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

Fan Wang†‡, Xiaomin Fang∗, Lihang Liu∗, Hao Tian†, and Zhiming Peng∗
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 978-1-4503-7899-x/20/08...
https://doi.org/10.1145/nnnnnn.nnmmmm

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 cannot 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 $d_1$ in the 3rd step in trajectory $T_a$, which yields 2 “Likes” considering the instant and future utilities. It is hard to judge whether $d_1$ 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 $d_1$ in $T_b$ yields 3 following “Likes”, by which we may judge that $d_1$ 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 $(S, A, R, µ)$. The details are explained as follows:

- **State $s_t \in S$** 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.,

  $$ s_t = (a_1, a_2, ..., a_{t-1}) $$

where we use $a_t$ to represent the pair of exposed item $a_t^o$ and corresponding feedback $b_t^o$, i.e., $a_t = (a_t^o, b_t^o)$. For simple we also use $o_{t:1:t-1}$ to represent the trajectory $a_1, a_2, ..., a_{t-1}$.

- **Action $a_t \in A_t$** denotes the selected item by the agent, with $A_t$ being the candidate set that is passed from the earlier stage of the recommender system.

- **Reward $r_t \in R$** is the utility that we want to maximize. Usually, it depends on the observed user behaviors.

- **Conditional Transition Probabilities $µ(s’ | s, a)$** 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 $b_t \in B$, where $B = \{B_1, B_2, ..., B_n\}$ is the set of candidate user behaviors (e.g., ‘Click’, ‘Skip’, ‘Click and Thumbs Up’). Also, notice that we use the superscript $o$ to denote that an action $a_t$
by minimizing the loss function:

$$L_{\text{MFR}} (\theta) = \frac{1}{|\mathcal{D}|} \sum_{o \in \mathcal{D}} \sum_{t=1}^{T} (\hat{Q}_{\text{target}}(o_{t:t-1}, a_{t}^{\pi}) - \hat{Q}_{\theta}(o_{t:t-1}, a_{t}^{\theta}))^2.$$  

\(\hat{Q}_{\text{target}}\) 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 \(\hat{Q}_{\text{target}} = r_{t} + \gamma \max_{a'} Q_{\theta}(o_{t:t-1}, a').\) \(\theta'\) here is the set of parameters which is periodically copied from \(\theta\) 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 online data. Growing Batch-RL lies somewhere in between the Batch-RL and online learning. The data can be collected 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 \(\pi_{k}\) 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 \(\pi_{k}\) to \(\pi_{k+1}\).

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

### 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 Advantages 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
To eliminate the intractable noises hidden in the feedbacks, we introduce a masking item into the model. The motivation of this is to approximate \( \hat{\mu}(s_t, a_t, s_{t+1}) \) to predict the reward \( r(s_t, a_t) \) to approximate the reward \( r(s_t, a_t) \) to approximate the reward. 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).

\[
\mu(s_{t+1}|s_t, a_t) = \mu(a_t^0|s_t, a_{t-1}^0, a_t^0) = \mu(b_t^0|s_{t-1}, a_{t-1}^0)
\]

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 \( \tilde{\mu} \) and \( \tilde{r} \). We introduce \( f_\phi(o_{t-1}, a_t^0, b_t^0) \) with trainable parameters \( \phi \) to approximate \( \mu(b_t^0|o_{t-1}, a_{t-1}^0) \), which predicts the probability of the next user behavior being \( b_t^0 \). The transition and the reward are then naturally approximated with

\[
\hat{\mu}(s_t, a_t, s_{t+1}) = f_\phi(o_{t-1}, a_t^0, b_t^0)
\]

\[
\tilde{r}(s_t, a_t) = \sum_n \mathcal{R}_n \cdot f_\phi(o_{t-1}, a_t^0, b_t^0)
\]

### 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 \( o \), we denote the trajectory where the actions at positions \( y = \{t_1, t_2, \ldots \} \) are replaced by \( a_M \) as \( M(o, y, a_M) \). For example, suppose \( y = \{2, 3\} \), it gives a masked trajectory of

\[
M(o, \{2, 3\}, a_M) = (a_1^0, b_1^0, a_M, b_2^0, a_M, b_3^0, a_M, b_4^0, \ldots)
\]

where actions \( a_2^0 \) and \( a_3^0 \) in \( o \) are replaced by \( a_M \).

The training is straightforward. We sample random positions \( y^o \) for each trajectory \( o \), such that each position has uniform probability of \( p_{\text{mask}} \) 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 \( D_M = \{M(o, y^o, a_M) | o \in D\} \), we maximize the likelihood, or minimize the negative log-likelihood (NLL).

\[
\mathcal{L}_{\text{MEM}}(\phi) = -\frac{1}{|D_M|} \sum_{o \in D_M} \frac{1}{T} \sum_{t=1}^T \log f_\phi(o_{t-1}, a_t^0, b_t^0).
\]

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 \( o_{t-1} \). As we need to encode \( o_{t-1} \) and \( a_t^0 \) 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 \( b_{t-1}^0 \) and \( a_t^0 \) as input and output the probability of the next possible behavior. An additional \( b_t \) is introduced as the start of the observed user behavior (see Figure 3).

Concretely, the architecture is formulated as follows.

\[
\begin{align*}
\mathcal{L}_{\text{MEM}}^0(u) &= \mathcal{L}_{\text{MEM}}(u) \\
\mathcal{L}_{\text{MEM}}^M(u) &= \mathcal{L}_{\text{MEM}}(u) \\
\mathcal{L}_{\text{MEM}}^T(u) &= \mathcal{L}_{\text{MEM}}(u) \\
\end{align*}
\]

Here \( \mathcal{L}_{\text{MEM}}(u) \) denotes a representation layer; \( \mathcal{L}_{\text{MEM}}^M(u) \) denotes a concat operation and MLP denotes a multilayer perceptron; \( \mathcal{L}_{\text{MEM}}^T(u) \) 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).

\[
A^\text{CFA}(o, t) = A^\text{future}(o, t) - A^\text{future}(M(o, \{t\}, a_M), t).
\]

Finally, we introduce the Future Advantage Model (FAM) denoted with \( \mathcal{L}_{\text{FAM}}(\eta) \), with trainable parameters \( \eta \). To train FAM, we minimize the mean square error shown in Equation (13).

\[
\mathcal{L}_{\text{FAM}}(\eta) = \frac{1}{|D|} \sum_{(o, t) \in D} \frac{1}{T} \sum_{t=1}^T (\mathcal{L}_{\text{FAM}}^\phi(o_{t-1}, a_t^0))^2.
\]

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
Future Utility
Predicted Behaviors
\{ 𝑏 \}
\{ 𝑜 \}

Actions
\{ 𝑎 \}

Backward
Forward
Item
User behavior
Masking Item

Figure 3: Illustration of the MBCAL architectures.

advantage) instead of a distribution, shown as follows:

\[
\begin{align*}
h_0^{FAM} &= \text{Emb}(a), \\
x_t^{FAM} &= \text{Concat}(\text{Emb}(h_{t-1}), \text{Emb}(a_t)), \\
h_t^{FAM} &= \text{GRU}(h_{t-1}^{FAM}, x_t^{FAM}), \\
g_t = \text{MLP}(h_t^{FAM}).
\end{align*}
\]

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 \( o_{[1:T]} \), we pick the next action according to Equation (18).

\[
a_t^* = \arg \max_a [r_o(a_{[1:t-1]}, a) + g_\pi(o_{[1:t-1]}, a)].
\]

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

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

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 \( \pi \) 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 of trajectory (that is \( o_{[t+1:T]} \)) 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\[1\], 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 \( B = \{0, 1, 2, 3, 4, 5\} \) corresponds to the star ratings, with the reward to be \( R = \{0, 1, 2, 3, 4, 5\} \). 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).

1http://files.grouplens.org/datasets/movielens/ml-20m-README.html
2https://www.kaggle.com/netflix-ncf/netflix-prize-data
3Data 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.

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.

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

#### 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 $\epsilon$-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.

For each experiment, we report the average reward per session in the test round, defined by $1/|\mathcal{D}_{test}| \cdot \sum_{a \in \mathcal{D}_{test}} \sum_{t=1}^{T} r_t$, 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 are 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 (\(\epsilon\)-greedy)) to MBR (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 \(\epsilon\)-greedy version of NN models (GRU4Rec (\(\epsilon\)-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 (\(\epsilon\)-greedy)**: It applies the \(\epsilon\)-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 $Q_{target} = \sum_{t} r_t$. 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 $p_{\text{mask}} = 0.20$ 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.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Average reward per session</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Movielens</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>77.93 ± 0.06</td>
</tr>
<tr>
<td>DDPG</td>
<td>70.99 ± 0.70</td>
</tr>
<tr>
<td>DQN</td>
<td>77.27 ± 0.06</td>
</tr>
<tr>
<td>DDQN</td>
<td>77.23 ± 0.02</td>
</tr>
<tr>
<td>MCPE</td>
<td>77.20 ± 0.10</td>
</tr>
<tr>
<td>Dyna-Q</td>
<td>77.25 ± 0.05</td>
</tr>
<tr>
<td>MBCAL</td>
<td>78.02 ± 0.03</td>
</tr>
<tr>
<td>MBCAL (w/o variance reduction)</td>
<td>77.70 ± 0.04</td>
</tr>
</tbody>
</table>

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
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


