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 [6], 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 this section, we present a general framework for addressing delay rewards or sparse rewards problems in offline setting. From a global perspective, the framework consists of two-stage learning tasks, in the first stage we perform reward modification strategy on the offline datasets, and then learn the offline policy from the modified datasets in the latter stage.

4.1 Reward modification

Our starting point is to find an appropriate conversion function 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.

Next, we will introduce 3 kinds of reward conversion strategies:

1. **Reward Decomposition.** Works in this area [1] consider establishing a decomposition strategy to convert the original delay or sparse rewards into dense rewards under constraints. We can train a

parameterized reward network $f_\theta(s, a)$ from the original delay rewards (sparse) data obeying that for a trajectory $\tau = (s_0, a_0, r_0, \cdots, s_T, a_T, r_T)$ sampled from offline datasets $\mathcal{D}$, the decomposition operation keeps the decomposed reward $r'_t = f_\theta(s_t, a_t)$ and the original reward r to be consistent on the trajectory level, that is: $\sum_{t=0}^{T} r_t = \sum_{t=0}^{T} r'_t$.

2. **Reward Shaping.** Reward shaping is well studied as it maintains the invariance of policy before and after the reward transformations. We follow the prior work [7] by adding an extra bounded real-valued shaping function $F : \mathcal{S} \times \mathcal{A} \times S \rightarrow \mathbb{R}$ to the raw reward function R, in practice, we parameterize this shaping reward function by differentiable neural network and train it with Policy Gradient methods [8].

3. **Reward Smoothing.** Delay rewards deviate greatly from the immediate reward in scale and space, smoothing is the strategy that attemps to re-distribute the single step delayed reward r_t^{delay} to multi-step immediate rewards $r_{t-k:t}$ satisfying the constraints that it keeps the consistency of the reward sum in the interval $[t-k:t]$: $\sum_{i=t-k}^{t} r_t = r_t^{delay}$. Specifically, we consider the following smoothing transformations:

    (a) **Minmax strategy**.

    (b) **Average strategy**.

    (c) **Ensemble strategy**.

4.2 Offline policy learning

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 conventional optimization algorithm that maximizing the expected rewards from offline data no longer applicable. Our goal is to find the optimal policy $\pi^* = \arg\max_\pi \mathbb{E}_{\tau \in \mathcal{D}} [R_\tau]$ that maximizes the cumulative reward from this offline datasets with delayed rewards.

5 Experiments

In this section, we conduct corresponding experiments on the D4RL benchmark [2] and the real-world recommendation system scenarios datasets based on RecSim [4]. 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 [5] and model-based algorithm MOPO [10] 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 tries to learn a "Conservative" Q-value function by incorporating random samples from state-action distributions, thereby avoiding action out of distribution(OOD) caused by offline Q-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 high-dimensional observation spaces, the delayed rewards introduces extra difficulty for learning effective control strategies in offline setting.

5.1 Evaluation on D4RL public datasets

We evaluate our methods together with prior offline RL algorithms on three continuous control tasks from the D4RL bechmark. As the rewards in original datasets are not delayed, we first construct the delay rewards datasets from the none-delayed rewards datasets.

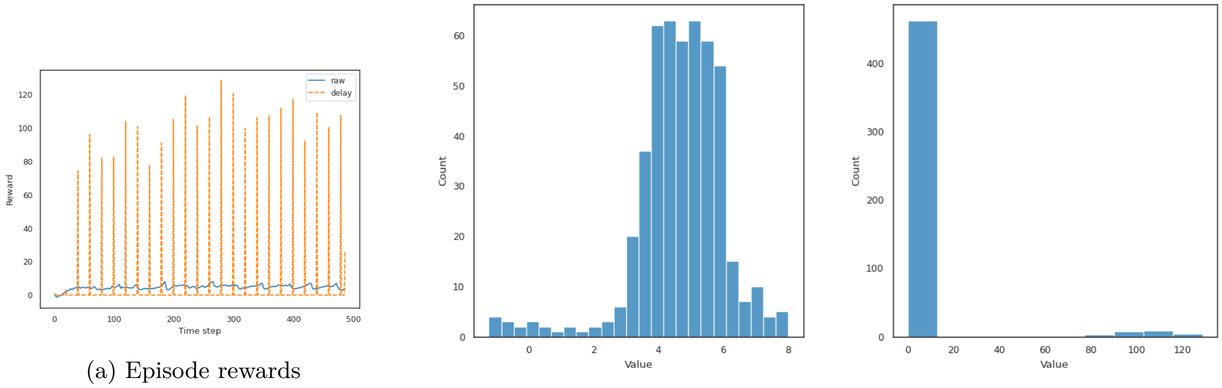
Datasets construction. Without dedicating extra efforts collecting data by training an agent from scratch, we directly delay the original rewards by performing the constant delay transformation: we first set a hyper-parameter K as the delay interval which controls the sparsity of the delayed rewards, the delayed rewards are then computed at equal intervals with K as the un-discounted cumulative rewards of corresponding interval, the missing rewards are filled with 0. Formally, for a trajectory $\tau = (s_0, a_0, r_0, \cdots, s_T, a_t, r_T)$, the delayed reward at time step t is defined as r_t^{delay}, where:

$$r_t^{delay} = \begin{cases} 0, & (t+1)\,\%K \neq 0; \\ \sum_{i=t-K+1}^{t} r_i, & \text{otherwise.} \end{cases}$$

The transformation above introduces 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:

$$\sum_{t=0}^{T} r_t = \sum_{t=0}^{T} r_t^{delay}$$

The comparison between the immediate rewards and delayed rewards are shown in Figure 1, the immediate rewards are more continuous in scale and dense in spatiality.



(a) Episode rewards

(b) Raw rewards distribution

(c) Delay rewards distribution

Figure 1: Illustrations of immediate rewards and delayed rewards on Walker2d-expert-v0 with delay interval 20. **Left:** The lineplot of immediate rewards and delayed rewards for a trajectory , the blue line denoted as *raw* is the raw immediate rewards, the delayed rewards are drawn in orange. **Center:** The distribution of the immediate rewards ranges in $[-2, 10]$. **Right:** The sparse distribution for delayed rewards, In all data samples, about 95% of the samples get reward 0, and the remaining data contains non-zero reward signals and these non-zero rewards are relatively large in value and distributed very scattered.

We compare to BC, CQL, and MOPO on Gym domain tasks. The performance of BC has nothing to do with the setting of delayed rewards, as it directly learns the policy that maps from observation space to action space, we report the numbers of BC from the original D4RL paper. The policy learning process of CQL and MOPO are strictly bound to the rewards, we run these algorithms based on the open source reproducing code base **OfflineRL** published by Qin et al. on NeoRL paper. Our results are shown in Table 1.

Delay interval size. Among above tasks, we set the delay interval to constant 20 and our approaches performs quite well under this setting. To investigate the impact of different delay interval size on policy

					MOPO			
Dataset	**Environment**	**Delay**	**BC**	**CQL**	**Normal**	**+Average**	**+Ensemble**	**+Minmax**
Random	HalfCheetah	20	2.1	-	0.1	**25.3 ± 1.5**		
Random	Hopper	20	1.6	-	3.1	**9.1 ± 1.5**		
Random	Walker2d	20	9.8	-	0.1	**12.6 ± 6.1**		
Medium	HalfCheetah	20	36.1	-	-0.8	**48.3 ± 2.4**		
Medium	Hopper	20	29	-	5.1	19.0 ± 8.1		
Medium	Walker2d	20	6.6	-	4.5	**72.3 ± 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	**59.1 ± 3.7**		
Medium-Expert	HalfCheetah	20	35.8	-	2.1	**73.6 ± 8.3**		
Medium-Expert	Hopper	20	111.9	-	9.4	22.1 ± 3.2		
Medium-Expert	Walker2d	20	6.4	-	6.8	**99.0 ± 3.7**		

Table 1: Results for D4RL datasets on the normalized return metric. We report the mean and variance over 3 random seeds. Among all tasks, MOPO+Average and MOPO+Ensemble outperforms BC algorithms on 11/12 tasks, CQL and MOPO algorithms basically fail to learn effective policies under delayed rewards.

perfomance, we conduct corresponding experiments varing the delay from small to large. In one case, the delay interval is set to be 1, which is the smallest delay known as immediate rewards, which is the scenario normal OfflineRL algorithms can be applied, while the other, delay is set to the maximum value that is equal to the length trajectory, the reward only appears in the last time step of the trajectory. Unsurprisingly, increasing the delay interval size can diminish the performance of policy, the relation between different delay interval size and policy performance is shown in Figure 2.

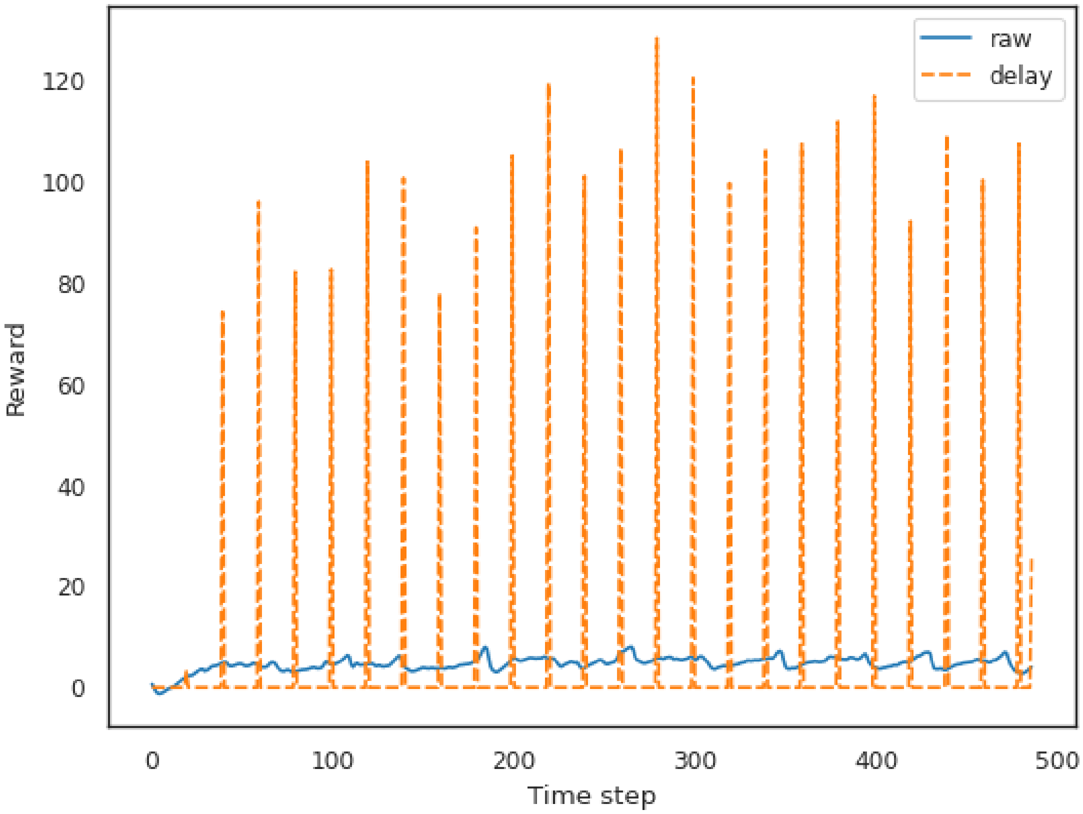


Figure 2: Relations between the size of delay interval and policy performance.

5.2 Evaluation on custom simulated datasets

In order to verify the effectiveness of the reward modification strategy proposed before, we added an artifical task on simulatied environment in a real-world scenario for verification.

Recommender systems is one of the important scenarios of reinforcement learning applications. In modern recommender systems, the interaction between the users and the recommender system can be abstracted into the interaction between the recommendation agent and the user environment. In the recommender system, it is very difficult to optimize the long-term retention of users, as retention signal is the 0-1 binary signal obtained at the final moment of a long-term interaction, which is very sparse especially for inactive users. Such retention signals and delay rewards are consistent in definition, therefore, in our experiments, out of user privacy and security requirements, we build a user retention simulation environment based on Googles open source reinforcement learning recommendation system framework RecSim [4].

Datasets for experiments are collected by training an agent interacting with the simulated environment using Soft Actor-Critic [3] from scratch and collecting 1000 trajectories at 4 different levels during training to make the data distribution more complex: More details on the construction of simulated environment and data collection are available in Appendix B.2.

6 Conclusions

In this paper, we propose a xxx.

References

[1] Jose A. Arjona-Medina et al. "RUDDER: Return Decomposition for Delayed Rewards". In: *arXiv:1806.07857 [cs, math, stat]* (Sept. 2019).

[2] Justin Fu et al. "D4RL: Datasets for Deep Data-Driven Reinforcement Learning". In: *arXiv:2004.07219 [cs, stat]* (Feb. 2021).

[3] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: `1801.01290 [cs.LG]`.

[4] Eugene Ie et al. "RecSim: A Configurable Simulation Platform for Recommender Systems". In: *arXiv:1909.04847 [cs, stat]* (Sept. 2019).

[5] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *arXiv:2006.04779 [cs, stat]* (Aug. 2020).

[6] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[7] Andrew Y. Ng, Daishi Harada, and Stuart Russell. "Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping". In: *In Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann, 1999, pp. 278–287.

[8] J. Peters. "Policy gradient methods". In: *Scholarpedia* 5.11 (2010). revision #137199, p. 3698. DOI: `10.4249/scholarpedia.3698`.

[9] Rongjun Qin et al. "NeoRL: A Near Real-World Benchmark for Offline Reinforcement Learning". In: *arXiv:2102.00714 [cs]* (Feb. 2021).

[10] Tianhe Yu et al. "MOPO: Model-Based Offline Policy Optimization". In: *arXiv:2005.13239 [cs, stat]* (Sept. 2020).

Appendix

A Reward Modification Implementation Details

XXXX

B Experiment Details

Our implementation is based on the OfflineRL library released by NeoRL team. For all experiments, we used default hyperparameter settings and minimal modifications to public implementations wherever possible, with different training iterations or gradient steps depending on the algorithm. We ran our experiments using single 1080 GPU machines.

B.1 D4RL benchmark

XXX

B.2 Custom simulated environment

We build our simulated environment on the RecSim framework for its flexibility and scalability, RecSim is published by google research team to build a configurable platform for building simulated environments for recommender systems (RSs) that naturally supports sequential interaction with users.

Datasets are collected by pretraining an agent in Soft Actor-Critic way from scratch and collecting 1000 trajectories at 4 different levels:

- Random:
- Low:
- Medium:
- Expert: