Enhanced Reinforcement Learning for Recommender Systems: Maximizing Sample Efficiency and Minimizing Variance

Abstract

Optimizing long-term user satisfaction in recommender systems, such as news feeds, is crucial during continuous user-system interactions. Reinforcement learning has shown promise in addressing this challenge. However, practical hurdles like low sample efficiency, potential risks, and high variance hinder the implementation of deep reinforcement learning in online systems. We introduce a new reinforcement learning approach called model-based counterfactual advantage learning (MBCAL) to tackle these challenges. MBCAL leverages the unique aspects of recommender systems and incorporates concepts from model-based reinforcement learning to enhance sample efficiency. It consists of two main parts: an environment model that predicts immediate user behavior sequentially and a future advantage model that forecasts future utility. Counterfactual comparisons from the environment model are used to mitigate the excessive variance when training the future advantage model. Consequently, MBCAL achieves high sample efficiency and significantly reduced variance, while utilizing existing user logs to avoid starting from scratch. Despite its capabilities, MBCAL maintains a relatively low implementation cost, making it suitable for real-world systems. The proposed method surpasses other supervised learning and RL-based methods in both sample efficiency and overall performance, as demonstrated through theoretical analysis and extensive experiments.

1 Introduction

Recommender systems are essential for delivering personalized content and improving the efficiency of information retrieval. Modern recommender systems, like news feeds, must consider multiple user-system interactions within a single session. The content recommended in past interactions can influence future user behavior. For example, exploring new topics might pique a user's interest in related areas, while repeatedly showing similar content could lead to a rapid decline in user engagement. Traditional recommender systems rely on collaborative filtering or neural networks to predict immediate user actions, such as clicks. However, solely focusing on immediate actions can result in issues like recommendation redundancy, ultimately harming the user's long-term experience.

Recently, deep reinforcement learning (Deep RL) has gained attention for its potential in recommender systems. Deep RL models user-system interactions as Markov Decision Processes (MDPs). Many studies in this field focus on model-free reinforcement learning (MFRL) methods. However, challenges persist, including the substantial data consumption during training, also known as low sample efficiency. Another challenge is the practical risks associated with implementing MFRL. On-policy RL struggles to utilize off-policy user logs, leading to difficulties in online infrastructure and initial performance. Conversely, off-policy RL faces the risk of non-convergence when combined with function approximation and offline training.

Model-based RL (MBRL) offers an alternative with improved sample efficiency and reduced practical risks. MBRL employs an environment model to predict immediate feedback and state transitions, along with a planning module to find an optimal trajectory. However, MBRL can be computationally intensive during inference. Planning is often infeasible in multi-stage retrieval frameworks commonly used in modern recommender systems. These systems generate candidate sets of items in earlier stages for subsequent stages, making it impossible to predetermine candidates. To address these issues, Dyna algorithms have been proposed for recommender systems. The Dyna algorithm accelerates convergence by generating virtual interactions using the environment model. However, this faster convergence comes at the cost of reduced asymptotic performance due to error accumulation from virtual interactions.

Another significant challenge in deploying RL is the excessive variance of gradients during optimization. This variance can stem from stochastic transitions, noisy rewards, and stochastic policies. Longer horizons tend to exacerbate the variance, significantly slowing down convergence and introducing instability. Prior research has shown that using an advantage function instead of a value function can reduce variance and improve performance. However, these proposals primarily target MFRL, and variance reduction in MBRL remains largely unexplored.

In recommender systems, variance can arise from various factors. First, there is substantial noise in observed user feedback. Some users may be more inclined to provide positive or negative feedback than others. Even individual users may exhibit different

behaviors at different times of the day. Second, for stochastic policies, resampling trajectories from any state can lead to varying long-term returns. While a large amount of data can mitigate the impact of variance, it still negatively affects performance due to data sparsity for specific users and items.

To address variance reduction, our work introduces the concept of comparing an observed trajectory with a counterfactual trajectory. This counterfactual trajectory shares all contexts with the original, including the user, historical interactions, and follow-up items, except for the current action being replaced. By comparing these trajectories, we can make more informed judgments about the advantage of taking a specific action. While finding such counterfactual records in user logs is impossible, we can leverage the environment model to simulate future rollouts and generate these trajectories.

Building on this idea, we propose a novel MBRL solution for recommender systems called Model-based Counterfactual Advantage Learning (MBCAL). MBCAL decomposes overall utility into immediate utility (rewards from the current step) and future utility (rewards from future steps). The environment model naturally predicts immediate utility, while future utility is approximated through simulated rollout. To further reduce variance in future utilities, we perform two comparative simulated rollouts. We introduce a masking item to the environment model, enabling us to generate simulated rollouts by masking the action of interest. We then calculate the counterfactual future advantage (CFA) as the difference in future utility with and without masking. Finally, we introduce the future advantage model to approximate the CFA.

We conducted experiments using three real-world datasets and compared our method with supervised learning, MFRL, and MBRL approaches. We also focused on Batch-RL and Growing Batch-RL settings, which are more aligned with practical infrastructures. The experimental results demonstrate the superiority of our proposed method.

2 Methodology

The core concept of MBCAL is illustrated by employing two models: the Masked Environment Model (MEM) and the Future Advantage Model (FAM). These models are designed to estimate immediate user behavior and future advantages, respectively. The training process begins with optimizing the environment model to predict user behaviors, incorporating a masking item into the model. Using the MEM, we compute the Counterfactual Future Advantage (CFA) by contrasting the future utility derived from masking the action against not masking it. The CFA then serves as the target for training the FAM. During inference, we combine both models to select actions.

We first formalize the environment model and then detail the MEM, FAM, and the overall learning process. Following this, we provide a theoretical analysis of the proposed method.

2.1 Environment Modeling

Typically, an environment model predicts transitions and rewards separately. Here, we use approximations for the transition probability and the reward function. Specifically, to formulate the environment model in a recommender system context, we can express the transition probability as the probability of observing the next user behavior given the past trajectory and the current action. This means that predicting the transition simplifies to predicting the immediate user behavior. Since the reward also depends solely on user behavior, a single model can replace the separate transition and reward approximations. We introduce a function with trainable parameters to approximate the probability of the next user behavior. The transition and reward are then approximated using this function.

2.2 Masked Environment Model

To mitigate the intractable noise in user feedback, we introduce a masking item into the model. This allows us to create a counterfactual comparison to the current trajectory, answering the question: "What would the future behavior be if this action were not taken?" We introduce a virtual item represented by a trainable embedding vector. Given an observation trajectory, we denote the trajectory where actions at specific positions are replaced by this virtual item as a masked trajectory.

Training is straightforward. We sample random positions for each trajectory, replacing each position with a uniform probability. The MEM aims to recover the user behavior as closely as possible when some items are masked. Using the collected masked trajectories, we maximize the likelihood or minimize the negative log-likelihood (NLL).

To model sequential observations, the MEM's architecture follows that of session-based recurrent recommender systems. We use a Gated Neural Network to encode the trajectory. Since we need to encode both the trajectory and the current action, we concatenate the input in a staggered manner. For each step, the model takes the previous behavior and the current action as input and outputs the probability of the next possible behavior. An additional start symbol is introduced as the beginning of the observed user behavior. The architecture is formulated as follows: a representation layer, a concatenation operation, a multilayer perceptron, and a Gated Recurrent Unit.

2.3 Counterfactual Future Advantage

Using the Masked Environment Model (MEM), we can estimate the difference in future utilities between the original trajectory and its counterfactual counterpart, which we term the Counterfactual Future Advantage (CFA). Given a trained MEM, we first define the Simulated Future Reward (SFR) for an observed trajectory at a specific time step. We then calculate the CFA by subtracting the SFR of the counterfactual comparison from the original one. Finally, we introduce the Future Advantage Model (FAM), with its own set of trainable parameters, to approximate this CFA. To train the FAM, we minimize the mean square error.

The FAM uses a similar neural architecture to the MEM, except for the final layer, but with different parameters. Instead of predicting a distribution, the FAM's last layer predicts a scalar value representing the advantage.

2.4 Summary of MBCAL

For inference, we select the item (action) based on both the MEM and FAM. Formally, given user information and the observation trajectory, we choose the next action by maximizing the sum of the immediate reward predicted by the MEM and the future advantage predicted by the FAM. To avoid local optima in policy improvement, we use an ε-greedy strategy. With probability ε, we select a random action; otherwise, we select the action that maximizes the combined reward and advantage.

MBCAL aligns well with the Growing Batch-RL settings. The algorithm involves iterative data collection and policy updates. Although we use the term "policy," we do not require an explicit policy formulation, unlike common policy gradient methods, which are often challenging to define in many recommender systems.

The variance reduction in MBCAL is primarily achieved through the subtraction in the CFA calculation, which eliminates noise from user feedback and other sources. While we borrow ideas from the advantage function concept, our CFA differs in that we do not resample the trajectory but keep the remaining part unchanged. Although this could introduce bias in many MDP problems, we argue that recommender systems exhibit weaker correlations between sequential decisions compared to other domains (e.g., robot or game control). Additionally, since the FAM averages the CFA across different trajectories, the bias becomes negligible compared to the benefits of variance reduction.

3 Experiments

3.1 Datasets

Evaluating RL-based recommender systems is challenging. The most reliable metric involves online A/B tests, but these are often too costly and risky for comparing all baselines in an online system. Offline evaluation of long-term utility using user logs is difficult because we lack feedback for actions not present in the log. To thoroughly assess the performance of the proposed systems, we follow previous works and construct simulators. However, instead of synthetic simulators, we use real-data-driven simulators. The datasets used include MovieLens, Netflix Prize, and NewsFeed.

- MovieLens: This dataset contains 5-star rating activities from MovieLens. User behavior corresponds to star ratings, with rewards matching these ratings. There are three types of features: movie-id, movie-genre, and movie-tag.
- Netflix Prize: This dataset consists of 5-star ratings from Netflix. Rewards follow the same setup as MovieLens. It includes only one type of feature: movie-id.
- NewsFeed: This dataset is collected from a real online news recommendation system. We focus on predicting the dwelling time on clicked news, partitioned into 12 levels, each corresponding to a different user behavior. Rewards range from 1 to 12. There are seven types of features: news-id, news-tag, news-title, news-category, news-topics, news-type, and news-source.

3.2 Experimental Settings

To ensure a fair evaluation, it is crucial to prevent the agent in the evaluated system from exploiting the simulator. We implement two specific settings in the evaluation process. First, all agents are restricted to using only a subset of features, while the simulator uses the full feature set. In MovieLens and Netflix, agents use only the movie-id feature. In NewsFeed, agents use four out of seven features (news-id, category, news-type, and news-source). Second, we intentionally set the model architecture of the simulator to differ from that of the agents. We use LSTM units for the simulators, while agents use GRU units.

To gauge the simulator's accuracy, we report micro-F1, weighted-F1, and RMSE for user behavior classification. The properties of the datasets and simulators are detailed in Table 2. For the NewsFeed dataset, we also analyzed over 400 historical A-B test records. The correlation between our simulator's predictions of long-term rewards (e.g., total clicks or session dwelling time) and the actual outcomes is above 0.90.

3.2.1 Evaluation Settings

The evaluation process consists of two types of iterations: training rounds and test rounds. During a training round, the agent generates actions using an ε-greedy policy ($\varepsilon = 0.1$ for all experiments) and updates its policy based on feedback from the simulator. In the test round, the agent uses a greedy policy, and the generated data is not used for training. Each session in both training and test rounds involves 20 steps of interaction between the simulator and the agent. Each round includes 256,000 sessions.

For each experiment, we report the average reward per session in the test round, calculated as the sum of rewards over all sessions in the test round divided by the number of sessions. Each experiment is repeated three times with different random seeds, and we report the mean and variance of the scores. We simulate both Batch RL and Growing Batch-RL evaluations separately. In Batch RL evaluation, the agent trains only on static user logs and interacts with the simulator during testing. In Growing Batch RL evaluation, the agent interacts with the simulator during both training and test rounds, with the training round repeating up to 40 times.

3.3 Methods for Comparison

We compare various methods, including Supervised Learning (GRU4Rec), bandits (GRU4Rec (ε-greedy)), MFRL (MCPE, DQN, DDQN, and DDPG), and MBRL (Dyna-Q). For bandits, LinUCB is a common baseline, but it performs poorly in our environments due to the limited representational power of linear models. Therefore, we use the ε-greedy version of NN models (GRU4Rec (ε-greedy)) instead of LinUCB.

The methods for comparison are:

- GRU4Rec: Uses GRU to encode interactive history to predict immediate user behavior, with an architecture equivalent to the environment model. We use entropy loss in GRU4Rec.
- GRU4Rec (ε-greedy): Applies ε-greedy item selection in GRU4Rec during training rounds.
- DQN: A classic off-policy learning algorithm. We use GRU for state representation to ensure fair comparison, similar to GRU4Rec and our method.
- DDQN: Double DQN, which uses a different action selection for value backup to avoid value overestimation in off-policy learning. The model architecture is equivalent to GRU4Rec.
- DDPG: Deep Deterministic Policy Gradient, an off-policy learning algorithm for continuous action spaces. The inferred action selects the nearest neighbor item for display. We use the same neural structure as GRU4Rec for both actor and critic networks.
- MCPE: Monte Carlo Policy Evaluation, a straightforward value iteration algorithm using the whole trajectory for value backup. The model architecture is the same as other baselines.
- Dyna-Q: An MBRL method that augments DQN with imagined rollouts from an environment model. The ratio of imagined rollouts to real trajectories is 1:1.
- MBCAL: The full version of our proposed method.
- MBCAL (w/o variance reduction): An ablated version of MBCAL where we use SFR instead of CFA as the label for FAM.

All parameters are optimized using the Adam optimizer with a learning rate of 10^{-3}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The discount factor for long-term rewards is $\gamma = 0.95$. Embedding sizes for item-id and other id-type features are set to 32. The hidden size for MLP is 32. For training MEM in MBCAL, we use $p_{mask} = 0.20$ to generate masked trajectories. In DDPG, we use a 4-dimensional action space due to poor performance with higher dimensions, and an additional layer maps item representations to this 4-dimensional space.

3.4 Experimental Results

3.4.1 Results of Batch-RL Evaluation

The results of the Batch-RL evaluation are presented in Table 3. We evaluate the reward per session based on the rewards generated by the simulator. The results indicate that MFRL methods cannot outperform MBRL methods across all three environments. Due to its sample inefficiency, MFRL tends to exhibit poor initial performance. Notably, DDPG demonstrates the weakest performance across all environments. Upon closer examination of the value functions in DDPG, we observed significant overestimation compared to other MFRL methods. This overestimation likely arises from value backups based on continuous actions that may not correspond to actual items.

As anticipated, MBCAL outperforms all other tested systems by substantial margins, showcasing its sample efficiency. However, the advantage of our method over the supervised learning method is less pronounced in the MovieLens and Netflix datasets compared to NewsFeed. This suggests that long-term rewards play a more significant role in the NewsFeed environment.

Furthermore, while learning to predict long-term utility requires more data than immediate rewards, the dominance of RL is not yet fully apparent in Batch-RL settings. Nevertheless, it is crucial that MBCAL's initial performance is already state-of-the-art, underscoring its low risk and high sample efficiency.

Table 1: Average reward per session of different algorithms and datasets in Batch-RL evaluation.

Algorithms	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

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

3.4.2 Results of Growing Batch-RL Evaluation

In all environments, GRU4Rec(ε-greedy) slightly outperforms the purely supervised GRU4Rec, highlighting the advantages of exploration in online systems. The performance of DDPG remains surprisingly poor across all three environments.

With the aid of the environment model, Dyna-Q initially gains some advantages but gradually diminishes as learning progresses. This observation aligns with expectations since the virtual experience loses its benefits as sufficient real user feedback accumulates. MBCAL maintains its performance lead over other methods in all environments. Even in Netflix and MovieLens, where other RL-based systems fail to outperform traditional GRU4Rec, MBCAL achieves a considerable margin. In NewsFeed, where long-term rewards are more critical, MBCAL further extends its lead.

MCPE, DQN, DDQN, and Dyna-Q lag behind other methods, including supervised learning baselines in MovieLens and Netflix, but not in NewsFeed. Investigating further, we modified GRU4Rec to output the immediate reward instead of user behavior classification, turning the task into regression and replacing entropy loss with mean square error loss. This change resulted in a significant performance drop in GRU4Rec, aligning more closely with the NewsFeed results. These findings suggest that classification and entropy loss benefit the system more than regression, and that user behavior contains richer information than rewards, giving MBRL an edge over MFRL.

3.4.3 Analysis of the variance

The critical aspect of MBCAL is variance reduction through counterfactual comparisons. Previous research indicates that the mean square error (MSE) in a well-trained model comprises model bias and label variance (noise). Since we use equivalent neural architectures across all comparison methods, they share the same model bias. Thus, the MSE is primarily influenced by noise. To assess whether CFA effectively reduces variance, we compare the MSE from the value backup equation and the CFA equation. We analyze the MSE of MCPE, DQN, Dyna-Q, MBCAL (w/o variance reduction), and MBCAL using interactive logs from the test round of Batch-RL evaluation.

Table 3: The mean square error (MSE) loss of different algorithms in different environments.

Algorithms	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

The average MSE is presented in Table 4. Consistent with theoretical analysis, longer horizon value backups exhibit higher variance. MCPE has a higher variance than DQN and Dyna-Q due to using the entire trajectory for backup. MBCAL (w/o variance reduction) has the second-largest variance, lower than MCPE because the environment model's simulated rollout partially eliminates noise. DQN and Dyna-Q have smaller variances due to one-step value backup. Compared to other methods, MBCAL shows significantly lower variance, confirming the expected variance reduction.

4 Conclusion

In conclusion, our work focuses on sequential decision-making problems in recommender systems. To maximize long-term utility, we propose a sample-efficient and variance-reduced reinforcement learning method called MBCAL. This method incorporates a masked environment model to capture immediate user behavior and a future advantage model to predict future utility. By employing counterfactual comparisons, MBCAL significantly reduces learning variance. Experiments conducted on real-data-driven simulations demonstrate that our proposed method surpasses existing approaches in both sample efficiency and asymptotic performance. Future work could involve theoretically calculating the error bound and extending the fixed horizon settings to infinite and dynamic horizon recommender systems.