The Adaptive Q-Network for Recommendation Tasks with Dynamic Item Space

Jianxiang Zhu[1], **Dandan Lai**[1], **Zhongcui Ma**[1], **Yaxin Peng**[1,2,*]

[1]the Department of Mathematics, College of Sciences, Shanghai University, Shanghai 200444, China
[2]the School of Future Technology, Shanghai University, Shanghai 200444, China
zjx0828@shu.edu.cn, 2328140942@shu.edu.cn, mzc1217@shu.edu.cn, yaxin.peng@shu.edu.cn

Abstract

Reinforcement learning (RL) algorithms can improve recommendation performance by capturing long-term user-system interaction. However, current RL-based recommendation tasks seldom consider the dynamism of the environment, and standard RL algorithms are ineffective in recommending items dynamically. In addressing these issues, we design a novel task termed dynamic recommendation, which takes the emergence of real-world recommendable items into consideration. Meanwhile, we propose Adaptive Q-Network (AdaQN) to tackle the dynamic recommendation task. Firstly, AdaQN predicts the value of different action characteristics, particularly during the testing phase, which can capture emerging new action characteristics. The above procedure helps AdaQN in effectively adapting to the dynamic action space. Secondly, AdaQN establishes a stable mapping that projects the discrete action space onto a continuous characteristic space. Finally, AdaQN employs a lightweight Q-Network design, which mitigates the complexity of the optimization process. Extensive experiments demonstrate that our approach has achieved state-of-the-art performance in the dynamic recommendation task.

Introduction

Recommender System (RS) (Jiang et al. 2022; Guo et al. 2017) typically relies on supervised learning to understand users' interests and predict their immediate reactions. Existing RS methods provide strong supervision signals that help the model acquire preferences accurately and provide reliable predictions on known data. They have been widely applied in fields such as social networks (Qin et al. 2020), online advertising (Hansen et al. 2020), and e-commerce (Smith and Linden 2017; Rohde et al. 2018). However, these methods struggle to effectively capture long-term impacts and may not fully consider the complex interactions between users and recommendation models, which can result in suboptimal recommendation outcomes.

Recently, reinforcement learning (RL) has been applied to RS owing to its capabilities in long-term planning (Lee et al. 2022; Yu and Zhang 2023; Wagenmaker and Pacchiano 2023). Through a long-term and continuous learning process, RL helps RS handle complex user interactions more effectively and give feedback, achieving self-optimization of personalized recommendations. This means that methods based on RL can further enhance the long-term performance of traditional recommendation methods (Ge et al. 2021).

However, the current application of RL in recommendation tasks overlooks the dynamic of the environment, maintaining a constant action space that represents an overly idealized scenario. For example, users are uploading new short videos created by themselves all the time, but it is unacceptable for the model to constantly recommend outdated videos. Therefore, the pool of recommendable items should be subject to change in the recommendation contexts.

To meet this demand, we have introduced a novel task called dynamic recommendation, which requires a dynamic action space to effectively simulate real-world recommendation environments. Specifically, new items are continually added to the recommendation pool, while obsolete ones are removed to ensure that the recommendations remain up-to-date. Based on this concept, we update the action space during the testing phase to recommend the most recent items. Besides, the items in the training and testing phases are mutually exclusive.

Nevertheless, standard RL is incapable of addressing the dynamic recommendation task. The current algorithms based on RL are unable to predict the level of user satisfaction with newly emerging items. One of the problems stems from methods based on discrete action space (Zhao et al. 2018; Zheng et al. 2018) neglect the connections between items for treating each item as an independent action. Additionally, some approaches based on continuous action space (Liu et al. 2023; Zhao et al. 2018) look for similar items in the test environment as those in the training one, overlooking the long-term impact of new items on the test environment. Overall, the above oversights will lead to poor generalization of the traditional RL algorithms for recommendation.

In this paper, we propose an original algorithm to deal with the dynamic recommendation task, termed Adaptive Q-Network (AdaQN). Specifically, AdaQN first learns a value function based on states and item characteristics, as well as effectively predicts the values of new items during the testing phase, which not only aids our algorithm in completely utilizing the relationships between items but also enhances its adaptability to the dynamic item space. Subsequently, we discuss how to establish a mapping from the discrete

*corresponding author

item space to the continuous characteristic space, which helps AdaQN to be free from exploring every new item in the discrete item space. Besides, our research indicates that establishing a stable mapping is crucial, endowing our algorithm with excellent predictive capabilities by learning a reliable value function. Finally, we design a lightweight Q-Network called Characteristic-based Efficient Q-Network (CEQN) that better adapts to AdaQN, addressing the issues of redundant parameters and optimization challenges associated with standard Q-Networks in our approach.

Our contributions can be summarized as follows.

- We design a dynamic recommendation task to model the evolving process of constantly shifting recommendable items in real-world applications.
- We develop AdaQN to predict the long-term rewards of new items that are received from users through utilizing a value function based on the item characteristic space.
- We establish a mapping from a discrete action space to a continuous characteristic space, which aids in learning a stable value function.
- We design the CEQN to reduce the number of parameters, enhancing compatibility with the AdaQN algorithm.

Preliminaries

RS needs to cope with dynamic environments by estimating rapidly changing user preferences and proactively recommending items to users. Typically, let $\mathcal{U}$ be a set of users of cardinality $|\mathcal{U}|$ and $\mathcal{I}$ be a set of items of cardinality $|\mathcal{I}|$. For each user $u \in \mathcal{U}$, we observe a sequence of user actions $\mathbf{X}^u = \{x_1^u, x_2^u, \cdots, x_t^u\}$ with item $x_t^u \in \mathcal{I}$, i.e., each event in a user sequence comes from the item set. The decisions that users make are referred to as interactions with items. Let Fe be feedback of users (i.e. click or rating behavior), RS maintains a recommendation policy π_t^u, which will be updated based on the feedback $fe_i^u \in Fe$ received during the interaction of the item $i \in \mathcal{I}$ at the timestamp t. The introduction of RL has promoted the breakthrough of RS. RL-based RS consists of three building blocks: environment construction, state representation and recommendation policy learning, where recommendation policy learning is a key component in understanding and predicting the future behavior of users. In RL-based RS, the interactive recommendation process is formulated as a Markov decision process (MDP) $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where

- State space $\mathcal{S}$, $s_t \in \mathcal{S}$ represents the current state of user at timestamp t. In recommendation scenarios, user characteristics and user history are usually modeled as states, where user history refers to the actions and corresponding feedback of each step in the previous interaction process.
- Action space $\mathcal{A}$, $a_t \in \mathcal{A}$ represents the action that RL agent take at timestamp t. In recommendation scenarios, the recommended item i is generally chosen as the action a.
- State transition probability function $\mathcal{P}$, $P(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ represents the transition probability from (s, a) to s'. In RS, the transition probability refers to users behavior probability.
- Reward function r, $r(s, a)$ denotes the reward by taking action a at state s. Rewards in recommendation scenarios are usually set to feedback signals provided by users, such as click behavior, favorite behavior, or dwell time.
- The discount factor γ, $\gamma \in [0, 1]$ is used to balance between future and immediate. The agent only focuses on the immediate reward when $\gamma = 0$ and takes into account all the (immediate and future) rewards otherwise.

Given a set of users $\mathcal{U} = \{u_1, u_2, \cdots\}$ and a set of items $\mathcal{I} = \{i_1, i_2, \cdots\}$, the system first recommends item i to user u and get feedback fe_i^u. The system incorporates the feedback to improve future recommendations and determines an optimal policy π^* regarding which item to recommend to the user to achieve positive feedback. The MDP modeling treats the user as the environment and the system as the agent. Typically, it learns a recommendation policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize the cumulative reward G_T over the whole interactive recommendation process:

$$\max_{\pi} \mathbb{E}[G_T] = \max_{\pi} \mathbb{E}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)].$$

Given the current state s_t, we can use Q-value to evaluate each action at each time step t:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[\sum_{j=0}^{T-t} \gamma^j r_{t+j}] = \mathbb{E}[r_t + \sum_{j=1}^{T-t} \gamma^j r_{t+j}].$$

According to the optimal Bellman equation, the optimal Q^* is the maximum expected reward as follows:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a_t} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t].$$

Related Work

RL algorithms aim to take actions within an environment to maximize cumulative rewards. Standard RL can be categorized into three types: value-based methods (Mnih et al. 2013), policy-based methods (Williams 1992), and actor-critic-based methods (Schulman et al. 2017). Numerous studies have integrated RL algorithms into RS to enhance recommendation performance, given their capability of capturing the long-term interactions between users and the system. Zheng et al. (2018) pioneered the application of the Dueling DQN algorithm for the purpose of news recommendation. Huang et al. (2021) used the REINFORCE algorithm (Williams 1992) and modeled the sequential interactions between users and the recommender system by adapting a recurrent neural network. Chen et al. (2019) applied the REINFORCE algorithm to a production-scale RS with an action space in the magnitude of millions.

In contrast to the discrete action-based methods delineated above, certain methods utilize RL algorithms based on continuous action spaces, which typically determine the ranking scores of items by computing the inner product of the generated action vectors and item embedding. DeepPage (Zhao et al. 2018) was designed to optimize the presentation

of one-page items in real-time, leveraging user feedback to ensure an appropriate display. Cai et al. (2023) used RLUR to optimize user retention of the short video recommender system, significantly increasing the number of daily active users. HAC (Liu et al. 2023) employed a latent hyper-action mechanism to recommend a list of items from a large item set.

However, the majority of the works fail to provide a detailed discussion of the dynamic action space. Additionally, the generated action vectors are inclined to bias the action space during the training phase and are incapable of adapting to the constantly changing action space. Based on this situation, our research not only designs dynamic recommendation tasks for a detailed discussion of the dynamic action space but also formulates the AdaQN algorithm to accommodate the shifting action space.

Method

In this section, we provide an extensive elaboration on the AdaQN algorithm. Prior to this, we initially present the dynamic recommendation task that we have proposed.

Predictive Performance of RL-Based RS: Dynamic Recommendation Task

Retraining with every new batch of data is cost-intensive in real recommendation scenarios. Therefore, there is a need for the recommendation model to possess predictive capabilities, which include the ability to provide effective recommendation policies for items without interaction data. Based on this requirement, we designed the following task to estimate the predictive performance of the current model on unknown items.

We select the top $a\%$ of items by their order of appearance to constitute our training environment and denoted as set I_{train}. The remaining items are designated as the test environment I_{test}. Notably, there is no overlap between set I_{train} and set I_{test}, ensuring that $I_{train} \cap I_{test} = \emptyset$ (the case where the two overlap in the Ablation Studies on Task Setting). This approach simulates the dynamic real-world recommendation landscape, where fresh items are continually introduced and outdated ones are naturally phased out. Specifically, we conduct the experiment using the publicly available dataset KuaiRec and MovieLens.

Standard recommendation RL ultimately aims to learn a policy, denoted as π, which dictates the selection of items to make recommendations for users based on the current state. In this task, the algorithm needs to train a policy π within the training environment, denoted as $\pi_{train}(i|s), i \in I_{train}$.

However, due to the alteration of the recommended items set in the test environment, the algorithm must adopt a different strategy within the test environment, represented by $\pi_{test}(i|s), i \in I_{test}$. Therefore, the policy expression of standard value-based RL on the test item set as an example:

$$\pi_{test}(s) = \arg\max_{i \in I_{test}} Q^{\pi_{test}}(s, i).$$

Due to I_{train} and I_{test} are two discrete sets and $I_{train} = \cap I_{test} = \emptyset$, it is difficult to directly estimate $Q^{\pi_{test}}(s, i_{test})$ through $Q^{\pi_{train}}(s, i_{train})$. Besides, we believe that the core reason why traditional recommendation RL algorithms cannot effectively complete this task is that they ignore the similarity between items. This not only leads to the inability of algorithms to recommend new items but also prevents mutual assistance in recommendations among items.

Adaptive Q-Network

To address this issue, we have developed a new value-based RL algorithm called Adaptive Q-Network (AdaQN), which does not interact directly with the objects but rather with their characteristics.

First, we have established a mapping f from the item space I to the characteristic space V:

$$\begin{aligned} &v = f(i), \quad i \in I, v \in V, \\ &I = \{i_1, i_2, \cdots, i_{|I|}\}, \\ &V \in \mathbb{R}^n. \end{aligned}$$

Through this approach, our algorithm maps the discrete action space onto a continuous characteristic space. The detailed formulation of the mapping function f will be introduced in Section The selection of mapping f.

Next, we project both the training item set I_{train} and the test item set I_{test} onto the continuous characteristic space V through the mapping function f, obtaining the training item characteristic set V_{train} and the test item characteristic set V_{test}. It is assumed that $V_{train} \subset V$, $V_{test} \subset V$.

Specifically, the core of our method is to replace the original state-item value function $Q^\pi(s, i)$, i.e. $Q^\pi(s, a)$, by a state-characteristic value function $Q^\pi(s, v)$, which can be written as:

$$\begin{aligned} Q^\pi(s_t, v_t) &= \mathbb{E}[\sum_{j=0}^{T-t} \gamma^j r(s_{t+j}, v_{t+j})] \\ &= \mathbb{E}[r(s_t, v_t) + \sum_{j=1}^{T-t} \gamma^j r(s_{t+j}, v_{t+j})]. \end{aligned}$$

The subscript 't' represents the moment in time 't' and $r(s_t, v_t) = r(s_t, i_t)$ if $f(i_t) = v_t$.

By this way, AdaQN has effectively learned the relationships between items that similar items have similar characteristics. Consequently, their Q-values are very likely proximate in magnitude under the same state, providing the possibility for the generalization of our algorithm.

Therefore, we can fit a value function $Q^{\pi_{train}}(s, v), v \in V$ over the entire characteristic space V by interacting with the characteristic set of the training item set V_{train}. However, it is unable to directly ascertain the value of $Q^{\pi_{test}}(s, v)$ as the items within the test environment remain unknown. Instead, we can make the optimal trajectory distribution in the test environment approximate the optimal trajectory distribution in the training environment.

In Fig. 1, we observed that the trajectory in the test environment closely matches the one in the training environment for most timestamps. However, the trajectories diverge at certain specific timestamps due to differences in the action spaces between the testing and training phases.

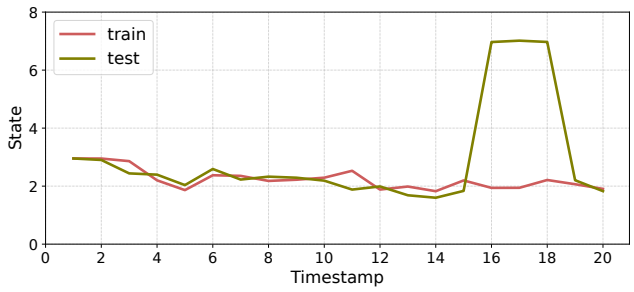


Figure 1: The trajectory plot displays the L_1 norm of the state on the vertical axis and the timestamp on the horizontal axis. A trajectory is randomly sampled from the training environment, and its approximation is observed in the test environment.

The entire update process can be formulated as:

$$\begin{aligned} & Q^{\pi_{test}}(s,v), v \in V \\ \approx & Q^{\pi_{train}}(s,v), v \in V \\ \approx & Q^{\pi_{train}}(s,v), v \in V_{train} \\ = & r(s,v) + \gamma \max_{v' \in V_{train}} Q^{\pi_{train}}(s',v'), v \in V_{train}. \end{aligned}$$

Simultaneously, our policy during testing can be formulated as

$$\pi_{test}(s) = \arg\max_{i \in I_{test}} Q^{\pi_{test}}(s, v = f(i)). \quad (1)$$

It is worth noting that our algorithm is also applicable to standard recommendation RL tasks, where the training set I_{train} is equivalent to the test set I_{test}. We attribute this to the continuous characteristic space effectively alleviating the issue of low learning efficiency caused by the excessively large discrete item space.

The Selection of Mapping f Further analysis of the mapping function f is provided in this part. Every item possesses inherent characteristics, but these inherent characteristics are often discrete and sparse, such as prize, shape, color, and so on. Unfortunately, we hope that the characteristics of the items are continuous and dense vectors, which facilitates the RL algorithms better to learn the value function regarding state and characteristic.

To tackle the above issue, a common approach is training an embedding layer. We pre-train a click-through rate (CTR) prediction model DeepFM (Guo et al. 2017) using interaction records. It is worth noting that the interaction records will neither contain items from the training environment nor record items from the testing environment (Details in Appendix). Subsequently, we record the weights of the embedding layer in DeepFM. Finally, we **load** the weights to the embedding layer in AdaQN as the function f and **freeze** it.

The reason for this is that reinforcement learning involves the agent actively interacting with the environment, and this interaction can introduce instabilities into the network learning process. These instabilities not only hinder the algorithm from learning a good embedding, but also disrupt the previously well-learned embedding layer.

Algorithm 1: Adaptive Q-Network

Training process:

Require: Item set in the training environment I_{train}, the weights of pre-trained embedding layer.

Output: the weights of CEQN.

1: Initialize all trainable parameters in the state-characteristic value function CEQN, **load** and **freeze** the weights of the pre-trained embedding layer as function f.
2: Initialize replay buffer **B**.
3: Project item set I_{train} on the characteristic set V_{train} through f.
4: **for** each iteration **do**
5: Apply current behaviour policy π^b_{train} through Eq. (2), collect and store samples to **B**.
6: Sample mini-batch (s_t, v_t, r_t, s_{t+1}) from **B**.
7: Update CEQN according Eq. (3).
8: **end for**

Testing process:

Require: Item set in the test environment I_{test}, the weights of pre-trained embedding layer and weights of CEQN.

1: Project item set I_{test} on the characteristic space V_{test} through f.
2: Apply current policy π_{test} based on Eq. (1).

Characteristic-based Efficient Q-Network In standard continuous RL, the state-action pair (s, a) is concatenated to form a single vector, which serves as the input to a feedforward neural network for estimating the Q-values. However, this approach is not suitable for AdaQN for the reason that characteristics are often high-dimensional vectors. Directly feeding the concatenated state and characteristics into a neural network will result in a large number of parameters and lead to optimization difficulties.

We designed a Characteristic-based Efficient Q-Network (CEQN) to solve this problem. After aligning the dimensions of the state and characteristics, we replaced the concatenation operation with the Hadamard product and then passed the result through a neural network to obtain the Q-values:

$$CEQN(s,v) = m_q(m_s(s) \odot m_v(v)),$$

where m_s and m_v are functions that unify the dimensions of states and characteristics, respectively. $\odot$ represents the Hadamard product, which has been used to replace the previous concatenation operation. Meanwhile, the m_q function outputs the Q-value corresponding to the current state and characteristic.

By employing this method, we have significantly reduced the number of parameters and lowered the difficulty of optimization.

Policy Learning For the input interaction history sequence $x_{1:t} = \{x_1, x_2, \cdots, x_t\}$, a sequential model is used to encode this information into a state representation $s_t =$ **StateEnc**$(x_{1:t})$. Specific methods are described in Section

Environment module and state representation. Then, AdaQN loads and freezes the weights of the pre-trained embedding layer as function f. Therefore, AdaQN applies a behavior policy in the training phase:

$$\pi^{b}_{train}(s) = \arg\max_{i \in I_{train}} Q^{\pi_{train}}(s, v = f(i)). \quad (2)$$

Each complete interaction sequence is treated as an individual trajectory and is stored within a buffer. Thereafter, the target policy is trained using the trajectories that have been accumulated in it.

Parameterize the CEQN with ϕ, and update the network by minimizing the Mean Squared Error (MSE):

$$L(\phi) = \mathbb{E}_{\tau \sim \pi_{train}}[CEQN_{\phi}(s_t, v_t) - (r(s_t, v_t) + \gamma \max_{i \in I_{train}} CEQN_{\phi}(s_{t+1}, v = f(i))]^2, \quad (3)$$

where τ is the sampled trajectory. In particular, s_{t+1} and a_{t+1} come from the behavior policy π^{b}_{train} while s and a come from the target policy π_{train}.

During the test phase, we also incorporate an embedding layer as f into our model and apply policy through Eq. (1). The complete algorithmic process is presented in Algorithm 1.

Experiments

This section is structured as follows. Firstly, we introduce the experiment setting and the evaluation criteria. We then compare the performance of AdaQN for different recommendation tasks. Next, we present relevant ablation experiments for the dynamic recommendation task. Eventually, we provide a series of ablation studies on AdaQN in detail.

Experimental Settings

Datasets We conduct experiments on two representative datasets MovieLens[1] and KuaiRec[2] (Gao et al. 2022). For specific details, please refer to Appendix.

Environment module and state representation We construct the environment module based on the two public datasets mentioned above. As previous studies (Wang et al. 2023; Zhao et al. 2023), we implement the RL environment using the APIs of OpenAI Gymnasium (Brockman et al. 2016). Compared to simulation platforms, environments built from public datasets are much lighter and allow for faster training.

We primarily assess the dynamic recommendation task, and the ratio of items in the training environment to the testing environment is 0.5. A ratio of 0.5 ensures that the number of items in the training set is similar to that in the test set. Additionally, we provide settings for other ratios in ablation studies on task setting.

In recommendation scenarios, the state cannot be directly obtained from the environment and requires manual modeling. In most research works (Chen et al. 2022; Gao et al. 2023; Zhang et al. 2022), the state is encoded using a sequential model based on user characteristics and interaction history, which is used to capture the preferences of users and action sequence values at the current moment. We implemented various different state sequence models as follows:

- Average (Liu et al. 2020). Average combines the user embedding with the result of the average pooling of historical actions to serve as the state representation.
- GRU (Hidasi et al. 2015). GRU utilizes Recurrent Neural Networks (RNNs) to model user action sequences.
- SASRec (Kang and McAuley 2018). SASRec is a sequential model based on self-attention mechanisms that can balance short-term intentions with long-term preferences.
- NextItNet (Yuan et al. 2019). NextItNet is a generative model that can learn advanced representations from both short-term and long-term item dependencies.

In our work, the sequential model we primarily utilized is Average for its simplicity. Furthermore, we conducted a comprehensive ablation study on different state sequence models in Appendix for State sequence models.

Baselines We compare our method with various baselines, including Random, Supervised Learning (SL), DQN (Mnih et al. 2013), PPO (Schulman et al. 2017), PG(C) (Williams 1992), and HAC(C) (Liu et al. 2023). Specifically, (C) represents that this is an RL method based on a continuous action space, otherwise it is an RL algorithm based on a discrete action space. In addition, to handle the situation where the item sets in the training space and the test space are inconsistent, we will mask the test set items during the training phase, and vice versa.

Evaluation We follow the work (Yu et al. 2024) in order to better simulate real online user behavior. This work introduces a quitting mechanism and provides three evaluation modes: FreeB, NX_0_, NX_10_,

- FreeB: allow repeated recommendations, interactions are terminated by quit mechanism.
- NX_0_: prohibit repeated recommendations, interactions are terminated by quit mechanism.
- NX_10_: prohibit repeated recommendations, interactions are fixed as 10 rounds without quit mechanism.

To evaluate the long-term effects of RL policies, we consider the cumulative reward R to measure cumulative gain for one interaction trajectory, which is usually applied in the RL scenario.

For all tasks, the goal of the recommender policy is to maximize the long-term satisfaction represented by total reward. For the training stage, all policies are trained with 100 epochs. The policy is evaluated using 100 interaction trajectories after each epoch, and the maximum recommended sequence length is limited to 30. Following (Yu et al. 2024), we report the mean values of all metrics during the last 25% of training epochs to achieve a fair comparison.

To define the reward setting, we initially consider single-step rewards that can be obtained from static datasets, such as clicks, ratings, etc. However, the algorithm's ultimate goal

[1]https://grouplens.org/datasets/movielens/1m/

[2]https://kuairec.com/

Method	KuaiRec			MovieLens		
	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$
Random	8.1875	8.8605	8.8829	11.1909	11.2373	26.9256
DQN	4.5886	7.9822	9.3379	6.5533	7.9897	24.4507
PPO	8.1536	8.2306	8.4924	11.3272	11.2633	26.8535
PG(C)	9.4740	8.5514	9.7703	7.1760	8.3019	32.3740
HAC(C)	6.7015	11.1578	11.3377	4.1933	4.6435	8.1406
AdaQN	**13.1100**	**14.4005**	**13.6655**	**13.0869**	**13.0905**	**32.5607**

Table 1: Evaluation results on dynamic recommendation task.

Method	KuaiRec			MovieLens		
	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$
Random	8.5041	8.1472	8.5309	10.1579	10.2689	29.3595
SL	7.4569	15.2410	**20.1416**	8.6129	9.4193	**40.5054**
DQN	20.8960	14.1351	10.1075	22.0477	23.9807	28.9467
PPO	19.1449	18.0902	5.5861	**34.0812**	**27.7432**	26.9448
PG(C)	30.9474	11.7333	11.1364	18.1728	26.4361	30.9596
HAC(C)	24.1329	22.1748	17.6489	26.5643	19.9467	31.7904
AdaQN	**31.5592**	**31.8439**	17.9038	25.4937	27.1657	33.5826

Table 2: Evaluation results on standard recommendation task.

is to maximize cumulative reward, which requires considering the long-term engagement of users. Specifically, the system must account for the fact that users may lose interest if exposed to too much similar content. Therefore, algorithms are unlikely to achieve optimal recommendation outcomes for adopting a short-sighted, greedy strategy without considering long-term user preferences.

Performance

In order to evaluate the effect of AdaQN on performance, we evaluate different basic algorithms on both dynamic recommendation and standard recommendation tasks.

Performance on dynamic recommendation task We compare AdaQN with a series of baselines on dynamic recommendation tasks. According to Table 1, the poor performance of DQN is primarily attributed to its inability to effectively handle new items. In other words, items must be sufficiently trained for DQN to recommend them effectively. The PPO algorithm based on a discrete action space also views items in isolation, failing to effectively accomplish the dynamic recommendation task. HAC(C) completes the task by training a hyperactor. However, the trained hyperactor cannot interact directly with newly emerging items, leading to the inability of HAC(C) to adapt to the dynamic item space. In comparison with these methods, AdaQN has effectively learned the relationships between items through stable embedding. Furthermore, AdaQN, through its dynamic Q-Network, can adapt well to the changing item space and effectively recommend newly emerging items.

Performance on standard recommendation task We also compare AdaQN with the baselines on standard recommendation tasks, as shown in Table 2. It is worth mentioning that the supervised learning method performs well only under the NX_10_ protocol, as the long-term optimal under this protocol is equivalent to the greedy stepwise optimal. However, in real-world recommendation environments, an ideal recommendation algorithm needs to suggest a diverse range of content to prevent users from becoming bored with overly similar recommendations. In this aspect, SL falls short compared to rl-based algorithms. In rl-based algorithms, the performance of AdaQN is comprehensively superior to that of DQN, stemming from that AdaQN does not directly learn the recommended items but instead learns the characteristics of the recommendations. This greatly mitigates the curse of dimensionality caused by an excessive number of items. Otherwise, AdaQN performs comparably to other methods, indicating that AdaQN can also achieve satisfactory results in standard recommendation tasks.

Ablation Studies on Task Setting

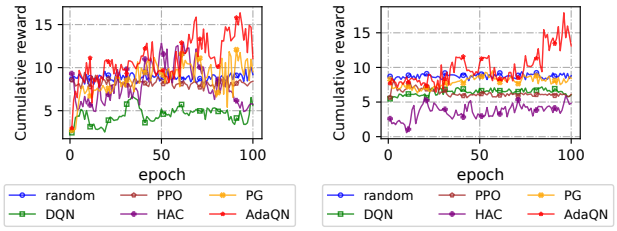
In this section, we study the effects of items' partial overlap and different proportions of training items on the KuaiRec environment.

Partial overlap of items We make an interesting adjustment to the environment that set a 20% overlap rate for the items between the training set and the test set. We have established a new metric: overlap item ratio. It is defined as the ratio of recommended items that overlap between the test set and the training set for all recommended items. The objective of this index is to observe whether the algorithm tends to recommend items that have appeared in the training set during the test, so as to evaluate the willingness of the algorithm to recommend new items.

Method	R_{FreeB}	Overlap item ratio
Random	8.1849	33.2%
DQN	6.8732	99.7%
PPO	7.6194	100%
PG(C)	7.8166	2.7%
HAC(C)	10.8161	15.5%
AdaQN	**14.1107**	32.8%

Table 3: Ablation result for partial overlap between the training set and the test set.

According to Table 3, we observed different performances of RL based on discrete and continuous action spaces. Lacking exploration of new items, algorithms based on discrete action spaces will hardly recommend new items. In contrast, algorithms based on continuous action spaces are willing to recommend new items. However, the algorithms disregard the long-term returns of new items to the environment as they rank items based on action vectors and item embedding, leading to unsatisfactory recommendation outcomes. Surprisingly, the overlap items ratio of AdaQN is closest to that of the random algorithm, indicating that AdaQN can recommend all items in the test set equally. Furthermore, AdaQN can consider the long-term impact of new items on the environment, achieving better recommendation results.



(a) KuaiRec (b) MovieLens

Figure 2: The training curve of cumulative rewards under the FreeB evaluation mode.

Ablation Studies on Algorithm

To further verify the effectiveness of our proposed algorithm, we conducted a series of ablation experiments in the KuaiRec environment with a ratio set to 0.5.

Visualization From Fig. 2, it can be observed that the training curves of most methods are essentially parallel to the x-axis, indicating that they fail to learn from the training environment. Different from these methods, the performance of AdaQN gradually increases. This implies that AdaQN not only effectively learns the value distribution of states and item characteristics but also captures the attributes of newly emerging items. This enables AdaQN with a strong capability to adapt to dynamic item spaces.

Mapping function f In this subsection, we discussed the impact of different mapping functions on the performance. Through Table 4, we discovered that an embedding layer without pre-training or a pre-trained but unfrozen embedding layer fails to yield satisfactory outcomes. We attribute this to the instability of environmental supervision. This leads to the algorithm's inability to train a high-quality embedding layer and can even degrade the performance of a pre-trained embedding layer. Conversely, employing a pre-trained and frozen embedding layer effectively reserves the desired performance.

Method	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$
Without pre-train	8.0821	9.8770	11.5704
Pre-trained (unfroze)	7.1675	8.2758	8.7405
Pre-trained (freeze)	**13.1100**	**14.4005**	**13.6655**

Table 4: Ablation results for mapping function f.

Characteristic-based Efficient Q-Network To validate the effectiveness of the designed CEQN, we compared it with the standard approaches in continuous RL. According to Table 5, CEQN not only has fewer parameters than the standard continuous RL methods but also achieves better results. This is attributed to the direct concatenation of states and characteristics, which leads to dimensional redundancy, causing optimization difficulties. On the contrary, our method is more adaptable to AdaQN.

Method	Parameters	R_{FreeB}	$R_{NX\_0\_}$	$R_{NX\_10\_}$
Concat	87637	12.6659	14.0898	11.4686
CEQN	**19828**	**13.1100**	**14.4005**	**13.6655**

Table 5: Ablation results for CEQN.

Conclusion and Future Work

This article first proposes a novel dynamic recommendation task to meet the needs of the constantly changing pool of recommendable items in real-world scenarios. To overcome this challenge, we propose the Adaptive Q-Network (AdaQN). The AdaQN learns a function that models the characteristics to predict the value of new items, facilitating recommendations that satisfy user preferences. Subsequently, we establish a mapping that projects the item space onto characteristic space in order to learn a stable value function. Finally, the Characteristic-based Efficient Q-Network (CEQN), an elegantly lightweight Q-function, is designed to integrate with the AdaQN, enhancing the performance of AdaQN.

Moreover, our method has the potential to be extended to cross-domain recommendations. In our future work, we will apply AdaQN to cross-domain recommendations by learning cross-domain mapping. It is our aspiration that the outcomes of our research will significantly bolster the advancement of RS.

Acknowledgments

This work is supported by the National Science Foundation of China (12471501), and the Sci-Tech Innovation Initiative

by the Science and Technology Commission of Shanghai Municipality (24ZR1419000).

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540*.

Cai, Q.; Liu, S.; Wang, X.; Zuo, T.; Xie, W.; Yang, B.; Zheng, D.; Jiang, P.; and Gai, K. 2023. Reinforcing user retention in a billion scale short video recommender system. In *Companion Proceedings of the ACM Web Conference 2023*, 421–426.

Chen, M.; Beutel, A.; Covington, P.; Jain, S.; Belletti, F.; and Chi, E. H. 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*, 456–464.

Chen, M.; Xu, C.; Gatto, V.; Jain, D.; Kumar, A.; and Chi, E. 2022. Off-policy actor-critic for recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, 338–349.

Gao, C.; Li, S.; Lei, W.; Chen, J.; Li, B.; Jiang, P.; He, X.; Mao, J.; and Chua, T.-S. 2022. Kuairec: A fully-observed dataset and insights for evaluating recommender systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 540–550.

Gao, C.; Wang, S.; Li, S.; Chen, J.; He, X.; Lei, W.; Li, B.; Zhang, Y.; and Jiang, P. 2023. CIRS: Bursting filter bubbles by counterfactual interactive recommender system. *ACM Transactions on Information Systems*, 42: 1–27.

Ge, Y.; Liu, S.; Gao, R.; Xian, Y.; Li, Y.; Zhao, X.; Pei, C.; Sun, F.; Ge, J.; Ou, W.; et al. 2021. Towards long-term fairness in recommendation. In *Proceedings of the 14th ACM international conference on web search and data mining*, 445–453.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247*.

Hansen, C.; Hansen, C.; Maystre, L.; Mehrotra, R.; Brost, B.; Tomasi, F.; and Lalmas, M. 2020. Contextual and sequential user embeddings for large-scale music recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*, 53–62.

Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; and Tikk, D. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Huang, L.; Fu, M.; Li, F.; Qu, H.; Liu, Y.; and Chen, W. 2021. A deep reinforcement learning based long-term recommender system. *Knowledge-Based Systems*, 213: 106706.

Jiang, Y.; Li, Q.; Zhu, H.; Yu, J.; Li, J.; Xu, Z.; Dong, H.; and Zheng, B. 2022. Adaptive domain interest network for multi-domain recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 3212–3221.

Kang, W.-C.; and McAuley, J. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, 197–206. IEEE.

Lee, H.; Hwang, D.; Min, K.; and Choo, J. 2022. Towards validating long-term user feedbacks in interactive recommendation systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2607–2611.

Liu, F.; Tang, R.; Li, X.; Zhang, W.; Ye, Y.; Chen, H.; Guo, H.; Zhang, Y.; and He, X. 2020. State representation modeling for deep reinforcement learning based recommendation. *Knowledge-Based Systems*, 205: 106170.

Liu, S.; Cai, Q.; Sun, B.; Wang, Y.; Jiang, J.; Zheng, D.; Jiang, P.; Gai, K.; Zhao, X.; and Zhang, Y. 2023. Exploration and regularization of the latent action space in recommendation. In *Proceedings of the ACM Web Conference 2023*, 833–844.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Qin, J.; Ren, K.; Fang, Y.; Zhang, W.; and Yu, Y. 2020. Sequential recommendation with dual side neighbor-based collaborative relation modeling. In *Proceedings of the 13th international conference on web search and data mining*, 465–473.

Rohde, D.; Bonner, S.; Dunlop, T.; Vasile, F.; and Karatzoglou, A. 2018. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Smith, B.; and Linden, G. 2017. Two decades of recommender systems at Amazon.com. *IEEE Internet Computing*, 21: 12–18.

Wagenmaker, A.; and Pacchiano, A. 2023. Leveraging offline data in online reinforcement learning. In *International Conference on Machine Learning*, 35300–35338. PMLR.

Wang, K.; Zou, Z.; Zhao, M.; Deng, Q.; Shang, Y.; Liang, Y.; Wu, R.; Shen, X.; Lyu, T.; and Fan, C. 2023. RL4RS: A Real-World Dataset for Reinforcement Learning based Recommender System. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2935–2944.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8: 229–256.

Yu, Y.; Gao, C.; Chen, J.; Tang, H.; Sun, Y.; Chen, Q.; Ma, W.; and Zhang, M. 2024. EasyRL4Rec: A User-Friendly Code Library for Reinforcement Learning Based Recommender Systems. *arXiv preprint arXiv:2402.15164*.

Yu, Z.; and Zhang, X. 2023. Actor-critic alignment for offline-to-online reinforcement learning. In *International Conference on Machine Learning*, 40452–40474. PMLR.

Yuan, F.; Karatzoglou, A.; Arapakis, I.; Jose, J. M.; and He, X. 2019. A simple convolutional generative network for next item recommendation. In *Proceedings of the twelfth ACM international conference on web search and data mining*, 582–590.

Zhang, Q.; Liu, J.; Dai, Y.; Qi, Y.; Yuan, Y.; Zheng, K.; Huang, F.; and Tan, X. 2022. Multi-task fusion via reinforcement learning for long-term user satisfaction in recommender systems. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 4510–4520.

Zhao, K.; Liu, S.; Cai, Q.; Zhao, X.; Liu, Z.; Zheng, D.; Jiang, P.; and Gai, K. 2023. KuaiSim: A comprehensive simulator for recommender systems. In *Advances in Neural Information Processing Systems*, volume 36, 44880–44897.

Zhao, X.; Xia, L.; Zhang, L.; Ding, Z.; Yin, D.; and Tang, J. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM conference on recommender systems*, 95–103.

Zheng, G.; Zhang, F.; Zheng, Z.; Xiang, Y.; Yuan, N. J.; Xie, X.; and Li, Z. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 world wide web conference*, 167–176.