out CFA across different trajectories, the bias turn out to be negligible compared with the benefits of reducing the variances.

4 EXPERIMENTS

4.1 Datasets

Evaluation of RL-based recommender systems is challenging. The most convincing metric requires running online A/B tests, but it is not only too costly but also too risky to compare all the baselines in an online system. Offline evaluation of long-term utility using user logs is tricky as we can not have the feedback if the agent recommends a different item than what was stored in the log. In order to thoroughly study the performance of the proposed systems, we follow the previous works to build simulators [6, 14, 19, 32]. But instead of synthetic simulators, we use real-data-driven simulators. The datasets used include: MovieLens ml-20m[1] [10], Netflix Prize[2] and NewsFeed[3], as shown in Table 2. Details of the datasets are explained as follows.

- *MovieLens ml-20m*: The dataset describes 5-star rating activities from MovieLens. The user behavior $\mathcal{B} = [0, 1, 2, 3, 4, 5]$ corresponds to the star ratings, with the reward to be $\mathcal{R} = [0, 1, 2, 3, 4, 5]$. There are 3 kinds of features (*movie-id*, *movie-genre* and *movie-tag*).
- *Netflix Prize*: The dataset is a 5-star rating dataset from Netflix. The reward follow the setting of MovieLens. There are only 1 type of features (*movie-id*).

[1]http://files.grouplens.org/datasets/movielens/ml-20m-README.html
[2]https://www.kaggle.com/netflix-inc/netflix-prize-data
[3]Data collected from Baidu App News Feed System

- *NewsFeed*: The dataset is collected from a real online news recommendation system. We focus on predicting the dwelling time on the clicked news. The dwelling time is partitioned into 12 levels (e.g., dwelling time < 30, 30 < dwelling time < 40, ...), corresponding to 12 different user behaviors, with the corresponding rewards to be $\mathcal{R} = [1, 2, ..., 12]$. There are 7 kinds of features (*news-id*, *news-tag*, *news-title*, *news-category*, *news-topics*, *news-type*, *news-source*).

4.2 Experimental Settings

4.2.1 Simulator Details. In order to fairly evaluate different methods, it is necessary to avoid the agent in the evaluated system to "hack" the simulator. For this purpose, we add two special settings in the evaluation processes. First, all the agents to be evaluated are allowed to use only a *subset* of features, while the simulator uses the full feature set. In *MovieLens* and *Netflix*, only 1 feature (*movie-id*) is used in the agents. In *NewsFeed*, 4 kinds out of 7 are used (*news-id*, *category*, *news-type*, and *news-source*). Second, we artificially set the model architecture of the simulator to be different from that of the agents. We use the LSTM unit for the simulators, while GRU is used in the agents. To get a view of how close the simulator is to the real environment, we list the micro-F1, weighted-F1, and RMSE with respect to the accuracy of user behavior classification. Properties of datasets and simulators are shown in Table 2.[4] In *NewsFeed*, we also retrieved over 400 historical A-B test records online. Concerning long-term rewards, including total clicks or dwelling time of a session, the correlation of prediction of our simulators to the real case is 0.90+.

Table 2: Properties of Datasets and Simulators.

Properties	*MovieLens*	*Netflix*	*NewsFeed*
# of Users	130K	480K	920K
# of Items	20K	17K	110K
# of Different Labels	6	6	12
# of Types of Features	3	1	7
Size of Training Set	2.48M	4.53M	9.41M
Size of Validation Set	1.27M	2.27M	4.70M
Simulator Macro-F1	0.545	0.511	0.923
Simulator Weighted-F1	0.532	0.498	0.887
Simulator RMSE	0.770	0.848	1.810

4.2.2 Evaluation Settings. There are two types of iterations in the evaluation: the training round and the test round. For a training round, the agent to be evaluated produces actions by using ϵ-greedy policy ($\epsilon = 0.1$ throughout all experiments). It then updates its policy using the collected feedback from the simulator. In the test round, the algorithm produces actions by using a greedy policy, which is evaluated by the simulator. The data generated in the test round are not used in training. For each session in training or test rounds, we consider $T = 20$ steps of interaction between the simulator and the agent. Each training round or test round includes 256,000 sessions.

[4]The source code can be found at: https://github.com/LihangLiu/MBCAL

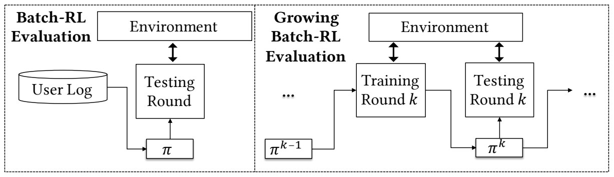


Figure 4: The evaluation processes.

For each experiment, we report the *average reward per session* in the test round, defined by $1/|\mathcal{D}_{test}| \cdot \sum_{\mathbf{o}\in\mathcal{D}_{test}} \sum_{t=1}^{T} r_t$, where $\mathcal{D}_{test}$ is the collection of trajectories in test round. Each experiment is repeated with different random seed for three times, the mean and variance of the score is reported. We also simulate the *Batch RL* and the *Growing Batch-RL* evaluation separately (Figure 4). In the *Batch RL* evaluation, the agent can use only the static user log for training, and it interacts with the simulator for test. In *Growing Batch RL* evaluation, for both training round and test round, the agent needs to interact with the simulator. The training round repeats up to 40 times.

4.3 Methods for Comparison

We compare different methods ranging from Supervised Learning (GRU4Rec), bandits (GRU4Rec (ϵ-greedy)) to MFRL (MCPE, DQN, DDQN, and DDPG) and MBRL (Dyna-Q). For bandits, LinUCB [3] is commonly used as a baseline. However, in our environments, LinUCB performs poorly due to the insufficient representative power of the linear model. Thus, we post the results of ϵ-greedy version of NN models (GRU4Rec (ϵ-greedy)) instead of LinUCB.

The methods for comparison include:

- **GRU4Rec** [13]: It adopts GRU to encode the interactive history to predict the instant user behavior. The model architecture is equivalent to the environment model. We use the entropy losses in GRU4Rec.
- **GRU4Rec (ϵ-greedy)**: It applies the ϵ-greedy selection of items in GRU4Rec during the training rounds.
- **DQN** [34]: A classical off-policy learning algorithm [20]. For state representation, to ensure a fair comparison between different learning algorithms, GRU is used to encode the historical observations, which is equivalent to GRU4Rec and our method.
- **DDQN** [34]: Double DQN [28] uses a different action selection for value backup to avoid the value overestimation in off-policy learning. The model architecture is kept equivalent to GRU4Rec.
- **DDPG** [32]: Deep Deterministic Policy Gradient (DDPG) [17] is an off-policy learning algorithm for continuous action space. The inferred action is used to select the item that lies closest to it for display (nearest neighbor). We use the same neural structure as GRU4Rec for both the actor and critic networks in DDPG.
- **MCPE**: Monte Carlo Policy Evaluation [5] is a straightforward value iteration algorithm. The Monte Carlo evaluation of the whole trajectory is used, i.e., we apply Equation (2) with $\hat{Q}_{\text{target}} = \sum_t r_t$. Again, we keep the model architecture equal to the other baselines.

- **Dyna-Q** [19, 35]: Dyna-Q is an MBRL method that augments DQN with the imagined rollouts from an environment model. The ratio of imagined rollouts to real trajectories is set to 1:1.
- **MBCAL**: The full version of our method.
- **MBCAL (w/o variance reduction)**: It is an ablative version of MBCAL. We use SFR instead of CFA as the label of FAM, i.e., in Equation (13), $\hat{R}_{\phi}^{\text{future}}$ is used instead of $\hat{A}_{\phi}^{\text{future}}$.

All the parameters are optimized by adam optimizer with learning rate = 10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The decay factor in the long-term reward is set to be $\gamma = 0.95$. The embedding sizes for the *item-id* and other id type features are all set to 32. The hidden size for MLP is set to 32. For training MEM in MBCAL, we use $p_{mask} = 0.20$ to generate the masked trajectories D_M. In DDPG, we found that high dimensional action space yields really poor performance. Thus we use a 4-dimensional action space. Correspondingly we use an additional layer to map the item representation into a 4-dimensional vector.

4.4 Experimental Results

Table 3: Average reward per session of different algorithms and datasets in Batch-RL evaluation.

Algorithms	Average reward per session		
	Movielens	*Netflix*	*NewsFeed*
GRU4Rec	77.93 ± 0.06	79.63 ± 0.02	11.58 ± 0.14
DDPG	70.99 ± 0.70	72.50 ± 0.35	10.90 ± 0.42
DQN	77.27 ± 0.06	77.75 ± 0.01	12.44 ± 0.33
DDQN	77.23 ± 0.02	77.70 ± 0.04	12.48 ± 0.17
MCPE	77.20 ± 0.10	77.70 ± 0.03	13.21 ± 0.53
Dyna-Q	77.25 ± 0.05	77.81 ± 0.02	13.04 ± 0.33
MBCAL	**78.02** ± 0.03	**79.71** ± 0.04	**16.32** ± 0.24
MBCAL(w/o variance reduction)	77.70 ± 0.04	79.50 ± 0.04	15.61 ± 0.38

4.4.1 Results of Batch-RL Evaluation. The results on Batch-RL evaluation are shown in Tab. 3. We evaluate the reward of a session according to the reward generated by the simulator. To conclude from the result, MFRL can not compete with MBRL in all three environments. As MFRL is sample inefficient, it tends to have poor startup performance. Surprisingly DDPG has the weakest performance in all three environments. By carefully investigating the value functions in DDPG, we found that DDPG overestimates the value function a lot compared with the other MFRL. We thought that the overestimation comes from value backups from continuous actions that may not correspond to realistic items. The overestimation problem in actor-critic methods has also been thoroughly investigated [7].

As is expected, the MBCAL leads the performance of all the tested systems with substantial margins, demonstrating its sample efficiency. However, for *Movielens* and *Netflix*, our method earns a smaller margin over the supervised learning method compared with that of *NewsFeed*. It is likely that the long term reward plays a more significant role in *NewsFeed* than the other two environments. Furthermore, as learning to predict long-term utility requires more data than the instant reward, the preponderance of RL has not yet been sufficiently revealed in Batch-RL settings. However, it is essential that the performance of MBCAL at the start stage is already state-of-the-art, which proves that MBCAL has low risks and high sample efficiency.

4.4.2 Results of Growing Batch-RL Evaluation. Figure 5 shows the results of the Online Evaluation in three environments. GRU4Rec(ϵ-greedy) surpasses the purely supervised learning GRU4Rec by a small margin in every environment, showing the benefit of exploration in online systems. Performances of DDPG in all three environments are again surprisingly bad. In Figure 5, we show DDPG curve in *NewsFeed* environment only because in the other two environments, DDPG lags too much behind all the other methods. We believe that continuous action space for recommendation systems with dynamic discrete item space can not work well enough.

With the assistance of the environment model, Dyna-Q gains some advantages at the beginning, but it gradually subsides as the learning continues. This phenomenon is just in line with the expectations, for the virtual experience quickly loses its benefit with the accumulation of sufficient real user feedback. MBCAL again keeps its performance ahead of the other methods in all the environments. Even for *Netflix* and *Movielens*, where the other RL-based system fails to gain any benefits over traditional GRU4Rec, MBCAL wins with a considerable margin. In *NewsFeed*, where the long term rewards play a more critical role, MBCAL strengthens the leading edge.

MCPE, DQN, DDQN, and Dyna-Q lag entirely behind the other methods, including supervised learning baselines in *Movielens* and *Netflix* environment, while this is not true in *NewsFeed*. We investigate the reason by setting the output of GRU4Rec to the instant reward instead of the user behavior classification, which turned the classification into regression, and the entropy loss to mean square error loss. We found a significant drop in performance in GRU4Rec, which is more consistent with the results in *NewsFeed*. The results show that classification and entropy loss benefit the system more than regressions. An explanation is that user behavior contains more abundant information than the rewards, which also made MBRL more advantageous than MFRL.

4.4.3 Analysis of the variance. The key point in MBCAL is the variance reduction through counterfactual comparisons. The previous proposals [8] suggest that the mean square error (MSE) in a well-trained model is composed of the model bias and the variance(noise) in the labels. As we use equivalent neural architectures in all the methods for comparison, they share the same model bias. Therefore the mean square error shall be dominated by the noise. To study whether CFA truly reduces the variance, we compare the MSE from Equation (2) and Equation (13). We compare the MSE of MCPE, DQN, Dyna-Q, MBCAL (w/o variance reduction), and MBCAL, based on the interactive logs collected in the test round of Batch-RL evaluation. The average MSE is presented in Table 4.

According to the previous theoretical analysis, using value backup of longer horizons suffer from more significant variance. The variance of MCPE is indeed higher than that of DQN and Dyna-Q, as backup of the whole trajectory is used. The MBCAL (w/o variance reduction) has the second-largest variance. It is smaller compared

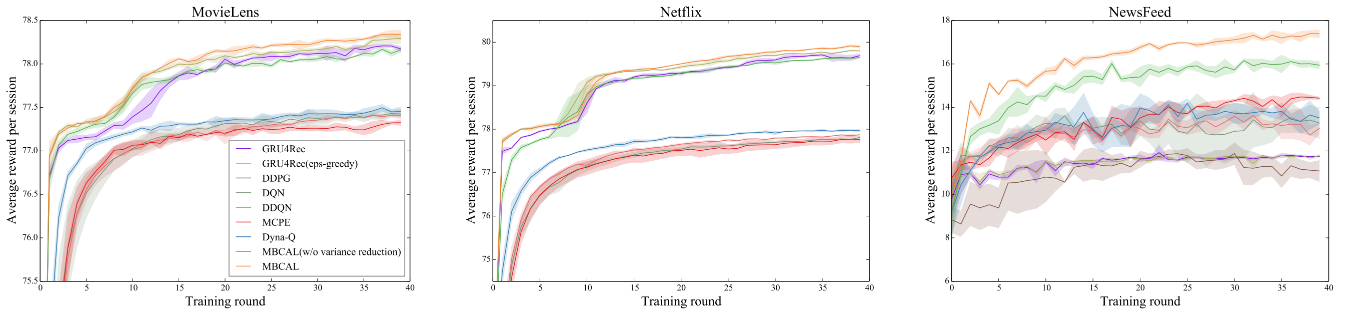


Figure 5: Average reward per session of different algorithms and datasets in Growing Batch-RL evaluation, with the horizontal axis represent the training rounds.

Table 4: The mean square error (MSE) loss of different algorithms in different environments.

Algorithms	**MSE loss**		
	Movielens	***Netflix***	***NewsFeed***
DQN	1.50	1.22	4.29
MCPE	17.1	9.21	46.9
Dyna-Q	0.94	1.04	7.87
MBCAL	**0.004**	**0.009**	**0.07**
MBCAL (w/o variance reduction)	3.45	3.29	3.07

with MCPE because using simulated rollout from the environment model already eliminates part of the noises. The variances of DQN and Dyna-Q are smaller because one-step value backup is employed. Compared with the other methods, MBCAL embraces significantly lower variances, which shows that variance has been reduced as expected.

5 CONCLUSION

To conclude this work, we are focused on the sequential decision-making problems in the recommender systems. To maximize the long-term utility, we propose a sample efficient and variance reduced reinforcement learning method: MBCAL. It involves a masked environment model to capture the instant user behavior, and a future advantage model to capture the future utility. Through counterfactual comparison, MBCAL significantly reduces the variance in learning. Experiments on real-data-driven simulations show that the proposed method transcends the previous ones in both sample efficiency and asymptotic performances. Possible future extensions to this work may be to theoretically calculating the error bound, and to extend the fixed horizon settings to infinite and dynamic horizon recommender systems.

REFERENCES

[1] 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.

[2] 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.

[3] Wei Chu, Lihong Li, Lev Reyzin, and Robert E. Schapire. 2011. Contextual Bandits with Linear Payoff Functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. 208–214. http://proceedings.mlr.press/v15/chu11a/chu11a.pdf

[4] Marc Peter Deisenroth and Carl Edward Rasmussen. 2011. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. 465–472. https://icml.cc/2011/papers/323_icmlpaper.pdf

[5] Christos Dimitrakakis and Michail G. Lagoudakis. 2008. Rollout sampling approximate policy iteration. *Machine Learning* 72, 3 (2008), 157–171. https://doi.org/10.1007/s10994-008-5069-3

[6] 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).

[7] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 1582–1591. http://proceedings.mlr.press/v80/fujimoto18a.html

[8] Stuart Geman, Elie Bienenstock, and René Doursat. 1992. Neural Networks and the Bias/Variance Dilemma. *Neural Computation* 4, 1 (1992), 1–58. https://doi.org/10.1162/neco.1992.4.1.1

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947

[10] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 173–182.

[12] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*. 843–852. https://doi.org/10.1145/3269206.3271761

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1511.06939

[14] 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).

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[16] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer, 45–73.

[17] 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 *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1509.02971

[18] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. 2018. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027* (2018).

[19] Qianlong Liu, Baoliang Cui, Zhongyu Wei, Baolin Peng, Haikuan Huang, Hongbo Deng, Jianye Hao, Xuanjing Huang, and Kam-Fai Wong. 2019. Building Personalized Simulator for Interactive Search. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 5109–5115. https://doi.org/10.24963/ijcai.2019/710

[20] 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.

[21] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*. 1842–1850.

[22] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.

[23] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1506.02438

[24] Richard S. Sutton. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Machine Learning, Proceedings of the Seventh International Conference on Machine Learning, Austin, Texas, USA, June 21-23, 1990*. 216–224. https://doi.org/10.1016/b978-1-55860-141-3.50030-4

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

[26] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 2. MIT press Cambridge.

[27] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. 2018. Deep Reinforcement Learning and the Deadly Triad. *CoRR* abs/1812.02648 (2018). arXiv:1812.02648 http://arxiv.org/abs/1812.02648

[28] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. 2094–2100. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

[29] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[30] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 1995–2003. http://proceedings.mlr.press/v48/wangf16.html

[31] 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).

[32] 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*. ACM, 95–103.

[33] 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*. 1040–1048.

[34] 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*. International World Wide Web Conferences Steering Committee, 167–176.

[35] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation. (2020).