Offline reinforcement learning with delay rewards

xxx
Nanjing University
xxx

xxx
Kuaishou Corp.
xxx

Abstract

blablabla

1 Introduction

Offline reinforcement learning has received lots of attentions in recent years [**levine2020offline**], since it tries to learn policies with pre-collected data, which is especially attractive in real-world applications such as robotics, recommender systems etc. Among these tasks, an important but less studied setting is offline reinforcement learning with delay sparse rewards, which is quite common in some applications. For example in recommender systems, the company often focus on long-term engagement of users such as daily active number or retention rate, instead of instant indicators like click or watch time. However, this kind of long-term indicators are sparse and delayed, since they can be only observed once a day after interactions between system and users in previous time. This kind of delayed and sparse signals are hard to be optimized directly, especially in more restricted and hard offline setting, so current recommender systems try to improve user engagement through short term response, which may not be optimal. In this paper, we focus on how to tackle offline reinforcement learning with delayed sparse rewards.

However, most of existing algorithms designed for offline RL rely on dense rewards [XXX].

2 Related work

3 Preliminaries

3.1 Offline reinforcement learning

4 Method

In offline scenarios, trial-and-error and exploration of the environment becomes impossible, and delaying the reward signal in most cases introduces the problem of reward sparsity, which makes the strategy optimization algorithm that optimizes the expected reward from offline data invalid. Our goal is to find the optimal policy $\pi^* = \arg\max_\pi \mathbb{E}_{\tau \in D}\left[R_\tau\right]$ that maximizes the cumulative reward from this offline data set of delayed rewards.

In this section, we present a general framework for addressing sparse reward problems in offline setting. From a global perspective, the framework consists of two-stage learning tasks, in the first stage we perform specific modification strategy on the offline datasets, and the second stage is a regular offline reinforcement learning task.

Reward modification. Our starting point is to find an appropriate conversion function $\mathcal{F} : S \times A \times R \to R$ of the original delayed rewards, such modification restores a dense reward as much as possible so that helps the offline policy learning phase afterwards. Such conversion only occurs at the offline data level, and any offline policy learning algorithm can be seamlessly combined with it in the subsequent strategy learning phase.

Offline policy optimization.

5 Experiments

In this section, we conducted corresponding experiments in the D4RL public datasets and the real-world recommendation system scenarios. Our primary points of comparison are reward modification strategies include: scale, return min-max, average, ensemble. Furthermore, these strategies can be combined with any offline RL algorithms without extra efforts, model-free offline RL algorithms BC, CQL and model-based algorithm MOPO are included depend on the environment as follows:

- Behaviour Cloning(BC). BC uses supervised learning for policy learning, and the learning process does not depend on the rewards. Therefore, delaying the rewards has no effect on its policy learning process. In this part, we directly quote the results published in the D4RL paper.
- Conservative Q-Learning(CQL). CQL is a type of model-free algorithm. It learns a Q-value function directly from the data, thereby avoiding action out of distribution(OOD) caused by offline reinforcement learning.
- Model-based Offline Policy Optimization(MOPO). MOPO is a model-based algorithm. It first learns multiple supervised learning models (including state transition and reward function) from offline data, and the strategy interacts with the learned model. The reward adds an additional transition to the joint model. Deterministic estimation, as an empirical reward Lower Bound, has verified its effectiveness in theory and experiment.

we verify the effectiveness of aforementioned methods in solving the delayed rewards problems. Experiments conducted both on simuated offline datasets (OpenAI Gym) and customized real-world datasets. All tasks involves delayed rewards and high-dimensional observation spaces, xxx.

5.1 Evaluation on D4RL public datasets

In this section, we compare to BC, CQL, and MOPO on three continuous control tasks from the D4RL bechmark. As the original datasets We first construct the delay rewards datasets from the original none-delayed rewards datasets on 4 levels setting:

1. Random: 1 million timesteps generated by a random policy.
2. Medium: 1 million timesteps generated by a medium policy that achieves approximately one-third the score of an expert policy.
3. Medium-Replay: the replay buffer of an agent trained to the performance of a medium policy (approximately 25k-400k timesteps in our environments).
4. Medium-Expert: 1 million timesteps generated by the medium policy concatenated with 1 million timesteps generated by an expert policy.

Datasets construction. In order to make a public comparison with other offline algorithms, we delay the dense reward based on the D4RL gym datasets. Specifically, we perform constant delay setting: we set a hyper-parameter K as the delay interval which controls the sparsity of the delayed rewards. We set the delayed reward at equal intervals with K as the un-discounted cumulative rewards of corresponding interval, the missing reward filled with 0, formally:

For a trajectory τ with length T, we set the delayed reward of time step t as r_t^{delay}, where:

$$r_t^{delay} = \begin{cases} 0, \text{if } t \mod K \neq 0; \\ \sum_{i=t-K+1}^{t} r_t, \text{otherwise.} \end{cases}$$

Such construction strategy introduce sparsity to delay rewards, meanwhile, it keeps the rule that for any trajectory in the original datasets, the un-discounted cumulative rewards keep constant before and after the operation, mathematically, we have:

$$\sum_{t=1}^{T} r_t = \sum_{t=1}^{T} r_t^{delay}$$

To facilitate intuitive and clear comparison of the delay reward:

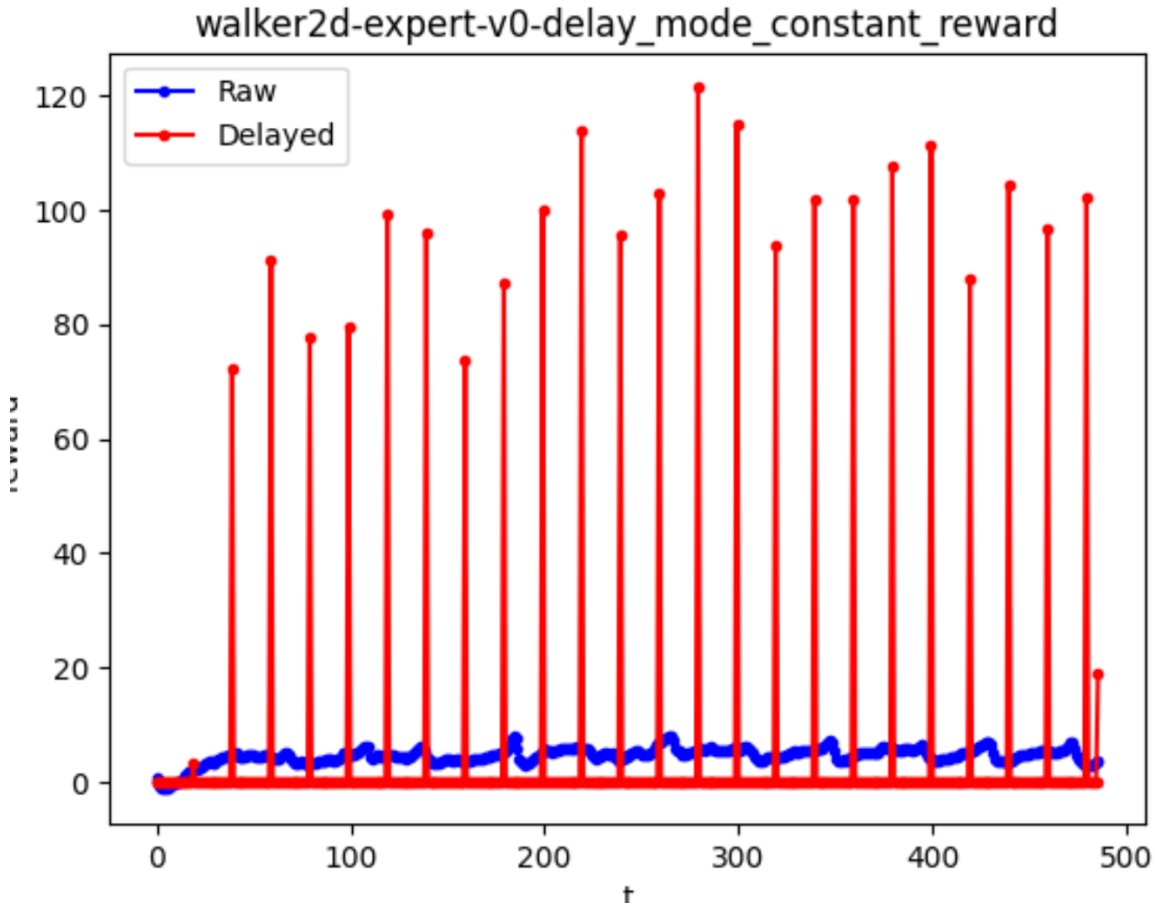

Figure 1: Delayed rewards xxx

We compare to BC, CQL, and MOPO. BC numbers are reported from the original D4RL paper, CQL and MOPO results are run by us. Our results are shown in Table 1. MOPO(average) achieves the highest scores in 11/12 of the tasks.

5.2 Evaluation on recommender system simulated datasets

In order to verify the effectiveness of the reward modification strategy proposed in the article, we added a simulation environment in a real-world scenario for verification.

6 Conclusions

In this paper, we propose a xxx.

Dataset	Environment	Delay	BC	CQL	MOPO		
					Normal	+Average	+Ensemble
Random	HalfCheetah	20	2.1	-	0.1	$\mathbf{25.3 \pm 1.5}$	
Random	Hopper	20	1.6	-	3.1	$\mathbf{9.1 \pm 1.5}$	
Random	Walker2d	20	9.8	-	0.1	$\mathbf{12.6 \pm 6.1}$	
Medium	HalfCheetah	20	36.1	-	-0.8	$\mathbf{48.3 \pm 2.4}$	
Medium	Hopper	20	29	-	5.1	19.0 ± 8.1	
Medium	Walker2d	20	6.6	-	4.5	$\mathbf{72.3 \pm 0.9}$	
Medium-Replay	HalfCheetah	20	38.4	-	1.9	54.5 ± 3.2	
Medium-Replay	Hopper	20	11.8	-	1.9	89.8 ± 5.7	
Medium-Replay	Walker2d	20	11.3	-	0.9	$\mathbf{59.1 \pm 3.7}$	
Medium-Expert	HalfCheetah	20	35.8	-	2.1	$\mathbf{73.6 \pm 8.3}$	
Medium-Expert	Hopper	20	111.9	-	9.4	$\mathbf{22.1 \pm 3.2}$	
Medium-Expert	Walker2d	20	6.4	-	6.8	$\mathbf{99.0 \pm 3.7}$	

Table 1: Results for D4RL datasets[2]. We report the mean and variance for three seeds. MOPO+Average outperforms conventional RL algorithms on almost all tasks.

Appendix

A First Part